DISS. ETH NO. 24392

Deep Neural Networks and Hardware Systems for Event-driven Data

A thesis submitted to attain the degree of

DOCTOR OF SCIENCES OF ETH ZURICH

(Dr. sc. ETH Zurich)

presented by

DANIEL NEIL

M.Sc., University of Zürich and ETH Zürich 2013
B.S. Stanford University 2008

born on *15.01.1986*

citizen of the United States of America

accepted on the recommendation of

PD Dr. Shih-Chii Liu
Prof. Tobi Delbruck
Prof. Daniel Lee
Prof. Kevan A.C. Martin
2017

Deep Neural Networks and Hardware Systems for Event-driven Data

A DOCTORAL THESIS for ETH Zürich covering developments on event-based sensors, deep neural networks, and machine learning for bio-inspired applications.

BY **Daniel Neil**

First printing, July 2017

This work can only exist because of the family of students at the Institute of Neuroinformatics in Zürich, my inspiring flatmates Saee Paliwal and Hazael Montanaro, the brilliant Dane Corneil, the perennial encouragement of my advisors Shih-Chii Liu and Tobi Delbruck, the brilliant guidance of Michael Pfeiffer, all of my coauthors and colleagues who profoundly enriched the experience, and the warm encouragement of my family and all of my friends.

Contents

Acronyms

ANN Analog Neural Network. 17, 59, 62, 64, 65, 75, 76, 77, 78, 79, 80, 81, 82

API Application programming interface. 53

ASIC Application-Specific Integrated Circuit. 39, 53, 133

BN Batch-normalized. 18, 93, 94, 96

BRAM Block Random Access Memory. 42, 46

CD Contrastive Divergence. 60

CNN Convolutional Neural Network. 17, 20, 27, 61, 62, 63, 64, 66, 67, 68, 69, 70, 71, 70, 71, 72, 96, 124, 128, 129, 128, 129, 130

CPU Central Processing Unit. 37, 38, 39, 47, 49, 53, 57, 61

DAS Dynamic Audio Sensor. 31

DAVIS Dynamic and Active-pixel Vision Sensor. 15, 29

DBN Deep Belief Network. 16, 37, 38, 39, 40, 43, 45, 52, 53, 54, 56, 59, 60, 61, 62

DN Delta Network. 19, 20, 125, 126, 128

DSP Digital Signal Processor. 15, 42, 44, 45

DVS Dynamic Vision Sensor. 15, 29, 32, 43, 134

FPGA Field-programmable Gate Array. 37, 38, 39, 40, 42, 46, 53

FPS Frames Per Second. 128

GHz Gigahertz. 47

GPU Graphical Processing Unit. 38, 39, 61, 124, 153

GRU Gated Recurrent Unit. 20, 87, 99, 101, 114, 117, 120, 121, 122, 123, 125, 126, 127, 128, 129, 128, 129, 154

IF Integrate-and-Fire. 65, 66, 68, 69, 70, 76

IOE Integral of Error. 19, 107, 108

ISI inter-spike interval. 31

JIT Just-in-time. 135

LIF Leaky Integrate-and-Fire. 38, 40, 47, 52, 53, 59, 62

LSTM Long Short-Term Memory. 18, 19, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 108, 109, 110, 111, 112, 113, 114, 117, 134, 135

MFCC Mel-Frequency Cepstral Coefficients. 19, 97, 96, 98, 118, 125

MHz Megahertz. 42, 46, 47

NLP Natural Language Processing. 111

PLL Phase-locked loops. 46

PLSTM Phased LSTM. 19, 103, 104, 105, 106, 108, 109, 110, 111, 112, 113, 134, 136, 153

PSC Post-synaptic currents. 43, 47, 46, 47, 49, 50

RAM Random Access Memory. 15, 44

RBM Restricted Boltzmann Machine. 16, 39, 51, 60

RC Resistor-Capacitor. 40

ReLU Rectified Linear Unit. 62, 64, 65, 66, 67, 68, 76, 78, 96, 102

RNN Recurrent Neural Network. 19, 20, 27, 87, 88, 90, 93, 96, 97, 101, 103, 108, 117, 118, 117, 124, 127, 128, 129, 128, 129, 130, 131, 134, 154

ROM Read-only Memory. 15, 44, 45

SAE Stacked Auto-Encoder. 79, 81, 82, 83

SNN Spiking Neural Network. 17, 62, 63, 64, 65, 69, 71, 72, 75, 76, 77, 78, 79, 80, 81, 82, 84, 85

TF-IDF Term frequency-inverse document frequency. 49

USB Universal Serial Bus. 15, 29, 41, 44, 47, 50

WER Word Error Rate. 128

List of Figures

List of Tables

Abstract

Event-based sensors, built with biological inspiration, differ greatly from traditional sensor types. A standard vision sensor uses a pixel array to produce a frame containing the light intensity at every pixel whenever the sensor is sampled; a standard audio sensor produces a waveform of sound amplitude over time. Event-based sensors, on the other hand, are typically substantially sparser in their output, producing output events that occur upon informative changes in the scene, usually with low latency and accurate timing, and are data-driven rather than sampled.

The outputs produced by these novel sensor types differ radically from traditional sensors. Unfortunately, these differences make it hard to apply standard data analysis techniques to event-based data, despite the advanced state of computational techniques for image understanding and acoustic processing. Machine learning especially has made great strides in recent years towards scene understanding, and particularly in the area of deep learning.

The goal of this thesis is to study how to make use of these novel sensors to draw from the state-of-the-art in machine learning while maintaining the advantages of event-based sensors. This thesis takes the view that frame-based, traditional data has limited the scope of discovery for new kinds of machine learning algorithms. While machine learning algorithms have reached great success, their achievements pale in comparison to biological reasoning, and perhaps this arises from the fundamental assumptions about *what* is processed in addition to *how*. That is, by relaxing expectations on the kinds of data that will be processed, perhaps even better algorithms can be discovered that not only work with biologically-inspired event-based sensors but also outperform traditional machine learning algorithms.

This thesis is studied at multiple levels of abstraction. In Chapter 2, custom hardware platforms are introduced that prototype an existing machine learning algorithm in hardware. That work aims to ensure that the advantages of both state-of-the-art machine learning and the novel sensor types are maintained at the most fundamental hardware level and to understand the limitations of the algorithms better. Indeed, this revealed that the most significant bottleneck when combining both is the accuracy loss compared to traditional machine learning algorithms, and motivates the work in Chapter 3 that dramatically increases the accuracy of event-driven neural networks for fixed, unchanging scenes (e.g., image analysis, perhaps the most well-studied problem in deep learning currently). With that primary limitation addressed, Chapter 4 explores advantages that are unavailable to traditional deep learning but are available to event-driven deep networks.

Chapter 5 forms perhaps the key contribution of this thesis by introducing a novel algorithm, Phased LSTM, that natively works with event-driven sensors observing dynamic and changing scenes. Indeed, as hypothesized above, Phased LSTM offers significant advantages over traditional deep neural networks, both for event-driven inputs and for standard frame-based inputs. Chapter 6 investigates the source of these advantages to identify if the model is sufficiently simple and advantageous. Finally, an observation made in the development of Phased LSTM motivates examining a principle of event-based

sensing within computation as well, explored in Chapter 7, and demonstrates the significant computational speedups that can result when sensor principles are also applied to computation.

Overall, this thesis introduces hardware implementations and algorithms that use inspiration from deep learning and the advantages of event-based sensors to add intelligence to platforms to achieve a new generation of lower-power, faster-response, and more accurate systems.

Abstract (Deutsch)

German translation thanks to Adrian Huber

Von der Biologie inspirierte ereignisbasierte Sensoren unterscheiden sich in vielerlei Hinsicht von traditionellen Sensoren. Ein typischer Bildsensor misst die einfallende Lichtintensität mithilfe einer Pixelmatrix an jedem Pixel und erzeugt ein Einzelbild, welches mit der Abtastgeschwindigkeit ausgegeben wird; ein typischer Audiosensor wiederum zeichnet die Schallamplitude über die Zeit auf. Das von ereignisbasierten Sensoren erzeugte Ausgangssignal ist hingegen im Allgemeinen deutlich kärger als das konventioneller Sensoren, da nur im Falle grösserer Veränderungen des zu messenden Eingangssignals ein Ausgangssignal mit geringer Latenz und präziser Zeitinformation ausgegeben wird. Ereignisbasierte Sensoren erzeugen demnach ein Ausgangssignal, das primär vom Signal selbst abhängt, und zwar in Hinblick auf das entstehende zeitliche Abtastmuster.

Die Ausgangssignale dieser neuartigen Sensoren unterscheiden sich daher im höchsten Masse von traditionellen Sensoren. Diese Unterschiede erschweren die Anwendung üblicher Datenanalysemethoden auf ereignisbasierte Daten, obschon die Bildverarbeitung und die akustische Signalverarbeitung über fortgeschrittene Algorithmen verfügen. Insbesondere das maschinelle Lernen führte im Laufe der letzten Jahre zu grossen Fortschritten im Bereich Szene-Verständnis; unter den vielen Methoden des maschinellen Lernens trug vor allem Deep Learning zu diesen Fortschritten bei.

Das Ziel dieser Doktorarbeit besteht darin, neuartige Sensoren so einsatzfähig zu machen, dass ihre spezifischen Vorteile erhalten bleiben, wobei der neueste Stand der Technik aus dem Bereich des maschinellen Lernens berücksichtigt wird. Diese Doktorarbeit vertritt die Position, dass herkömmliche, regulär abgetastete Daten die Entwicklung neuer Algorithmen im maschinellen Lernen einschränkten. Obwohl das maschinelle Lernen grosse Erfolge verzeichnen kann, verblassen diese Errungenschaften doch, sobald man sie mit menschlichem Denken vergleicht. Mitunter entsteht diese Diskrepanz auf Grund fundamentaler Annahmen bezüglich der Frage, welche Daten zur weiteren Verarbeitung ausgewählt werden, und wie dieselben verarbeitet wird. Abgeschwächte Bedingungen in Hinblick auf zu verarbeitende Daten ermöglichten vielleicht das Auffinden besserer Algorithmen, welche nicht nur mit von der Biologie inspirierten ereignisbasierten Sensoren arbeiten könnten, sondern die auch herkömmliche Algorithmen des maschinellen Lernens übertreffen könnten.

Diese Doktorarbeit ist in mehrere Abstraktionsebenen aufgeteilt. In Kapitel 2 werden speziell erstellte Hardwareplattformen eingeführt, welche es erlauben, bereits bestehende Algorithmen des maschinellen Lernens als Prototypen in Hardware zu implementieren. Das Ziel dieser Arbeit liegt darin, die Vorteile der neuartigen Sensoren sowie modernster Algorithmen des maschinellen Lernens auf Hardwareebene zu erhalten, und weiterhin die Limitationen dieser Algorithmen besser zu verstehen. Der bedeutendste Engpass, der aus der Kombination der neuen Sensoren und modernster Algorithmen resultiert, ist ein zu beobachtender Genauigkeitsverlust, insofern das System mit üblichen Algorithmen des maschinellen Lernens verglichen wird. Diese Beobachtung veranlasste die Arbeit, die in Kapitel 3 präsentiert wird, und welche die Genauigkeit ereignisgetriebener neuronaler Netze, angewendet auf starre, sich nicht

ändernde Szenen wie im Falle der Bildanalyse, dem wahrscheinlich am eingehendsten untersuchten Problem im Bereich des Deep Learning, deutlich erhöhte. Nach dieser Erörterung untersucht Kapitel 4 dann die spezifischen Vorteile ereignisgetriebener tiefer Netzwerke, die für herkömmliches Deep Learning nicht verfügbar sind.

Kapitel 5 ist wohl der Schlüsselbeitrag dieser Doktorarbeit. In diesem Kapitel wird ein neuer Algorithmus, Phased LSTM, eingeführt, der auf Grund seiner intrinsischen Beschaffenheit auf natürliche Art und Weise mit ereignisbasierten Sensoren, welche dynamische und sich verändernde Szenen beobachten, umgehen kann. Wie weiter oben als Vermutung ausgedrückt, verfügt der Phased LSTM Algorithmus über erhebliche Vorteile verglichen mit üblichen Deep Learning Netzwerken, und zwar sowohl für ereignisbasierte als auch für regulär abgetastete Eingangssignale. Kapitel 6 untersucht sodann den Ursprung dieser Vorteile, insbesondere um zu verstehen, ob das Modell genügend einfach und vorteilhaft ist. Zum Schluss wird in Kapitel 7 noch eine Beobachtung aufgegriffen, die während der Entwicklung des Phased LSTM Algorithmus gemacht wurde. Es wird untersucht, inwieweit das Prinzip der ereignisbasierten Signalerfassung auch in Berechnungen anwendbar ist, wobei deutliche rechentechnische Beschleunigungen erzielt werden, insofern die Prinzipien ereignisbasierter Sensoren auf Rechenoperationen angewendet werden.

Diese Doktorarbeit führt damit Hardwareimplementierungen sowie Algorithmen ein, welche von Deep Learning inspiriert sind und die die Vorteile ereignisbasierter Sensoren aufgreifen, um somit intelligente Plattformen zu ermöglichen, die eine neue Generation von zuverlässigeren und schnellansprechenden Systemen mit geringem Stromverbrauch gestatten.

1
An Introduction to Event-Based Sensors and Machine Learning

1.1 The Amazing Progress of Deep Learning

Deep learning currently represents the state-of-the-art solution in virtually all relevant tasks in machine learning across an incredibly diverse variety of domains [1,2]. The advantage these architectures provide over shallow architectures is the ability to extract hierarchies of abstracted features from the underlying input data, giving rise to hierarchical data transformations which have been optimized for a specific task.

Convolutional Neural Networks (CNNs), one kind of deep neural networks, have provided significant leaps in accuracy on difficult computer vision benchmarks such as ImageNet [3,4], currently reaching performance levels that rival humans [5,6] on image classification. Formerly "hard" computer vision datasets such as the CIFAR-10 10-class image classification dataset [7] and the MNIST handwritten digit dataset [8] are now considered to be solved tasks [9].

Yet the advances are not limited to computer vision. In recent years, interest in Recurrent Neural Networks (RNNs), which equip recurrent neural networks with memories, has greatly increased through the application of ever-greater computing resources, improved training algorithms, and larger training databases to enable breakthroughs in temporal sequences. Advances using deep neural networks first broke through in speech recognition [10] and natural language processing [11], but have continued across other domains including image generation [12], question answering [13], and even novel computation types [14,15].

Even more useful, however, is the composition of these models together or as elements in a hierarchy of computation. Image captioning networks [16,17] combine CNNs for visual processing with a language model RNN that can describe the picture in words. New reinforcement learning pipelines [18] also rely heavily on CNNs for visual input processing in concert with standard techniques for reinforcement learning, which was demonstrated with great success for the game Go [19]. Even audio generation at the waveform level is

[1] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *Nature* 521.7553 (2015), pp. 436–444

[2] Jürgen Schmidhuber. "Deep learning in neural networks: An overview". In: *Neural Networks* 61 (2015), pp. 85–117

[3] Jia Deng, Wei Dong, R. Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. "ImageNet: A large-scale hierarchical image database". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848

[4] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Proc. of NIPS*. 2012, pp. 1097–1105

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". In: *The IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 1026–1034

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 770–778

[7] Alex Krizhevsky and Geoffrey Hinton. "Learning multiple layers of features from tiny images". In: (2009)

[8] Yann LeCun, Corinna Cortes, and Christopher JC Burges. *The MNIST database of handwritten digits*. 1998

[9] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. "Deep networks with stochastic depth". In: *European Conference on Computer Vision*. Springer. 2016, pp. 646–661

now possible, using hierarchies of convolutional neural networks that produce substantially more natural-sounding human speech [20].

1.2 Towards Artificial Agents that Exist in the World

HOWEVER, these application areas have arisen from a tradition in machine learning and computer science that is rather divorced from the real world. The current standard implementation for machine learning is to produce a dataset of tuples $< x, y >$ in which every data sample is composed of an input x and a target y, and a dataset consists of aggregates of these tuples together. Datasets are typically cleaned and normalized, and in many cases, a pre-defined training and test split of the data is produced to equalize results from many groups.

This characterization ignores many aspects that epitomize the real world. As these themes will reoccur throughout this thesis, it is worth noting them here:

1. First, power constraints are an ever-present constraint on real-world implementations: they cannot consume arbitrary amounts of energy to perform their intended task.

2. Second, real-world timing and latency constraints are perennial. The state of the natural environment unfolds in continuous time, and in the rare case in which machine learning considers the aspect of time, it is nearly always discrete and time-stepped. Similarly, agents in natural environments cannot take arbitrarily long to respond to input stimuli, as the world continues to evolve during the presentation of input and the calculation of the output.

3. Third, noise is an inescapable feature of the natural environment which must be addressed.

These features are often considered to be confounds on the true interesting problem of creating machines that learn, but it is far from obvious that these aspects can be abstracted away without substantially altering the types of learning methods that are uncovered. This thesis will examine the discoveries that can be yielded when these considerations are indeed taken into account, and opens a door for substantial further research that bridges the domain of deep learning with the many unanswered questions of agents that aim to exist in the real world.

1.3 An Introduction to Event-Based Sensors

NOVEL MACHINE LEARNING METHODS MUST begin with a novel method of acquiring data. Much of the constraints of machine

[10] Alex Graves, Abdel-Rahman Mohamed, and Geoffrey Hinton. "Speech recognition with deep recurrent neural networks". In: *2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2013, pp. 6645–6649

[11] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate". In: *arXiv preprint arXiv:1409.0473* (2014)

[12] Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. "Pixel recurrent neural networks". In: *arXiv preprint arXiv:1601.06759* (2016)

[13] Jason Weston, Sumit Chopra, and Antoine Bordes. "Memory networks". In: *arXiv preprint arXiv:1410.3916* (2014)

[14] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. "Hybrid computing using a neural network with dynamic external memory". In: *Nature* 538.7626 (2016), pp. 471–476

[15] Alex Graves, Greg Wayne, and Ivo Danihelka. "Neural Turing Machines". In: *arXiv preprint arXiv:1410.5401* (2014)

[16] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard Zemel, and Yoshua Bengio. "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention". In: *International Conference on Machine Learning*. 2015

[17] Justin Johnson, Andrej Karpathy, and Li Fei-Fei. "Densecap: Fully convolutional localization networks for dense captioning". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 4565–4574

[18] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (2015), pp. 529–533

[19] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529.7587 (2016), pp. 484–489

learning arise from the way that the natural environment is sampled by conventional sensors. The data is complete, static, and discretized in a distinctly computational way, dramatically different than the world with which biological agents interact. Instead of the standard data approach, this thesis concerns itself primarily with event-driven sensors, which, unlike conventional time-sampled sensors, act as the driver of output data. When a meaningful change occurs in the environment, the sensing element autonomously transmits an event that carries information about this change.

Importantly, this encoding has two fundamental advantages. First, the transmission of data only occurs with informative changes, decreasing redundancy. It also cascades to significantly reduce the downstream power that would have been lost due to data transmission, wasted computation cycles, and extraneous forced computation on irrelevant features. Second, this permits faster latency because the sensor does not need to wait until the next externally-driven acquisition time to communicate this information; it is permitted to send the data as soon as the change occurs. This allows extremely low latencies in a variety of domains.

1.3.1 *Dynamic Vision Sensors*

A major success in event-based sensors is the Dynamic Vision Sensor (DVS) [21], originally proposed in [22] and whose advantages were extended in later variants such as the higher sensitivity DVS [23] and DVS with spike encoding using an asynchronous delta sigma modulator [24]. Furthermore, versions were produced that emit both an event-driven output and standard frame-like intensity output, in the Dynamic and Active-pixel Vision Sensor (DAVIS) [25] and the ATIS [26] vision sensors, permitting standard frame-based vision algorithms to be used alongside event-driven image algorithms using a single sensor.

Fig. 1.1, taken with permission from [27], demonstrates many aspects of the current state-of-the-art DAVIS. Note the simplified DAVIS pixel uses the same photoreceptor for both dynamic vision sensing (event-based vision) and traditional image sensing (frame-based vision). The event-based vision sensor compares the voltage stored on a capacitor to an on- and off-threshold value and produces an output spike when the log photocurrent changes the charge on the capacitor by a threshold amount. This vision sensor was built with an analogy to biological vision, with the photodiode-capacitor circuit representing biological photoreceptors, the differencer representing a function of bipolar cells, and the comparator representing the ganglion cells of the retina.

As an example, the function of the dynamic vision sensor can be seen in the upper right of Fig. 1.1. As the sensor operates on logarithmic intensity, it has much greater dynamic range than a standard image sensor. At top, the log intensity of the scene changes in continuous time. When the log intensity increases by a threshold

[20] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. "Wavenet: A generative model for raw audio". In: *CoRR abs/1609.03499* (2016)

[21] C. Posch, T. Serrano-Gotarredona, B. Linares-Barranco, and T. Delbruck. "Retinomorphic Event-Based Vision Sensors: Bioinspired Cameras With Spiking Output". In: *Proceedings of the IEEE* 102.10 (Oct. 2014), pp. 1470–1484. ISSN: 0018-9219. DOI: 10.1109/JPROC.2014.2346153

[22] Patrick Lichtsteiner, Christoph Posch, and Tobi Delbruck. "A 128× 128 120 dB 15 μs latency asynchronous temporal contrast vision sensor". In: *IEEE Journal of Solid-State Circuits* 43.2 (2008), pp. 566–576

[23] T. Serrano-Gotarredona and B. Linares-Barranco. "A 128 × 128 1.5% Contrast Sensitivity 0.9% FPN 3 μs Latency 4 mW Asynchronous Frame-Free Dynamic Vision Sensor Using Transimpedance Preamplifiers". In: *IEEE Journal of Solid-State Circuits* 48.3 (Mar. 2013), pp. 827–838. ISSN: 0018-9200. DOI: 10.1109/JSSC.2012.2230553

[24] M. Yang, S. C. Liu, and T. Delbruck. "A Dynamic Vision Sensor With 1% Temporal Contrast Sensitivity and In-Pixel Asynchronous Delta Modulator for Event Encoding". In: *IEEE Journal of Solid-State Circuits* 50.9 (Sept. 2015), pp. 2149–2160. ISSN: 0018-9200

[25] R. Berner, C. Brandli, M. Yang, S. C. Liu, and T. Delbruck. "A 240 × 180 10mW 12μs latency sparse-output vision sensor for mobile applications". In: *2013 Symposium on VLSI Circuits*. June 2013, pp. C186–C187

[26] C. Posch, D. Matolin, and R. Wohlgenannt. "A QVGA 143 dB Dynamic Range Frame-Free PWM Image Sensor With Lossless Pixel-Level Video Compression and Time-Domain CDS". in: *IEEE Journal of Solid-State Circuits* 46.1 (Jan. 2011), pp. 259–275. ISSN: 0018-9200

[27] Daniel Neil, Tobi Delbruck, and Shih-Chii Liu. "Event-Driven Deep Multi-Layered Network Architectures". In: IEEE, 2017

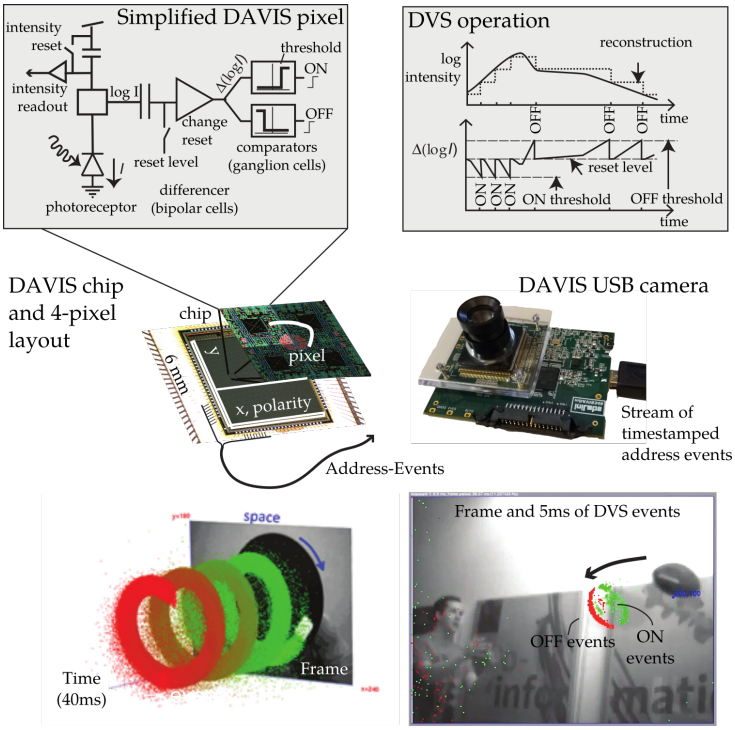


Figure 1.1: Summary of the Dynamic Vision Sensor. At upper left, a simplified circuit diagram of the DAVIS pixel. Upper right, an example schematic of the operation of a DVS sensor. Center, picture of the DAVIS chip and 4-pixel layout; at right, the assembled DAVIS Universal Serial Bus (USB) camera. Bottom left, a frame and event-based view of a spinning dot; at right, a natural scene in which the chip's principle designer catches a football. Note that the frames lag behind the low-latency input events. Used with permission.

(ON threshold) it produces an ON event (bottom plot) and is then reset; when the log intensity decreases by a threshold (OFF threshold) an OFF event is produced. Note that the data rate, therefore, is dependent on the rate of change in the scene and not the duration of the recording.

These sensors with both dynamic and active sensors allow easier comparison between traditional (frame-based) inputs and event-based inputs. Example inputs can be seen in the bottom row of Fig. 1.1. At left, a dot attached to a fan produces a continuous helix in space-time, smoothly describing the rapid motion of the dot. A traditional image sensor would produce a frame-based snapshot that is discontinuous and marred by a blurring of this rapid input. The event-based input, however, maintains a smooth surface over time and does not suffer from blurring. Moreover, if the dot stops spinning, the frame-based sensor will continue to capture and transmit a full frame of unchanging data, while an event-based sensor will produce no further events until a change again occurs.

Finally, the lower right of Fig. 1.1 demonstrates the lower latency of the input events. In this case, a frame is captured every 100ms and read out; between these frames, events are generated which are more current than the last frame captured. In this case, a thrown football approaches the designer of the chip. The events quickly communicate the current, accurate position of the football, but the frames show an older, lagged, and now inaccurate position.

1.3.2 *Dynamic Audio Sensors*

The other major class of sensor that this thesis will focus upon is the Dynamic Audio Sensor, also know as the silicon cochlea, which models aspects of the biophysics of the biological cochlea. In particular, these hardware implementations focus on the tonotopy that emerges from the spatial filtering of the basilar membrane in the cochlea. The circuits focus on implementing bandpass filters that have a range of characteristic frequencies that cover the input frequencies of interest. Furthermore, current designs include a model that mimics the event-driven output of the auditory nerve fibers of the biological cochlea [28,29,30]. The most recent design contains a binaural 64-channel pitch decomposition, demonstrating a low mismatch between filters, low power consumption on the order of 64 μW, a programmable quality factor Q for the filters, and most recently, an asynchronous delta modulation scheme for spike encoding [31].

Similar to the above Dynamic Vision Sensor, the Dynamic Audio Sensor (DAS) produces output events, but these events occur on changes in the per-frequency band amplitude. Two principal methods have been used to transform these analog waveforms into events. In previous versions, an output event was emitted upon zero-crossings of the analog bandlimited waveform, resulting in events that are phase-locked to the signal, allowing the inter-spike interval (ISI) to predict the frequency of the waveform accurately as well as allowing for phase-locking. Current versions [32] use the delta-modulation scheme that emits up- and down-spikes when the amplitude changes by a threshold amount, similar to the on- and off-spikes of a dynamic vision sensor to permit improved reconstructions.

Fig. 1.2 demonstrates the outputs from this audio sensor in response to a spoken digit input. At top, the raw audio amplitude; in the middle, each red dot represents an event; at bottom, the corresponding event rate. Gaps between spoken syllables have no event rate, triggering no processing, allowing processing to be saved only for the spoken sections. The broadly-tuned frequency bands produce spikes in response to the pitch of the spoken word, and the dual up/down events allow accurate reconstruction of the original audio waveform.

[28] S-C. Liu, A. van Schaik, B. Minch, and T. Delbrück. "Asynchronous Binaural Spatial Audition Sensor with $2 \times 64 \times 4$ Channel Output". In: *IEEE Trans. Biomed. Circuits Syst.* 8.4 (2014), pp. 453–464. DOI: 10.1109/TBCAS.2013.2281834

[29] B. Wen and K. Boahen. "A silicon cochlea With active coupling". In: *IEEE Trans. Biomed. Circuits Syst.* 3.6 (2009), pp. 444–455

[30] V. Chan, S. C. Liu, and A. van Schaik. "AER EAR: A Matched Silicon Cochlea Pair With Address Event Representation Interface". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 54.1 (Jan. 2007), pp. 48–59. ISSN: 1549-8328. DOI: 10.1109/TCSI.2006.887979

[31] M. Yang, C. H. Chien, T. Delbruck, and S. C. Liu. "A 0.5 V 55 μW 64 x 2 Channel Binaural Silicon Cochlea for Event-Driven Stereo-Audio Sensing". In: *IEEE Journal of Solid-State Circuits* 51.11 (Nov. 2016), pp. 2554–2569. ISSN: 0018-9200. DOI: 10.1109/JSSC.2016.2604285

[32] M. Yang, C. H. Chien, T. Delbruck, and S. C. Liu. "A 0.5 V 55 μW 64 x 2 Channel Binaural Silicon Cochlea for Event-Driven Stereo-Audio Sensing". In: *IEEE Journal of Solid-State Circuits* 51.11 (Nov. 2016), pp. 2554–2569. ISSN: 0018-9200. DOI: 10.1109/JSSC.2016.2604285

1.4 *Event-based Inputs and Networks: New Challenges and New Opportunities*

However, these novel sensor types produce radically different outputs than traditional sensors. This thesis concerns itself primarily with how to make use of the sensors in a way that takes from the state-of-the-art in machine learning while maintaining the advantages of event-based sensors. These event-based sensors produce a new set of challenges, and with them, a new set of opportunities,

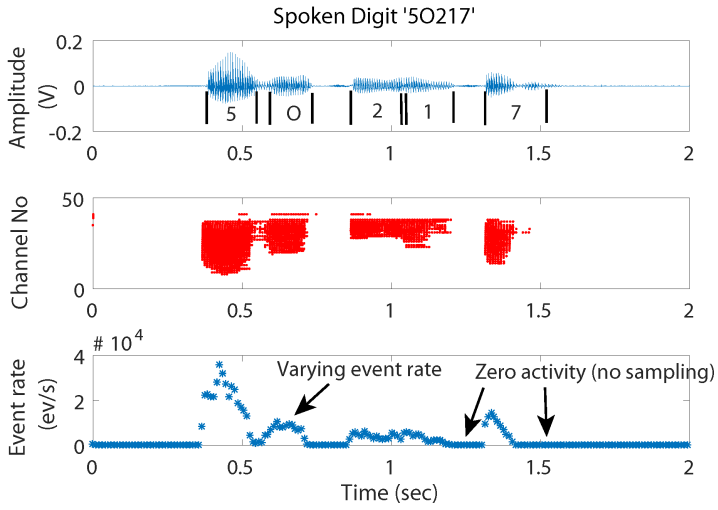


Figure 1.2: Summary of the CochleaLP, the low-power cochlea. Events are produced within the cochlea at a varying rate, and outputs are only processed during active periods. Used with permission.

Constraint	Advantage	Challenge
Externally-driven data	Sparse, meaningful input	Unpredictable in space, time, and density
Gradual Presentation	Adjustable accuracy tradeoffs	Requires stateful representation
Pseudo-simultaneous Results	Low latency	Requires multiple time points to produce quality answer

Table 1.1: Advantages and Challenges in Event-driven Machine Learning

which are summarized in Table 1.1.

The challenges emerge naturally from the function of the sensors, and, while solved in traditional methods, remain persistent challenges when using event-based sensors. First, the inputs are far more unpredictable in event-based sensors. A traditional sensor produces a large but coherent collection of data, then no additional data until the next set of inputs arrive. When input data arrives in a traditional computer, the platform is free to parallelize, reorder, shard, or otherwise aggregate inputs in whatever way will result in the greatest optimization. Indeed, the ability to parallelize across multiple inputs and do similar computations on multiple pieces of data is a primary source of optimization in modern hardware architectures. However, these optimizations cannot directly translate to event-based sensors, in which the external environment determines the statistics of the input data. The inputs are *unpredictable in space, unpredictable in time, and of unpredictable density*, making it quite challenging to apply standard optimization methods to event-driven inputs.

Moreover, an event-based sensor will gradually update the state of a scene over time, as its environment changes in informative ways. Therefore, even a perfect computation on the given data may

yet be incomplete at a given point in time, because not enough data has yet arrived, and that a sequence of multiple inputs may be required to perform a task. In an example on the DVS, the background is invisible until it changes, so a task to classify the background is impossible to solve until either the motion passes across the background or the background changes on the sensor. There is a fundamentally *gradual* presentation of input, and with it, an upper bound on the possible accuracy of the task. Gradual inputs stand in contrast to a standard frame-based implementation of a vision sensor, for example, which produces nothing at all until it presents a complete pixel representation of the environment. Standard computer vision tasks ignore the period before the presentation of input (in which the task is entirely unsolvable), then immediately calculate the result of the task upon presentation of the input (in which the task is assumed to be entirely solvable). Event-based sensors reveal the period between these two states.

Similarly, this gradual presentation of input requires a *stateful representation*. The system that computes the result of the task must maintain a state over time to process the gradual aggregation of inputs. This persistence places an additional burden on the engineering for the computation, as standard frame-based inputs can instantly compute the task for the scene, then discard all information between the presentations. Event-based inputs, however, do not have this luxury and must store the partial results of their computation to aggregate meaning over multiple inputs. Indeed, the construction of a static representation from an image in flux is a deep question explored in neuroscience [33]. Moreover, the gradual presentation of input implies that the answer is likely not available within a single time step, and may require *multiple time points* to aggregate a quality answer.

[33] H Sebastian Seung and Daniel D Lee. "The manifold ways of perception". In: *science* 290.5500 (2000), pp. 2268–2269

However, the advantages that are offered are indeed novel and powerful as well. In return for not having a complete representation of the input, the inputs received from an event-based sensor are fundamentally *sparse* and driven only by changes in the environment. Moreover, the sensors have been designed so that these changes correspond to meaningful changes in the environment, allowing a theoretically minimal amount of information to be communicated over the input link and triggering minimal processing on the receiving system.

The natural implication of a gradual computational system is a new axis of optimization, the *latency-accuracy tradeoff*. This tradeoff contains the unique ability to produce lower-quality answers more quickly, at lower latencies, and through the gradual accumulation of evidence, refine these initial guesses to acquire greater accuracy with greater input. Similarly, by extensions, a gradual system also allows a *computation-accuracy tradeoff*, as a data-driven architecture can provide ways to use the partial result so far to abort computation early if sufficient evidence has been gathered. In data-driven

architectures, data-driven networks save on computation if exposed to fewer data points. Therefore, there can be a tradeoff between computation and accuracy, with greater computation often conferring greater accuracy if the system is permitted to run for more data points or over a longer time.

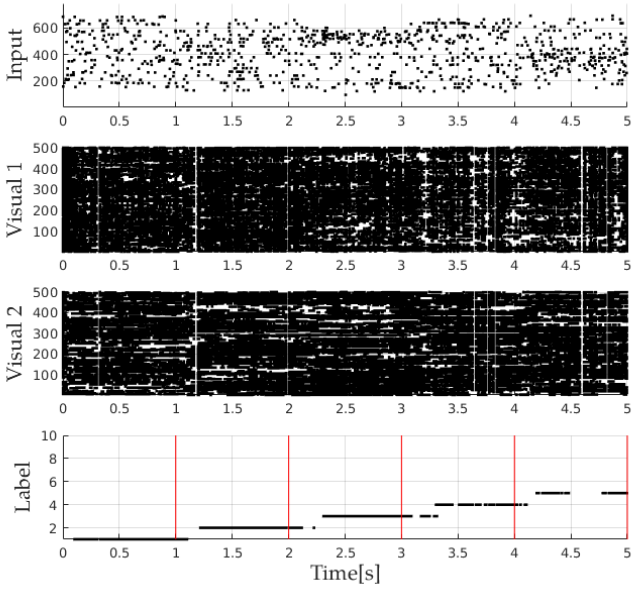


Figure 1.3: Pseudo-simultaneity of input and computation. Three-layer fully-connected deep belief network receives inputs (top), processes them through two intermediate visual abstraction layers (center) to produce an output classification (bottom). Each black dot represents a spike from a neuron, each of which is indexed by its address along the vertical axis. Five sequential handwritten digits (0-4) are presented to the network, each for a duration of 1 second, with red vertical lines indicating input switching. Note that correct output events are emitted quickly after the onset of a new input stimulus and persist throughout the presentation of the stimulus.

Finally, as a consequence of a gradual presentation of the input and the requirement for a stateful representation, the system exhibits what has been termed *pseudo-simultaneity* [34]. That is, the system may produce outputs while inputs are still being presented. While standard machine learning techniques assume a sequential computation paradigm in which an input is presented, an algorithm is applied, and an output is produced, pseudo-simultaneous systems possess the ability to produce outputs while the input is still being computed. This effect can be seen in Fig. 1.3. In this example, a fully-connected neural network receives a spike train drawn from a handwritten input image, with events emitted over the course of a full second before switching to a different digit at the next second. Note that the network begins integrating and classifying very shortly after the onset of a new digit (output spikes are made while input spikes continue to arrive and clarify the representation), and can successfully classify correctly even as the input changes. This property can indeed be useful for agents as it is straightforward to imagine when the initial input, providing enough evidence for a lower-quality rapid output result, is sufficient to react correctly to an environmental stimulus. Pseudo-simultaneity allows dramatically lower latencies, even latencies below what would be considered the minimum latency of a traditional system: the time required to capture a frame.

[34] Clément Farabet et al. "Comparison between frame-constrained fix-pixel-value and frame-free spiking-dynamic-pixel convNets for visual processing". In: *Frontiers in Neuroscience* 6 (2012)

To BEGIN INVESTIGATING the implications of these sensors on algorithms, let's begin at the bottom, at the hardware level, to greater understand the ramifications for the tradeoffs of algorithms that will be considered.

2
Event-based Hardware Systems for Deep Networks

2.1 Why Hardware?

Text in this chapter relating to the Field-programmable Gate Array (FPGA) neural accelerator Minitaur was adapted from [1], while text relating to the SpiNNaker ASIC implementation was adapted from [2], and the novel rounding algorithm's text originally appeared in [3].

[1] Daniel Neil and S-C Liu. "Minitaur, an event-driven FPGA-based spiking network accelerator". In: *IEEE Trans on Very Large Scale Integration (VLSI) Systems* 22.12 (2014), pp. 2621–2628

[2] Evangelos Stromatias, Daniel Neil, Francesco Galluppi, Michael Pfeiffer, Shih-Chii Liu, and Steve Furber. "Scalable Energy-Efficient, Low-Latency Implementations of Spiking Deep Belief Networks on SpiNNaker". In: *Proceedings of the 2015 IEEE International Joint Conference on Neural Networks (IJCNN)*. 2015, pp. 1–8. DOI: 10.1109/IJCNN.2015.7280625

[3] Evangelos Stromatias, Daniel Neil, Michael Pfeiffer, Francesco Galluppi, Steve B Furber, and Shih-Chii Liu. "Robustness of spiking Deep Belief Networks to noise and reduced bit precision of neuro-inspired hardware platforms". In: *Frontiers in Neuroscience* 9 (2015)

To best understand the advantages and disadvantages of processing a spiking input, it is perhaps best to begin with a homogenous, all-spiking hardware system. First, designing a novel hardware system aimed towards a spiking neural network implementation maximally exposes the complexity of the system, and reveals the full scale of advantages and disadvantages of spiking implementations to the designer. With full control over designing the hardware, maximum flexibility can be assured towards maintaining the advantages of spiking systems while opening up the full complexity of a hardware system to investigate areas for possible optimization or opportunities for new kinds of algorithms.

Second, the larger focus of this thesis into new algorithms that maintain the advantages event-driven sensors motivates a transition to an event-driven system for efficiency. Event-driven networks in hardware can have higher energy efficiency because a clock is not used in the network simulation, and not every neuron must update every time step. However, a standard software Central Processing Unit (CPU) is indeed a standard time-stepped, clock-driven processing unit. As the Dynamic Vision Sensor is event-driven, and the Deep Belief Network (DBN) is event-driven, the underlying hardware should also be event-driven to ensure efficiency, consistency, and maximal adaptation to a spiking domain. Concretely, an event-driven implementation would also allow minimizing the latency from input to output.

Third, the range of efficient customizations that can be applied to make spiking neural networks more attractive can be greater with a custom hardware implementation. Spiking neural network platforms often exploit optimizations not available to a general-purpose computer; the behaviour of a spiking network is constrained and

stereotyped, allowing greater optimizations to be applied. These optimizations span a variety of optimizations including specialized routing fabrics [4,5,6], very low-power analog neuron implementations [7], and asynchronous communication links [8,9,10] among others.

Finally, designing hardware directly for a spiking neural network implementation more clearly exposes fundamental bottlenecks in the design, allowing a more clear understanding of both which advantages are possible and which are the most significant bottlenecks in performance. Deep networks have substantial computational costs in execution (as well as during training), which has motivated a variety of platforms to accelerate that inference pass. Spiking neural networks can possibly be even cheaper, and further understanding the tradeoffs of a spiking implementation in light of deep neural networks could offer great advantages.

This chapter of the thesis concerns itself with the development of hardware to explore these questions. In Sections 2.2.1-2.2.13, a fully-custom FPGA implementation is introduced and discussed. Section 2.3 discusses an implementation of a spiking Deep Belief Network on SpiNNaker, a pre-existing hardware system optimized for spiking neural network systems. As anticipated, in the process of obeying the limitations of the hardware an important algorithmic result emerged, which is discussed in Section 2.4.

2.2 *Minitaur*

A fully-event driven deep spiking system was made possible after the introduction of a novel training paradigm [11] that allowed the conversion of a DBN [12] to a spiking neural network. That work further demonstrated a unified deep spiking neural network framework[13], driven by the event-based Dynamic Vision Sensor, and implemented on pools on event-based Leaky Integrate-and-Fire (LIF) neurons. The full methodology, using spike-based inputs, running on spike-based neurons, and producing output classification events in continuous time, was implemented on a standard software CPU architecture.

However, current computing architectures are not ideally suited for network architectures like neural networks. The inherent massive parallelism of neurons, in which each performs a similar computation at the same time, implies a more parallel architecture than what CPUs currently provide. Graphical Processing Units (GPUs) can capitalize on the parallelism of the network but they are not suited to event-driven computation. Current GPU programming paradigms use a kernel-launch approach in which a large chunk of computation is off-loaded onto the GPU with a batch of data instead of continuously run. Additionally, the power consumption of GPUs precludes most embodied robotics applications. Because of the above reasons, none of these platforms are suited for network architectures such as spiking DBNs, especially event-driven DBNs which combine the dramatic advances achieved with DBNs [14] with

[4] Ben Varkey Benjamin, Peiran Gao, Emmett McQuinn, Swadesh Choudhary, Anand R Chandrasekaran, J Bussat, Rodrigo Alvarez-Icaza, John V Arthur, PA Merolla, and Kwabena Boahen. "Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations". In: *Proceedings of the IEEE* 102.5 (2014), pp. 699–716

[5] Giacomo Indiveri, Federico Corradi, and Ning Qiao. "Neuromorphic architectures for spiking deep neural networks". In: *2015 IEEE International Electron Devices Meeting (IEDM)*. 2015, pp. 2–4

[6] Paul A Merolla, John V Arthur, Rodrigo Alvarez-Icaza, Andrew S Cassidy, Jun Sawada, Filipp Akopyan, Bryan L Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, et al. "A million spiking-neuron integrated circuit with a scalable communication network and interface". In: *Science* 345.6197 (2014), pp. 668–673

[7] Giacomo Indiveri, Federico Corradi, and Ning Qiao. "Neuromorphic architectures for spiking deep neural networks". In: *2015 IEEE International Electron Devices Meeting (IEDM)*. 2015, pp. 2–4

[8] Paul A Merolla, John V Arthur, Rodrigo Alvarez-Icaza, Andrew S Cassidy, Jun Sawada, Filipp Akopyan, Bryan L Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, et al. "A million spiking-neuron integrated circuit with a scalable communication network and interface". In: *Science* 345.6197 (2014), pp. 668–673

[9] R. Serrano-Gotarredona and others. "CAVIAR: A 45k Neuron, 5M Synapse, 12G Connects/s AER Hardware Sensory–Processing– Learning–Actuating System for High-Speed Visual Object Recognition and Tracking". In: *IEEE Trans on Neural Networks* 20.9 (2009), pp. 1417–1438

[10] Ben Varkey Benjamin, Peiran Gao, Emmett McQuinn, Swadesh Choudhary, Anand R Chandrasekaran, J Bussat, Rodrigo Alvarez-Icaza, John V Arthur, PA Merolla, and Kwabena Boahen. "Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations". In: *Proceedings of the IEEE* 102.5 (2014), pp. 699–716

[11] Peter O'Connor, Daniel Neil, Shih-Chii Liu, Tobi Delbruck, and Michael Pfeiffer. "Real-time classification and sensor fusion with a spiking Deep Belief Network". In: *Frontiers in Neuroscience* 7 (2013)

[12] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. "A fast learning algorithm for deep belief nets". In: *Neural computation* 18.7 (2006), pp. 1527–1554

the performance of event-based processing [15].

FPGA architectures are an excellent first step towards constructing new hardware. They are low cost devices, available off-the shelf in a variety of configurations, and are reconfigurable to allow updating and the reconfiguration of the source design as necessary. They inherently support parallel processing and contain enough local memory to cache the many weights present in a typical deep network. They are low power compared to CPUs and GPUs, and a successful design can be turned into a lower power, higher-performance Application-Specific Integrated Circuit (ASIC) in the future. Additionally, the design code can be shared electronically to allow researchers to collaborate and to upgrade their physical hardware without additional cost or physical adjustment.

The following sections covers the work of Minitaur, an event-driven FPGA-based spiking neural network accelerator. This accelerator is used to study the ability of an FPGA platform to implement a real-time, event-driven deep spiking network. Section 2.2.1 introduces prior work on FPGAs and spike-based neural network accelerators, followed by the theory of the event-based processing in Section 2.2.1. Section 2.2.4 discusses the specific design of Minitaur. In Section 2.2.7, Minitaur's performance characteristics as well as real-world performance on the MNIST and newsgroup classification tasks will be evaluated. Lastly, Section 2.2.13 reviews future challenges and raises questions for further investigation.

2.2.1 *Prior Work*

Significant work has been invested into software algorithms allowing the acceleration of artificial neural networks, and event-driven methods have emerged as one way of speeding up the simulation time of these networks. An efficient event-driven software implementation was described in [16], and the time complexity of scaling these networks was investigated in [17]. Event-driven optimizations have been considered for advanced neuron model implementations as well, notably including the Hodgkin-Huxley model in [18]. A comprehensive review of the performance of both event-driven and time-stepped software algorithms and implementations of spiking neural networks can be found in [19].

Hardware systems which accelerate these spiking networks, including FPGA-based designs, are reviewed in [20,21]. These FPGA systems are predominantly time-stepped hardware accelerators as in [22,23,24], achieving high speeds but with performance proportional to the size of the network. Event-driven, sparser-computation hardware implementations [25,26,27] are rare, typically focusing on using biologically descriptive neuron models such as the Izhikevich model [28] and biological network topologies.

In recent years, machine learning approaches have examined alternative network topologies such as those used in DBNs [29], a

[13] Peter O'Connor, Daniel Neil, Shih-Chii Liu, Tobi Delbruck, and Michael Pfeiffer. "Real-time classification and sensor fusion with a spiking Deep Belief Network". In: *Frontiers in Neuroscience* 7 (2013)

[14] G.E. Hinton and R.R. Salakhutdinov. "Reducing the dimensionality of data with neural networks". In: *Science* 313.5786 (2006), pp. 504–507

[15] Peter O'Connor, Daniel Neil, Shih-Chii Liu, Tobi Delbruck, and Michael Pfeiffer. "Real-time classification and sensor fusion with a spiking Deep Belief Network". In: *Frontiers in Neuroscience* 7 (2013)

[16] A. Delorme and S.J. Thorpe. "SpikeNET: an event-driven simulation package for modelling large networks of spiking neurons". In: *Network: Computation in Neural Systems* 14.4 (2003), pp. 613–627

[17] I. Marian, R. Reilly, and D. Mackey. "Efficient event-driven simulation of spiking neural networks". In: *Proceedings of 3rd WSES International Conference on: Neural Networks and Applications*. 2002

[18] C.J. Lobb, Z. Chao, R.M. Fujimoto, and S.M. Potter. "Parallel event-driven neural network simulations using the Hodgkin-Huxley neuron model". In: *Workshop on Principles of Advanced and Distributed Simulation (PADS) 2005*. 2005, pp. 16–25

[19] Romain Brette, Michelle Rudolph, Ted Carnevale, Michael Hines, David Beeman, James M Bower, Markus Diesmann, Abigail Morrison, Philip H Goodman, Frederick C Harris Jr, et al. "Simulation of networks of spiking neurons: a review of tools and strategies". In: *Journal of Computational Neuroscience* 23.3 (2007), pp. 349–398

[20] J. Misra and I. Saha. "Artificial neural networks in hardware: A survey of two decades of progress". In: *Neurocomputing* 74.1 (2010), pp. 239–255

[21] L.P. Maguire, T.M. McGinnity, B. Glackin, A. Ghani, A. Belatreche, and J. Harkin. "Challenges for large-scale implementations of spiking neural networks on FPGAs". In: *Neurocomputing* 71.1 (2007), pp. 13–29

[22] D.B. Thomas and W. Luk. "FPGA accelerated simulation of biologically plausible spiking neural networks". In: *17th IEEE Symposium on Field Programmable Custom Computing Machines (FCCM '09)*. 2009, pp. 45–52

multi-layered probabilistic generative model. The individual layers consist of undirected graphical models called Restricted Boltzmann Machines (RBMs) with a bottom layer of "visible" sigmoidal units and a top layer of "hidden" sigmoidal units, bidirectionally connected with symmetric weights. When RBMsare stacked to form a DBN, the hidden layer of the lower RBM becomes the visible layer of the next higher RBM. DBNs have proved effective in a variety of domains, with notable successes in areas such as machine vision [30] and machine audition [31,32]. In [33], spiking DBNs are constructed by replacing these sigmoidal units with spiking Leaky Integrate-and-Fire (LIF) neurons.

Minitaur extends this prior work on event-driven systems, FPGA implementations, and DBNs to accelerate spiking neural network implementations. It is a low-power, compact, event-driven system with a strong focus on spiking deep networks as an application domain. The system supports the loading of arbitrary spiking neural networks at runtime of up to 65,536 neurons and millions of synapses (Table 2.2). With its focus on optimized memory fetches and simplified neuron models as in [34], as well as on low power dissipation, it eschews the high-power, high-performance memory elements used in [35,36]. Its performance is validated on two common machine learning tasks using a DBN composed of spiking neurons as described in Section 2.2.7.

2.2.2 *Event-Driven Neural Model*

The neural model used on Minitaur is a common spiking model containing three sub-models: a soma described by the LIF model, an instantaneous synapse for input current, and a fixed-delay axon for spike generation. The LIF model is both mathematically and intuitively simple; the cell membrane is modeled as a capacitor with a leak [37]. This simple circuit forms an exponentially decaying Resistor-Capacitor (RC) system with decay time constant τ_m. In a time-stepped model, the cell membrane voltage V_{mem} on the $n+1$th step can be calculated as follows:

$$V_{mem}(n+1) = V_{mem}(n) \cdot e^{-\Delta t/\tau_m} \tag{2.1}$$

where Δt is the time step.

The synapse model has instantaneous dynamics and a step increase of $W^{i,j}$ is added to the membrane potential of neuron i when it receives a spike from input neuron j. The model implements three discontinuities to more accurately model biological systems: a threshold (V_{thr}) where a neuron makes a spike once $V_{mem} > V_{thr}$, a reset potential for the membrane after a spike (V_{reset}), and a refractory period (t_{ref}) during which a neuron cannot make a new spike after it spikes. The complete time-stepped model can be found in Algorithm 1.

This algorithm can easily be transformed into an event-driven

[23] A. Cassidy, A.G. Andreou, and J. Georgiou. "Design of a one million neuron single FPGA neuromorphic system for real-time multimodal scene analysis". In: *45th Annual Conference on Information Sciences and Systems (CISS)*. 2011, pp. 1–6

[24] B. Leung, Y. Pan, C. Schroeder, S. O. Memik, G.n Memik, and M. Hartmann. "Towards an 'early neural circuit simulator': A FPGA implementation of processing in the rat whisker system". In: *International Conference on Field Programmable Logic and Applications (FPL 2008)*. 2008, pp. 191–196

[25] K. Cheung, S.R. Schultz, and P.H.W. Leong. "A parallel spiking neural network simulator". In: *International Conference on Field-Programmable Technology (FPT 2009)*. 2009, pp. 247–254

[26] K. Cheung, S. R. Schultz, and W. Luk. "A large-scale spiking neural network accelerator for FPGA systems". In: *International Conference Artificial Neural Networks and Machine Learning (ICANN 2012)*. Vol. 7552. Springer, 2012, pp. 113–120

[27] R. Agis, E. Ros, J. Diaz, R. Carrillo, and E. M. Ortigosa. "Hardware event-driven simulation engine for spiking neural networks". In: *International Journal of Electronics* 94.5 (2007), pp. 469–480

[28] E.M. Izhikevich. "Simple Model of Spiking Neurons". In: *IEEE Transactions on Neural Networks* 14 (2003), pp. 1569–1572

[29] G.E. Hinton and R.R. Salakhutdinov. "Reducing the dimensionality of data with neural networks". In: *Science* 313.5786 (2006), pp. 504–507

[30] Dan Claudiu Ciresan, Ueli Meier, Luca Maria Gambardella, and Jürgen Schmidhuber. "Deep, big, simple neural nets for handwritten digit recognition". In: *Neural Computation* 22.12 (2010), pp. 3207–3220

[31] A. Mohamed, G. E. Dahl, and G. Hinton. "Acoustic modeling using deep belief networks". In: *IEEE Transactions on Audio, Speech, and Language Processing* 20.1 (2012), pp. 14–22

[32] F. Seide, G. Li, and D. Yu. "Conversational Speech Transcription Using Context-Dependent Deep Neural Networks". In: *Interspeech*. 2011, pp. 437–440

[33] Peter O'Connor, Daniel Neil, Shih-Chii Liu, Tobi Delbruck, and Michael Pfeiffer. "Real-time classification and sensor fusion with a spiking Deep Belief Network". In: *Frontiers in Neuroscience* 7 (2013)

Algorithm 1: Time-Stepped Updating of a LIF Network

Given: N input neurons connected to M neurons
Given: $S^t \in R^{N \times M}$, a binary matrix where the elements indicate the presence or absence of a spike between a particular input neuron j to neuron i at time t
Given: set of all weights $W \in R^{N \times M}$
$t_{re}^{1..M} \leftarrow 0$ ▷ Previous refractory end time
for $t \leftarrow 0 : \Delta t : t_{final}$ **do** ▷ Loop through all time
  for $i \leftarrow 1$ to M **do** ▷ Loop through all neurons
    if $t > t_{re}^i$ **then** ▷ Ensure not refractory
      $V_{mem}^i \leftarrow V_{mem}^i \cdot e^{-\Delta t / \tau_m}$ ▷ Decay membrane
      $V_{mem}^i \leftarrow W^{i,1..N} \cdot S^{t,i,1..N} + V_{mem}^i$ ▷ Input
    end if
    if $V_{mem}^i > V_{thr}$ **then**
      DoSpiking()
      $V_{mem}^i \leftarrow V_{reset}$ ▷ Reset membrane potential
      $t_{re}^i \leftarrow t + t_{ref}$ ▷ Set refractory end
    end if
  end for
  $t \leftarrow t + \Delta t$
end for

equivalent. Since the input current is instantaneous and the membrane potential decays away exponentially, it is only necessary to check for firing after the membrane potential has been updated when there is an input spike. The time of the previous update is stored; when the next spike arrives, the neuron membrane is decayed according to the time difference, then summed with the instantaneous input current. This yields the neuron model used in the Minitaur system. The neuron only updates on input spikes, so the computation speed is now proportional to network activity, not numbers of neurons. The complete event-driven execution is described by Algorithm 2.

2.2.3 *Simulation*

A model of the hardware was created in Matlab to ensure the viability of the design and to quickly prototype the effects of parameter adjustment. This implementation was primarily used for prototyping caching strategies since memory bandwidth, rather than compute time, fundamentally limits the performance of the hardware. For more details on these strategies, see Section 2.2.5.

2.2.4 *Spartan 6 FPGA Architecture*

Minitaur was designed using the low-cost Xilinx Spartan-6 platform. The full implementation was done on a ZTEX USB 1.15 board, which contains 128 MB of DDR2 RAM, a microSD card slot for storage, 128 KB flash memory for a bootloader, and an FX2 chip for USB interfacing. The commercially available complete board

[34] T. Schoenauer, N. Mehrtash, Andreas Jahnke, and H. Klar. "MASPINN: novel concepts for a neuroaccelerator for spiking neural networks". In: (1999), pp. 87–96. DOI: 10.1117/12.343072. URL: +%20http://dx.doi.org/10.1117/12.343072

[35] A. Cassidy, A.G. Andreou, and J. Georgiou. "Design of a one million neuron single FPGA neuromorphic system for real-time multimodal scene analysis". In: *45th Annual Conference on Information Sciences and Systems (CISS)*. 2011, pp. 1–6

[36] K. Cheung, S. R. Schultz, and W. Luk. "A large-scale spiking neural network accelerator for FPGA systems". In: *International Conference Artificial Neural Networks and Machine Learning (ICANN 2012)*. Vol. 7552. Springer, 2012, pp. 113–120

[37] N. Brunel and M. C. W. van Rossum. "Lapicqueś 1907 paper: from frogs to integrate-and-fire". In: *Biological Cybernetics* 97.5 (2007), pp. 337–339

Algorithm 2: Event-Driven Updating of a LIF Network

Given: set of all sorted input spike times Q_t
Given: set of corresponding destination neuron indices Q_{dest}
Given: set of corresponding source neuron indices Q_{src}
Given: set of all weights $W \in R^{N \times M}$
$t_{re}^{1..M} \leftarrow 0$ ▷ Reset refractory end time
$t_{prev}^{1..M} \leftarrow 0$ ▷ Reset previous input time
for k in length(Q_t) **do**
  $t \leftarrow Q_t^k$ ▷ Obtain spike time
  $i \leftarrow Q_{dest}^k$ ▷ Obtain index of neuron to update
  $V_{mem}^i \leftarrow V_{mem}^i \cdot e^{-(t - t_{prev}^i)/\tau_m}$ ▷ Decay membrane
  if $t > t_{re}^i$ **then**
    $V_{mem}^i \leftarrow V_{mem}^i + W^{i,Q_{src}^k}$ ▷ Add impulse
  end if
  if $V_{mem}^i > V_{thr}$ **then**
    DoSpiking()
    $V_{mem}^i \leftarrow V_{reset}^i$ ▷ Reset membrane potential
    $t_{re}^i \leftarrow t + t_{ref}$ ▷ Set refractory end
  end if
  $t_{prev}^i \leftarrow t$
end for

Table 2.1: Minitaur Parameters

Parameter	Size	Format
Simulation Parameters		
τ_m	16 bits	Fixed-Point (5.11)
t_{ref}	16 bits	Fixed-Point (5.11)
V_{thr}	15 bits	Fixed-Point (4.11)
Neuron State Parameters		
V_{mem}	16 bits	Signed Fixed-Point (5.11)
Timestamp	24 bits	Integer
t_{re}	16 bits	Integer
Address	16 bits	Integer
Refractory State	1 bit	Boolean
Connection Weight	16 bits	Signed Fixed-Point (5.11)

(http://www.ztex.de) is low cost and ideal for off-the-shelf interfacing and computation. The Xilinx Spartan-6 LX150 contains 150k logic cells, the largest of the Spartan-6 family.

In addition to the large number of logic cells, the Spartan-6 contains 180 Digital Signal Processor (DSP) units for low-power parallel math processing. These DSPs support two 18 bit operands for fixed-point multiplication and addition. Importantly, the Spartan-6 has a total of 549 KB of memory in 268 individually addressable low-latency Block Random Access Memorys (BRAMs). These BRAMs require one cycle for fetch and optionally one cycle to register the output of the fetch, making them ideal for core-specific caching.

The maximum supported clock speed of the Xilinx Spartan-6 is

400 Megahertz (MHz), though as on all FPGA devices, this number is heavily dependent on clock load and routing.

2.2.5 *Minitaur Design Principles*

The performance-limiting step in an event-based neural network system occurs during spike generation. When a neuron spikes, it performs two very memory intense operations: determining the recipient neurons of the spike (between 10^2 and 10^4 destinations, typically), and determining the weights of each of these connections. Accomplishing these two tasks quickly is of paramount importance in optimizing the system's performance.

To minimize the impact of connection lookups, rule-based connections are stored. Although true biological networks are typically recurrent and difficult to simplify, artificial neural networks tend to follow specific connectivity patterns. DBNs, autoencoders, single-layer restricted Boltzmann machines, and multilayer perceptrons all have a very stereotyped structure. Namely, there is a layer of neurons receiving projections from the previous layer and projecting connections to the following layer, typically in an all-to-all fashion. A connection rule can be stored very efficiently by stating connections in a ranged-rule format for example, "all neurons in layer 1 project to all neurons in layer 2," requiring only to store the source (SRC) start, SRC end, destination (DEST) start, and DEST end addresses to map the connection for an entire layer.

Managing the multitude of outgoing Post-synaptic currentss (PSCs) is also a challenge for a neural network accelerator system as neurons typically have a very large number of output connections. When storing spikes in a spike queue, instead of storing the addresses of neurons that are receiving spikes it is vastly more efficient to store the addresses of neurons from which spikes are coming. In this way, the spike queue stores the SRC addresses, not the DEST addresses, and only performs the rule lookup to get the DEST address when evaluating the post-synaptic update after axonal delay.

Finally, cache locality is critical to optimizing neuron weight and state lookups. In an event-driven system that interacts with the real world, there are also likely to be significant patterns in the input data that can be exploited by the system. In the DVS event-driven retina system [38], for example, a pixel that sees an ON event (on contrast change) is likely to have an OFF (off contrast change) event soon after because of the movement of a spatially-extended object. These inherent correlations offer a major advantage over time-stepped systems, and there is no wasted computation as every input spike represents a change in the world which necessitates an update of the output. For spatially similar input events, the weight and state values can be cached and fetched much more quickly. Ensuring locality was done by 'striping' the neurons across the computational cores. The last 5 bits of the neuron ID assigns each

[38] Patrick Lichtsteiner, Christoph Posch, and Tobi Delbruck. "A 128× 128 120 dB 15 μs latency asynchronous temporal contrast vision sensor". In: *IEEE Journal of Solid-State Circuits* 43.2 (2008), pp. 566–576

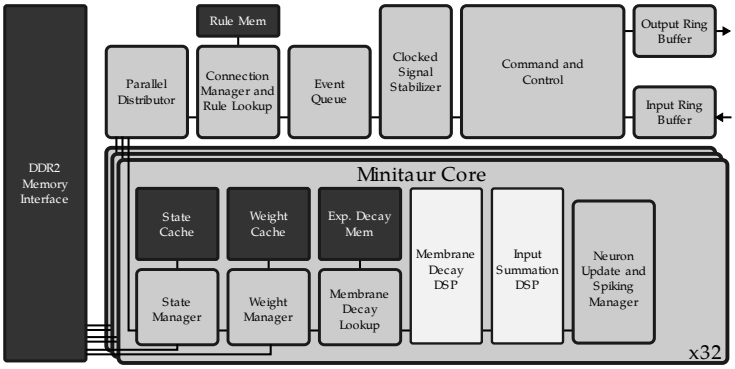


Figure 2.1: Simplified architecture of the Minitaur system. It contains 32 parallel cores and 128 MB of DDR2 for main memory. Each core has 2048 KB of state cache, 8192 KB of weight cache, and 2 DSPs for performing fixed-point math (one multiplying the decay, one for summation of the input current). The exponential decay lookup uses 2048 KB of Random Access Memory (RAM), preloaded from design and used as a Read-only Memory (ROM).

neuron to a core, allowing a given core to store state and weights for frequent and recently-active neurons without contention.

2.2.6 *Minitaur Implementation*

The simplified architecture diagram can be found in Fig. 2.1, and a list of the parameters and their formats can be found in Table 2.1. The system is designed to exploit commonalities in modern artificial neural networks to allow for greatly reduced computational load. Though Minitaur is a digital, clock-based system, no processing occurs except upon input events. This hybrid approach takes advantage of the ease-of-use of digital tools as well as the dramatic computation reduction of event-driven processing.

In this implementation, spikes (events) arrive over USB in packets, stamped with a 4-byte timestamp, a 1-byte layer indicator, and a 2-byte neuron address. After passing through a ring buffer to allow for rapid bursts of spikes, the spikes are dispatched to the event queue where they are sorted by timestamp and layer. Time sorting is presumed only necessary with axonal delays and mixing spikes from external sources (e.g., spike-based neuromorphic devices) with on-Minitaur computation layers.

In order to support recurrent connections and axonal delays, a time delay must exist between when a neuron spikes and when that spike is delivered to its receiving neuron. The event queue maintains a sorted list of incoming spikes as a priority queue, which allows for O(log(n)) operations of insertion and root node extraction. The event queue uses a 5-byte index key comprised of a 4-byte timestamp and a 1-byte layer index to store the neuron address. In this way, all spikes from the same layer at the same time (a common occurrence with all-to-all connectivity and identical delays) will be sorted together and separated from the spikes of another layer at the same time. Seven 2048-byte block RAMs are used to store up to 2048 events simultaneously. A flag ensures spikes cannot be emitted from a layer until all inputs to that layer at that time have been evaluated.

Simultaneously, the parallel distributor block, which connects all the cores together in Fig. 2.1, checks the event queue to extract the first available spike for processing. Spikes are stored according to

spike origin, not spike destination, so a connection lookup is necessary to determine which neurons will have membrane potential updates.

Each neuron is assigned an ID number and neurons in a given layer are assigned consecutive IDs. In this way, a connection can be very compactly represented: 4 bytes for the start and end of each of the SRC and DEST addresses yields a 16-byte range rule. To support multiple layer fanout, all possible rules are matched. Note that extremely complex, non-layer-based networks can still be represented in this form using point-to-point connectivity. The number of comparisons is usually very small when describing a typical DBN for Minitaur's intended use; for example, five rules are sufficient to describe the MNIST handwritten digit identification network shown in Fig. 2.3 (one for each layer, and one to map output to the computer). During a connection lookup, the source address is iteratively compared to all connection rules in the rule memory, and if a rule is found with a source range containing the input spike address, the output range is passed as the range to compute.

The bottom 5 bits of the address of the output range are used to assign a particular neuron update to a particular core: for example, a neuron with address ID 1025 will always be assigned to core 1. This allows for cache locality and obviates any issues causing stale caches from other cores updating a given neuron. The parallel distributor assigns a batch of 32 neurons to be updated; waits until they are completed; and increments to assign the next batch of 32. This continues until the entire destination range of neurons has been addressed.

The event-driven Algorithm 2 is executed by the core once a neuron is assigned. State fetching is done simultaneously with weight fetching to minimize latency, and as soon as the state is fetched it is passed to the DSP for computation. The state is stored as an 8-byte chunk: 2 bytes for the refractory end time, 3 bytes for the last-update timestamp, 2 bytes for state, and 1 byte for assorted information including the cache tag, an initialization bit, and a refractory bit. Because the DSP does not support exponentiation, a ROM lookup table is generated with exponential decay factors. Obtaining the time delta and dividing by the membrane time constant yields an integer lookup index in this memory, and the value at the address j, of 1024 addresses, is $e^{-j/128}$, approximated to 16 bits of fixed-point accuracy (5 integer bits and 11 fractional bits). This yields an accurate decay range from $\frac{1}{128}\tau$ to 8τ in steps of $\frac{1}{128}\tau$. The weight is stored as 2 bytes in the same fixed-point format. During neuron updates, the membrane is decayed, followed by an impulse according to the weight of the input neuron, and the potential is compared to the threshold. If the threshold is exceeded, then a flag is raised by the core. After arbitration and depending on system parameters, that spike is either assembled and sent to the computer or it is added to the spike queue with an axonal delay, where it will

be sorted according to its time and layer.

Fast local memory is key to optimizing neural network computations by minimizing the impact of weight lookups. This cache is implemented with a variant of the direct-mapped cache algorithm: each neuron or weight lookup has only one location that it can be mapped to, and at each location a reference counter tracks the number of consecutive misses. The penalty for a DDR2 swap is not as severe as a hard disk, so occasional contention for a specific memory location is an acceptable tradeoff for very fast lookup.

For neuron state cache lookups, the 8-bit cache address is formed using the middle part of the neuron's address. The lower 5 bits are common to all the neurons at a given core due to the computational partitioning; the upper 3 bits are used as a tag. Combined together, the core-specified 5 bits, the cache address's 8 bits, and the tag's 3 bits recovers the entire 16-bit neuron address. In the case of a cache hit (i.e., a value was successfully retrieved from cache), a reference bit is set to 1; in the case of a miss (i.e, the value is not cached and must be fetched from main memory), the bit is cleared, and if already cleared, the entry is swapped out. The neuron state is stored in 8 bytes including the tag, allowing for 256 entries per core using one 2KB BRAM.

For neuron weight lookups, a two-stage approach is used. Optimized for fanout, the local weight cache is separated into blocks of 16 entries. Each block of 16 has a single SRC neuron address and up to 15 DEST neuron addresses. On a cache lookup, the lower 7 bits of the SRC address are used to index the cache; if the SRC matches, then the 4 lower bits of the DEST addresses are used to calculate a jump offset from the initial SRC address. If the DEST address matches, then the weight is retrieved from the cache. Here, the reference bit for the SRC address is 2 bits since up to 15 DEST entries could be swapped out at once; thus, four consecutive misses are required to swap an address out. The DEST address uses only 1 reference bit, requiring two misses to swap out.

2.2.7 *Results*

The completed Minitaur design utilizes 22k logic slices, 23% of the capacity of the Spartan-6 LX150 FPGA. The power consumption of the FPGA is 1.5W, of which 400 mW supports electronics external to the FPGA. Of the on-chip power budget, 10.0% supports logic and signals, 16.2% supports the 200 (of 268) block memories heavily used as cache to speed up the system, and 73.8% is due to the Phase-locked loopss (PLLs), clock distributors, IOs, and leakage. The system makes use of primarily 3 clocks: the DDR2 clock at 132 MHz, the USB I/O clock at 48 MHz, and the logic clock operating at 75 MHz. Minitaur is an early-stage device, and many further optimizations can be made to significantly increase its performance. A summary of the results can be found in Table 2.2. Following [39], the connections per second, or more specifically the post-synaptic

[39] J. Misra and I. Saha. "Artificial neural networks in hardware: A survey of two decades of progress". In: *Neurocomputing* 74.1 (2010), pp. 239–255

Benchmark	Benchmark Results
Clock Rate	75 MHz
Power Consumption	1.1W (idle) - 1.5W (peak)
Average USB-to-USB Latency	236μs
Supported Number of Neurons	65536
Supported Number of Synapses	16.78 Million
Peak Performance	18.73 Million PSCs/sec
MNIST Accuracy	92.00%
MNIST Performance	4.88 Million PSCs/sec
20 Newsgroups Accuracy	71.07%
Newsgroups Performance	76 Thousand PSCs/sec
Core 2 Duo P7350 2.0GHz CPU	
Power Consumption	20W (peak)
MNIST Performance	33.6 Million PSCs/secc
Newsgroups Performance	173 Thousand PSCs/sec

Table 2.2: System Performance

currents per second (PSCs/sec), was chosen as the primary performance metric.

The performance statistics were gathered on a Intel Core 2 Duo P7350 clocked at 2.00 Gigahertz (GHz) running Ubuntu Linux 12.04. Minitaur was connected via USB using the Java libusb device wrapper and received input spikes from benchmarking software on the computer for performance benchmarking, the MNIST task, and newsgroup classification task. Additionally, the CPU PSCs/sec performance statistics were calculated using a high-performance parallel Matlab implementation of LIF neuron networks, rather than a block-wise model of Minitaur, to ensure a fair peak performance comparison.

2.2.8 *MNIST Handwritten Digits*

The system was tested extensively with the well-studied MNIST benchmark of handwritten digits [40]. Where not otherwise specified, performance results were obtained using the full test set of 10,000 handwritten digits after training on the full 60,000 digit training set. To convert the static images into events, the 28x28 images were vectorized into 784 neuron addresses and spikes were sent with probability proportional to the intensity of the pixel.

The final feed-forward network was 784-500-500-10 units in size (see Fig. 2.3). Since each layer is connected in the standard all-to-all fashion, this yields 647,000 synapses in this task and 1785 neurons. Using weights previously trained to achieve 94.2% accuracy on the MNIST task with LIF neurons in software ([41]) and shown in Fig. 2.2, Minitaur achieved 92% accuracy with 1000 spikes per image; the effects of different input volume per image is discussed later in this work (Fig. 2.5). The loss of accuracy is likely due to the fewer bits for representation of the weights, as the computer

[40] Yann LeCun, Corinna Cortes, and Christopher JC Burges. *The MNIST database of handwritten digits*. 1998

[41] Peter O'Connor, Daniel Neil, Shih-Chii Liu, Tobi Delbruck, and Michael Pfeiffer. "Real-time classification and sensor fusion with a spiking Deep Belief Network". In: *Frontiers in Neuroscience* 7 (2013)

Figure 2.2: Visualization of the weights between the first layer and second layer of the MNIST network trained using rate neurons. This figure shows a sample of 100 neurons in the second layer, in which the incoming 784 weights are reshaped into a 28x28 pixel image; the weights shown are typical of MNIST-trained networks.

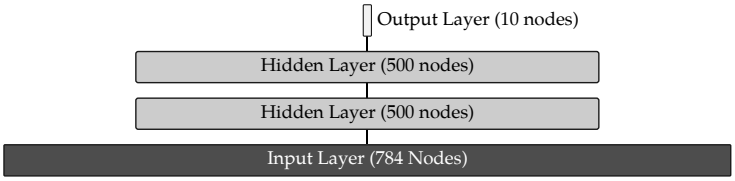


Figure 2.3: Visualization of the network configuration used for the MNIST task.

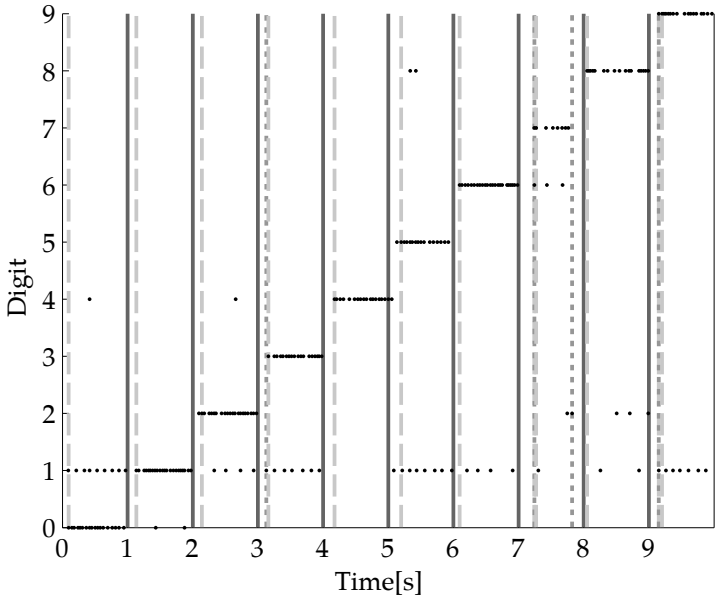


Figure 2.4: Example real-time run of digit identification, with an output spike represented by a black dot. Each digit was presented in order, from zero to nine, for a duration of one second during which 1000 spikes were presented; the probability of an input spike for a given pixel is proportional to the pixel's intensity. The winning digit is chosen according to an exponentially decaying histogram ($\tau = 0.11$s); the dark dotted line indicates a transition to a selection of an incorrect winning digit, while the lighter dashed lines indicate a transition to the correct choice for that digit. For the trial shown here, the average time to transition to a newly selected digit after a change in the input digit was 0.152 seconds.

performs the task with doubles (8 bytes) while Minitaur uses just 2 bytes for a neuron's weight. To ameliorate this, future versions can have a reduced-accuracy training paradigm to train weights that balance each other more accurately using less-precise representations, rather than simply truncating the more accurate representations.

The output behaviour of an example real-time execution of the Minitaur system on the MNIST classification task can be found in Fig. 2.4.

2.2.9 *Newsgroups Dataset Classification Performance*

Furthermore, to demonstrate the runtime configurability of Minitaur, a non-visual large dataset was selected in addition to the standard MNIST task. The twenty newsgroups dataset is a collection of approximately 20,000 documents evenly partitioned across 20 newsgroup forums, collected in 1995 [42] and commonly used in text classification and clustering. As shown in Table 2.3, several of the document types are very closely related, while others are more distant. The documents are presented in a bag-of-words model: Each document is represented with a sparse vector of word counts.

[42] K. Lang. "Newsweeder: Learning to filter netnews". In: *Proceedings of the Twelfth International Conference on Machine Learning*. 1995, pp. 331–339

Typically, the word counts of these documents are transformed by applying the Term frequency-inverse document frequency (TF-IDF) transform to control for commonly used words and to identify the more salient usage of infrequent words. This preprocessing step, however, requires both additional transformations and knowledge of the complete dataset. Minitaur is designed to save computation time and the system should operate with as little preprocessing and unnecessary computation as possible. Furthermore, word counts must be transformed into events to be used in the system.

In the approach used here, an event corresponding to each word is emitted whenever that word is encountered in text. Time no longer has a concrete meaning in this domain so it was chosen to assign random, small time steps to each additional word spike. In this way, the network represents the semantic context of the document which gradually either decays away or accumulates according to the types of words that are presented.

The network used in this case is a simple two-layer network of size 10,000-20, yielding a network of 10,020 neurons and 200,000 synapses. The number of neurons in the input layer is equal to the number of words used (10,000 in this experiment), and the number of neurons in the output layer equals the number of distinct classes (20 distinct newsgroups). The network was trained using standard backpropagation combined with dropout [43].

[43] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. "Improving neural networks by preventing co-adaptation of feature detectors". In: *arXiv preprint arXiv:1207.0580* (2012)

Without transforming the input using TF-IDF, word counts were emitted as word spike counts, and using the standard 60%-train, 40%-test split on these documents, the Minitaur system achieved 71% classification accuracy using the 10,000 most frequently used words in the dataset. The breakdown by category with the most common words can be found in Table 2.3.

The PSCs/sec performance of both the CPU and the Minitaur system decreased with the newsgroup classification task. A given news item to be classified may only have 100 non-common words, and the fanout for each of these words is only 20. This means that Minitaur cannot fully use all 32 cores during computation of neuron updates. In addition, the system has suboptimal caching from the significant weight convergence (10,000 nodes to 20), which limits the number of SRC neurons that can have cached weights. The CPU-based approach suffers as well because the parallelism

Newsgroup	Top Positive Terms	Perf
talk.religion.misc	order, bull, brian	44.22%
talk.politics.misc	kaldis, cramer, tax	46.45%
sci.electronics	circuit, electronics, hc	56.74%
comp.sys.ibm.pc.hardware	gateway, dx, bus	61.73%
comp.windows.x	motif, server, widget	61.79%
rec.autos	car, cars, dealer	64.30%
comp.os.ms-windows.misc	windows, win, cica	65.47%
comp.sys.mac.hardware	mac, apple, powerbook	66.58%
comp.graphics	graphics, image, polygon	66.84%
soc.religion.christian	rutgers, athos, christ	70.10%
talk.politics.guns	gun, guns, weapons	70.33%
sci.med	disease, doctor, msg	70.48%
alt.atheism	keith, atheism, mathew	72.64%
talk.politics.mideast	israel, israeli, turkish	74.47%
sci.space	space, orbit, launch	76.02%
misc.forsale	sale, offer, shipping	78.80%
rec.sport.baseball	baseball, phillies, runs	82.62%
rec.motorcycles	dod, bike, motorcycle	86.90%
rec.sport.hockey	hockey, nhl, playoff	87.47%
sci.crypt	encryption, clipper, key	87.59%

Table 2.3: Newsgroup classification

between trials is low; a given news item may only have 100 non-common words while another might have 10,000, so the CPU is not able to parallelize as many trials simultaneously as in the MNIST task.

2.2.10 *System Performance*

The current design has a benchmarked USB-to-USB latency of 236 μs (averaged over 10,000 trials), which is primarily dominated by the latency of the operating system issuing USB read and writes. Minitaur was benchmarked at processing 585 kEvts/sec or one input spike every 1.71 μs with all memory fetches drawing from local cache. With each input spike causing 32 PSCs (fully utilizing the parallel cores), Minitaur processed 18.73 million PSCs per second at its peak speed.

2.2.11 *Initial Response and Additional Accuracy*

Minitaur can be used to abort a computation early when sufficient accuracy is reached. When operating on a fixed input, event-based computation is a process of refinement rather than a static computation. Sequential input events add information to the system, and the system accumulates evidence over time to arrive at a more accurate answer.

This implies that an embodied robotic platform using Minitaur could use Minitaur's initial output after a very low response time to achieve a low-quality guess, or pay a small time cost to allow subsequent processing to increase the accuracy of that guess. In the MNIST task, 59.2% of the first output spikes (not shown) in-

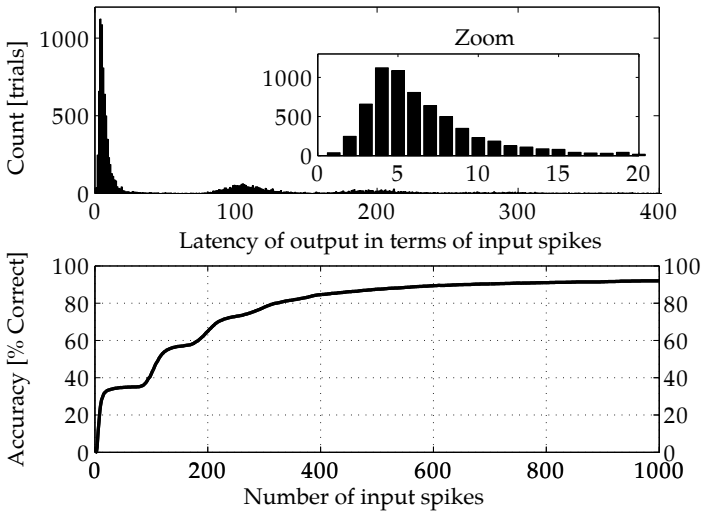


Figure 2.5: Increasing accuracy with additional information, using the complete 10,000 digits in the MNIST test set. For an event-based system the natural unit of time is number of input events, not seconds; each input event refines the answer estimate in the same way a long exposure time or multiple frames accumulates evidence in a time-stepped model. Moreover, latency is measured using input events because the system cannot produce an answer without accumulated information added to the system. The top plot shows a histogram of latency until the first output spike; most trials produce a result spike after 4 input spikes (as seen in the zoomed-in inset), but some trials can take hundreds of inputs to produce their first output spike. The bottom plot shows the effect of adding more events in the MNIST task; additional spikes cause the accuracy to asymptotically approach a 92% value.

dicated a correct answer occurring after the delay represented in the top half of Fig. 2.5. This would allow the system to make a low-accuracy guess after a very short delay. The bottom of Fig. 2.5 shows the increased accuracy of the system as the number of input events increases.

Both early-abort and longer-refinement use cases have obvious applications in robotics, and the freedom to choose at runtime is a major advantage of the Minitaur system.

2.2.12 *Noise Robustness and Indecision*

With the fallibility of sensors in general, and the likelihood of unexpected events in real-world datasets, robustness to noise is a significant part of designing a real-time event-driven system. To test the robustness of the system to noise, the MNIST dataset was employed with varying noise levels. As before, spikes are drawn from the image with probability proportional to pixel intensity. Then, a percentage of spikes are subsequently replaced with spikes from random pixels, drawn uniformly from the pixel space, and the accuracy of the system is calculated (Fig. 2.6). As can be seen in Fig. 2.7, the system is very robust to noise due to the weights of the Restricted Boltzmann Machine, which act to denoise the input by keeping only significant features as events propagate through the layers. Interestingly, the number of output spikes drops dramatically with increased noise; when receiving 90% noise, the system will average just over two output spikes for 1000 input spikes. The decrease of spikes has practical advantages as well; a downstream system using the output of Minitaur will be signaled with fewer spikes since Minitaur is less confident of its result.

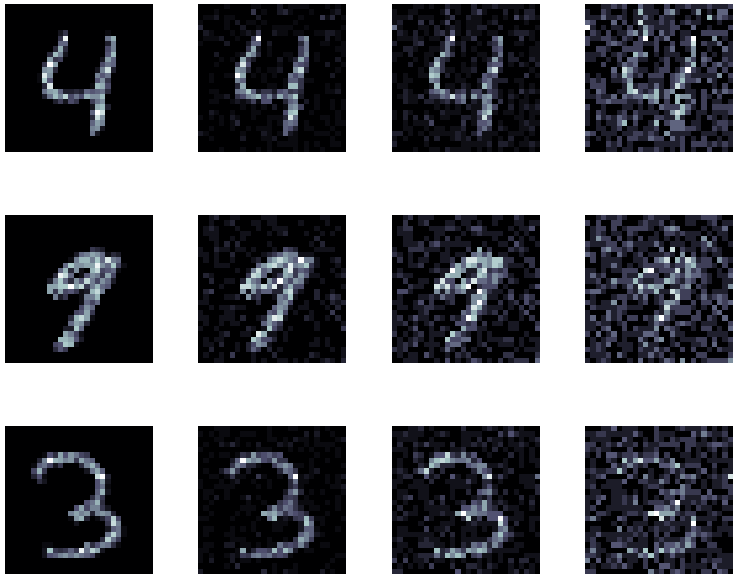


Figure 2.6: Visualization of 3 digits from the MNIST dataset with noise added. Shown here, from left to right, are 0% noise, 30% noise, 55% noise, and 80% noise for example handwritten digits 4, 9, and 3. Noise spikes were drawn uniformly from the pixel space and used to replace informative spikes.

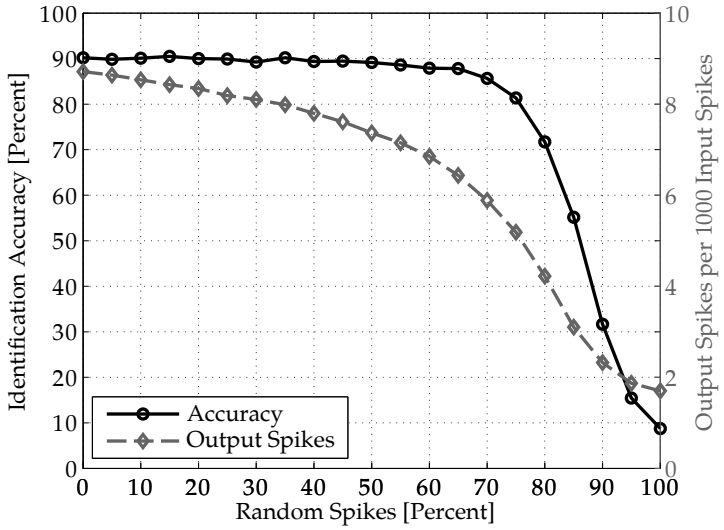


Figure 2.7: System performance is robust to noise. Even when the input is only 20% signal and 80% noise, the event-driven system still correctly classifies the digits with more than a 70% success rate. This is largely due to the robustness of RBMs to uniform noise; since no particular distribution is favored by uniform noise, it does not strongly affect the result. The increased noise of the data does create more indecision in the result; the number of output spikes drops dramatically with increased noise and accounts for the falling accuracy.

2.2.13 *Summary of FPGA Implementation*

The previous sections have introduced the Minitaur spiking network accelerator. In addition to the system's performance of 18.73 million post-synaptic currents per second, it consumes just 1.5W of power, enabling it to be used in embedded robotics applications. The system records 92% accuracy on the MNIST handwritten digit classification and 71% accuracy on the 20 newsgroups classification dataset. With proper weights, the system is remarkably robust to noise. Additionally, knowledge about the output spikes can be used to determine how difficult a task is, and to weigh the confidence of the output accordingly.

A significant challenge of using this system right now, however, is the dearth of effective training methods for LIF spike-based systems. Various approaches for learning the weights for spike-based LIF networks, in particular DBNs, are being explored [44] especially where such networks prevent traditional training regimens of real-

[44] Peter O'Connor, Daniel Neil, Shih-Chii Liu, Tobi Delbruck, and Michael Pfeiffer. "Real-time classification and sensor fusion with a spiking Deep Belief Network". In: *Frontiers in Neuroscience* 7 (2013)

valued sigmoidal activation functions and backpropagation. Further work on event-based learning is needed to improve training and runtime accuracy significantly.

2.3 SpiNNaker: An Optimized Hardware Implementation

After the fully-custom hardware investigation of Minitaur on FPGA, the next task was to examine a slightly more constrained design, closer to current computing infrastructures, and to study what algorithmic optimizations could remain viable given a system designed to optimize the computation of spiking neurons. One particularly attractive platform for this is the SpiNNaker project [45], a platform for spiking neural network computations, and so a collaboration was begun to produce an even more efficient implementation of a Deep Belief Network. While full control of the hardware architecture is available on FPGA, FPGAs are nonetheless not a particularly efficient substrate for hardware. Far more efficient would be a pure ASIC implementation, and the modified ARM cores that the SpiNNaker project runs on had been built specifically to perform digital computation, asynchronously, and with extremely high-performance data links to carry spike information between neurons. By leveraging this platform, designed to enable low-latency and low-power massively parallel large-scale simulations of spiking neurons in real-time, the work aimed to build a DBN that could be much more efficient than either a CPU or FPGA implementation. Indeed, while the energy efficiency of a laptop CPU was 1.68 MSops/W in real-time[46], and the energy efficiency of Minitaur on MNIST was measured to be between 12.48 and 18.73 MSops/W, a single SpiNNaker board with 48 SpiNN-5 component chips, which are the building blocks for creating larger SpiNNaker systems, provides up to 54.27 MSops/W in real-time [47].

The conversion process of the DBN for SpiNNaker also produced a model of power consumption for the SpiNNaker boards [48]:

$$P_{tot} = P_I + P_B + (P_N \cdot n) + (P_S \cdot s) \tag{2.2}$$

where P_I is the power dissipated by a SpiNNaker chip after the booting process with no loaded applications, P_B the baseline power dissipated by the kernel and Application programming interface (API), P_N is the power required to simulate a LIF neuron with a standard 1 ms timestep, n are the number of neurons, P_S is the energy consumed per synaptic event, and s are the total number of synaptic events. The parameter P_N was experimentally measured through a benchmark network varying the number of neurons, and disabling and enabling the transmission of spike events allows measurements of P_S. This fit explicitly points out the effect of scaling power load with "computational" load $P_S \cdot s$ which represents the firing rates of the neurons. Indeed, this model does well describe

[45] S.B. Furber, F. Galluppi, S. Temple, and L.A Plana. "The SpiNNaker Project". In: *Proceedings of the IEEE* 102.5 (May 2014), pp. 652–665. ISSN: 0018-9219. DOI: 10.1109/JPROC.2014.2304638

[46] Daniel Neil and S-C Liu. "Minitaur, an event-driven FPGA-based spiking network accelerator". In: *IEEE Trans on Very Large Scale Integration (VLSI) Systems* 22.12 (2014), pp. 2621–2628

[47] E. Stromatias, F. Galluppi, C. Patterson, and S. Furber. "Power analysis of large-scale, real-time neural networks on SpiNNaker". In: *Proceedings of 2013 International Joint Conference on Neural Networks (IJCNN)*. Aug. 2013, pp. 1–8. DOI: 10.1109/IJCNN.2013.6706927

[48] Evangelos Stromatias, Daniel Neil, Francesco Galluppi, Michael Pfeiffer, Shih-Chii Liu, and Steve Furber. "Scalable Energy-Efficient, Low-Latency Implementations of Spiking Deep Belief Networks on SpiNNaker". In: *Proceedings of the 2015 IEEE International Joint Conference on Neural Networks (IJCNN)*. 2015, pp. 1–8. DOI: 10.1109/IJCNN.2015.7280625

the power consumption of SpiNNaker and demonstrates the scaling behaviour [49] desired to grow towards very large and powerful networks while remaining event-driven.

However, in order to be executed on hardware platforms with fixed precision, approximations were necessary to adjust these models from full-precision floating-point parameters to a lower-precision. To ensure this did not introduce difficulties, following work [50] investigated the effect of these reduced precision parameter optimizations. This serendipitously led to an important method for training deep neural networks, as well as uncovering a principle which continues to be exploited today.

[49] Evangelos Stromatias, Daniel Neil, Francesco Galluppi, Michael Pfeiffer, Shih-Chii Liu, and Steve Furber. "Scalable Energy-Efficient, Low-Latency Implementations of Spiking Deep Belief Networks on SpiNNaker". In: *Proceedings of the 2015 IEEE International Joint Conference on Neural Networks (IJCNN)*. 2015, pp. 1–8. DOI: 10.1109/IJCNN.2015.7280625

[50] Evangelos Stromatias, Daniel Neil, Michael Pfeiffer, Francesco Galluppi, Steve B Furber, and Shih-Chii Liu. "Robustness of spiking Deep Belief Networks to noise and reduced bit precision of neuro-inspired hardware platforms". In: *Frontiers in Neuroscience* 9 (2015)

2.4 *Low-precision Approximations for Hardware Systems*

The following excerpt first appeared in [51].

To reduce the precision of the weights of a DBN, there is the straightforward method of rounding the weights after training to a given precision. However, beyond the method of rounding the weights of a DBN after training has been completed, this work introduces two additional approaches to create the lower-precision weights, and optimize the performance for low-precision simulations. Intuitively, the motivation for these novel methods arises from the idea that networks that incorporate knowledge about the eventual low-precision representation of the hardware during training may be able to perform better under those low-precision conditions than networks that have been optimized under the assumption of higher precision.

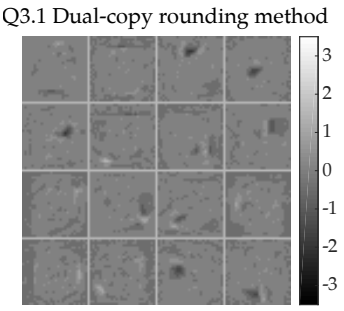
[51] Evangelos Stromatias, Daniel Neil, Michael Pfeiffer, Francesco Galluppi, Steve B Furber, and Shih-Chii Liu. "Robustness of spiking Deep Belief Networks to noise and reduced bit precision of neuro-inspired hardware platforms". In: *Frontiers in Neuroscience* 9 (2015)

The first proposed method, called *iterative rounding*, is similar to the fixed-point method mentioned in [52], in which the result of a computation is rounded whenever it is stored. The method originally proposed in [53], however, refers to the case when the forward pass of computing activities of neurons in all layers, and the computation of gradients for learning, are performed with full precision and only the weight is kept in reduced precision. For iterative rounding, the full-precision weight update is calculated from the contrastive divergence algorithm, and applied directly to the low-precision weights. After the full-precision weight update has been applied, the value is then rounded to the closest low-precision representation of that weight and stored.

[52] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. "Low precision arithmetic for deep learning". In: *arXiv preprint arXiv:1412.7024* (2014)

[53] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. "Low precision arithmetic for deep learning". In: *arXiv preprint arXiv:1412.7024* (2014)

However, one challenge with this approach is that the gradient update may be too small to change the values of low-precision weights. To address this potential difficulty, this paper introduces a key, novel method called *dual-copy rounding*, which uses both a temporary low-precision and a permanent high-precision representation of weights. This method incorporates knowledge about the eventual low-precision representation of the tested network as well as supporting the small, but highly important accumulation of error gradients over multiple iterations. For this method, two copies



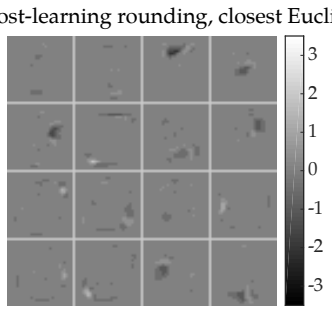


Figure 2.8: Impact of different rounding methods during learning on learned weight representations. Comparison of first-layer weights in networks trained with the dual-copy rounding method (left) and the post-learning rounding method (right). The weights shown here are representative samples from 16 clusters of weight vectors in the learned dual-copy rounding weight matrix. On the right, the weights from the post-learning rounding weight matrix that are most similar to these chosen weights are displayed. The dual-copy rounding method is able to preserve much more fine structure, compared to simply rounding the network weights after training, and is thus more suitable for training networks that will be executed with lower bit precision weights.

of the weight matrix W are maintained during training: a high-precision weight matrix (W_H) and a low-precision weight matrix (W_L), which is stored in Q$m.f$ numerical format. Learning proceeds as before, but the activities of the hidden layer and the visible layer after sampling are obtained using the low-precision weights W_L. The contrastive divergence update for W_H is thus parameterized as $\Delta w(W_L)$, and after the update both weight matrices are processed as

$$W_H = \begin{cases} -2^m & \text{where } W_H \leq -2^m \\ W_H & \text{where } -2^m < W_H < 2^m \\ 2^m & \text{where } W_H \geq 2^m \end{cases} \tag{2.3}$$

$$W_L = \text{round}(2^f \cdot W_H) \cdot 2^{-f} \tag{2.4}$$

where 2^m represents the largest possible value that can be stored in the Q$m.f$ format. Importantly, note that the low-precision weight matrix W_L is used to sample from the network, while the weight update is applied to the higher-precision representation W_H, and W_L is obtained via rounding. As in standard contrastive divergence, the weight update is calculated from the difference of pairwise correlations of the data-driven layers and the model-driven sample layers. Here, although the activations are calculated from the low-precision weights, the updates are accumulated in the high-precision weights. Then, the weights are checked to be within the maximum bounds of the given resolution (Eq. 2.3) for the given fixed-point precision. Finally, the weights are copied over into the low-precision matrix (Eq. 2.4). The learning can then proceed for another iteration, using the new updated low-precision weight matrix W_L. The additional cost of dual-copy rounding is to store a second weight matrix in memory, which is typically not a limiting factor for off-chip learning.

For qualitative differences, observe the weights shown in Figure 2.8. In order to show representative samples, the learned weights in the first layer from the dual-copy rounding method were clustered into 16 categories, and the post-learning rounding method weights with the closest Euclidean distance to these cluster exemplars were identified and plotted on the right. The dual-copy

rounding method preserves significantly more fine-grained structure, which would be lost with other rounding methods.

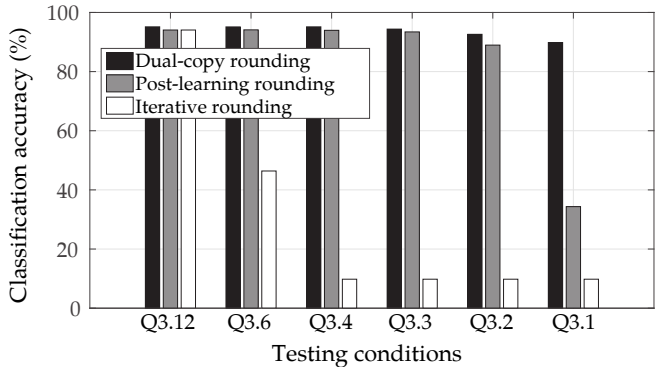


Figure 2.9: Effectiveness of the dual-copy rounding weight training paradigm. Training at full precision and later rounding performs consistently worse than the dual-copy rounding method introduced in this paper. Rounding the weights during training can prevent learning entirely at low-precision regimes. The results show averages of five independent runs with different random seeds.

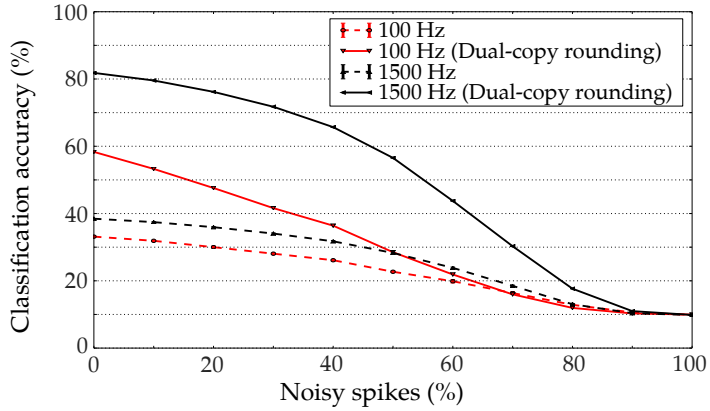


Figure 2.10: Increase in classification accuracy of a spiking DBN with Q3.1 precision weights due to the dual-copy rounding method for input rates of 100 Hz and 1500 Hz. Results over 4 trials.

For a quantitative analysis of the differences in performance, the classification accuracy in the MNIST task using different bit precisions and different rounding methods was measured. For performance reasons, the classification performance for this section unlike the other sections in this paper occurred in rate-based (non-spiking) conditions, but using the same training procedure for spiking neurons.

As there was no performance loss in the Q3.12 representation compared to full double-precision, this was taken as the full-precision reference point. Figure 2.9 shows the effect of the three investigated training methods on the classification accuracy, when testing the weight matrix at different levels of bit precision. Rounding a high-precision weight matrix does work effectively, but can fail for lower-precision weights. Unfortunately, the iterative rounding method of training works extremely poorly for low-precision cases; the weight update size is simply less than the precision of the weight, so learning halts entirely after the error gradient falls below a certain threshold.

Impressively, however, Figure 2.9 shows the clear advantage of the dual-copy training. Across all precision levels, incorporating information about the lower-precision weight matrix into the training yields noticeable and consistent improvements. This improvement

increases as precision decreases, so that at the Q3.1 representation level, where there is only a single sign bit and three bits of representation per weight, the network is able to achieve an impressive 91.35% accuracy in the highest-performing case. Under spiking conditions, the performance of this network drops to 82%, Figure 2.10. While this is substantially lower than the rate-based performance, it is still twice the accuracy of the default post-learning rounding method, and future work will determine improved ways to maintain the rate-based performance under spiking conditions.

2.5 *Lessons Learned from Hardware Spiking Systems*

So WHAT WAS LEARNED from this initial hardware investigation? Power efficiency was the primary optimization target, though the hardware still needed to be powerful enough to sufficiently run this large, complex networks.

As pointed out explicitly by the power consumption equation Eq. 2.2, power increases linearly with the number of neurons used (as expected), and also with the number of input synaptic events. Though that makes sense from a data-driven perspective, it is markedly different than traditional machine learning which has a relatively fixed cost for computation; data-driven computation scaling implies that inputs that produce higher intermediate firing rates will consume more power than inputs that produce lower firing rates. This should be considered in later chapters in which algorithms are designed.

In Minitaur, a major bottleneck on the system was the time it took to cache states and weights. The design methodology of the biological brain favors co-locating computation and memory, but modern electronic design processes prevent memory and computation from being collocated. Adding more memory locally increases the silicon area of the design, which increases the cost and hinders adoption. It also requires moving data further, consuming more power, and can slow down the clock speed of the device as it has to cover more area. Instead, the design must find the optimal design between the massively parallel, widely-connected, and low-power neurons of biology, and the opportunities afforded in a roughly 2D mesh of silicon. New state-of-the-art deep accelerators have since taken up this mantle and are further explored in Sec. 8.

As well as the storage of weights and states, the unpredictable access caused by the asynchronous and environmentally-triggered data severely hindered the efficiency of these systems. While caching is one method that was able to ameliorate these effects (fast-spiking neurons are likely to spike again in the future), it placed these designs at a greater disadvantage compared to traditional hardware accelerators that strongly order pipelines and data reuse to minimize overhead.

Nonetheless, this direction of the investigation is a successful

one. The TrueNorth chip [54] succeeded greatly at creating an event-driven asynchronous hardware platform, and while Minitaur offered an order-of-magnitude greater efficiency than a laptop CPU's 1.68 MSOps/W at 18.73 MSOps/W, and while the SpiNNaker implementation pushed that to 54.27 MSOps/W, the TrueNorth chip was able to achieve over 46GSops/W, increasing the efficiency by three orders of magnitude, emphasizing the low-power capability of a spiking system. The TrueNorth chip remains to date one of the most efficient platforms for event-driven inputs, and yet can widely support a variety of models. The work performed here [55,56] also translates successfully to these larger and newer platforms.

Additionally, the process of converting an event-driven deep neural network to hardware serendipitously uncovered a principle which will be exploited throughout this thesis: deep neural networks are overly powerful, to their own detriment. By decreasing the weight resolution dramatically in [57], not only did the network achieve greater accuracy, but it was more robust to noise. The power of deep neural networks should be constrained, both to achieve greater efficiency and also to achieve greater accuracy.

However, the unfortunate truth is that these spiking network implementations are achieving sufficient accuracy to be taken seriously by the greater machine learning community. MNIST was rapidly becoming considered a solved problem with greater than 99.7% accuracy [58] achieved, while deep spiking networks had trouble reaching greater than 96% accuracy [59], and even lower in hardware implementations [60]. While that 3% may not seem significant, an asymptotic approach means that the 99% was an incredibly challenging and elusive target. No spiking network had yet achieved scores close to that, which is problematic in the machine learning community that relies heavily on benchmarks. Spiking neural networks need to increase their accuracy significantly.

[54] Paul A Merolla, John V Arthur, Rodrigo Alvarez-Icaza, Andrew S Cassidy, Jun Sawada, Filipp Akopyan, Bryan L Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, et al. "A million spiking-neuron integrated circuit with a scalable communication network and interface". In: *Science* 345.6197 (2014), pp. 668–673

[55] Evangelos Stromatias, Daniel Neil, Michael Pfeiffer, Francesco Galluppi, Steve B Furber, and Shih-Chii Liu. "Robustness of spiking Deep Belief Networks to noise and reduced bit precision of neuro-inspired hardware platforms". In: *Frontiers in Neuroscience* 9 (2015)

[56] Evangelos Stromatias, Daniel Neil, Francesco Galluppi, Michael Pfeiffer, Shih-Chii Liu, and Steve Furber. "Scalable Energy-Efficient, Low-Latency Implementations of Spiking Deep Belief Networks on SpiNNaker". In: *Proceedings of the 2015 IEEE International Joint Conference on Neural Networks (IJCNN)*. 2015, pp. 1–8. DOI: 10.1109/IJCNN.2015.7280625

[57] Evangelos Stromatias, Daniel Neil, Michael Pfeiffer, Francesco Galluppi, Steve B Furber, and Shih-Chii Liu. "Robustness of spiking Deep Belief Networks to noise and reduced bit precision of neuro-inspired hardware platforms". In: *Frontiers in Neuroscience* 9 (2015)

[58] Li Wan, Matthew Zeiler, Sixin Zhang, Yann L Cun, and Rob Fergus. "Regularization of neural networks using dropconnect". In: *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*. 2013, pp. 1058–1066

[59] Peter O'Connor, Daniel Neil, Shih-Chii Liu, Tobi Delbruck, and Michael Pfeiffer. "Real-time classification and sensor fusion with a spiking Deep Belief Network". In: *Frontiers in Neuroscience* 7 (2013)

[60] Daniel Neil and S-C Liu. "Minitaur, an event-driven FPGA-based spiking network accelerator". In: *IEEE Trans on Very Large Scale Integration (VLSI) Systems* 22.12 (2014), pp. 2621–2628

3
Bringing in the State-of-the-Art from Deep Learning

To BEGIN investigating methods to improve event-based Deep Learning, it is important to recapitulate the prior work that has been accomplished. As mentioned in previous sections, the initial investigations into fully event-driven deep networks began with investigations into spiking Deep Belief Networks [1]. Section 3.1 introduces the key concepts and formulation behind that work. Pushing beyond Deep Belief Networks, Sections 3.2.1-3.6.3 perform a detailed analysis of the cause of performance loss in the conversion from standard networks to spiking networks, and results in the development of key algorithms for conversion from state-of-the-art convolutional networks to spiking networks with near-lossless conversion. Additionally, the algorithms permit an acceleration of classification time, requiring only a few spikes to achieve high accuracy. Finally, Section 3.7 concludes the chapter with the advances gained and how they relate to the principles of this thesis.

[1] Peter O'Connor, Daniel Neil, Shih-Chii Liu, Tobi Delbruck, and Michael Pfeiffer. "Real-time classification and sensor fusion with a spiking Deep Belief Network". In: *Frontiers in Neuroscience* 7 (2013)

3.1 *Prior work: Deep Belief Networks and Spiking Networks*

Text in this section originally appeared previously [2], and training of DBNs targeting a spiking network implementation is described in detail in [3]. The key idea of using a spike-based network instead of what is termed a Analog Neural Network (ANN), a conventional neural network, is to use spike rates as an approximation of the real-valued analog signal within an ANN. Within the probabilistic DBN framework, the ratio of the firing rates of Leaky Integrate-and-Fire neurons to their maximum firing rates represent the activation probabilities used in the training algorithm. The more precisely a continuous, analog rate-based approximation matches the true firing rates of a neuron, the more precisely the spiking version will match the analog value.

[2] Evangelos Stromatias, Daniel Neil, Michael Pfeiffer, Francesco Galluppi, Steve B Furber, and Shih-Chii Liu. "Robustness of spiking Deep Belief Networks to noise and reduced bit precision of neuro-inspired hardware platforms". In: *Frontiers in Neuroscience* 9 (2015)

[3] Peter O'Connor, Daniel Neil, Shih-Chii Liu, Tobi Delbruck, and Michael Pfeiffer. "Real-time classification and sensor fusion with a spiking Deep Belief Network". In: *Frontiers in Neuroscience* 7 (2013)

The so-called Siegert approximation [4] is used here, which approximates the output firing rate of a LIF neuron receiving both inhibitory and excitatory inputs. Let $\vec{\rho}_i$ and $\vec{\rho}_e$ be the vectors of inhibitory and excitatory input rates, and $(\vec{w}_i, \vec{w}_e)$ be the corresponding weights. In order to compute the expected output rate of the LIF neuron, a number of auxiliary variables first needs to be

[4] F. Jug, M. Cook, and A. Steger. "Recurrent competitive networks can learn locally excitatory topologies". In: *Proceedings of 2012 International Joint Conference on Neural Networks (IJCNN)*. June 2012, pp. 1–8. DOI: 10.1109/IJCNN.2012.6252786

computed. For completeness, the full equations are provided here, but refer to previous work for the derivation and interpretation of each variable [5,6]:

$$\mu_Q = \tau \sum(\vec{w}_e\vec{\rho}_e + \vec{w}_i\vec{\rho}_i) \qquad \sigma_Q^2 = \frac{\tau}{2}\sum(\vec{w_e^2}\vec{\rho}_e + \vec{w_i^2}\vec{\rho}_i)$$
$$\Upsilon = V_{\text{reset}} + \mu_Q \qquad \Gamma = \sigma_Q$$
$$k = \sqrt{\tau_{\text{syn}}/\tau} \qquad \gamma = |\zeta(1/2)|$$

Here, τ_{syn} denotes the synaptic time constant (for our purposes considered to be zero), and ζ is the Riemann zeta function. Then the average firing rate ρ_{out} of the neuron with reset potential V_{reset}, threshold voltage V_{thresh}, and refractory period T_{ref} can be computed as [7]

$$\rho_{\text{out}} = \left(T_{\text{ref}} + \frac{\tau}{\Gamma}\sqrt{\frac{\pi}{2}}\cdot \int_{V_{\text{reset}}+k\gamma\Gamma}^{V_{\text{thresh}}+k\gamma\Gamma} \exp\left[\frac{(u-\Upsilon)^2}{2\Gamma^2}\right]\cdot\left[1+\text{erf}\left(\frac{u-\Upsilon}{\Gamma\sqrt{2}}\right)\right]du\right)^{-1}. \tag{3.1}$$

While complex, this approximation of firing rates allows a direct translation between the analog activation probabilities required for Contrastive Divergence (CD) training and the resulting firing rates of a spiking neuron with those weights. During training of the spiking DBN, the Siegert approximation is used as the nonlinearity of the neuron instead of a sigmoidal function. The predicted rate ρ_{out} in (Eq. 3.1) can be converted into a probability by normalizing with the maximum firing rate $1/T_{\text{ref}}$. This allows sampling the activation probabilities, as is done in standard contrastive divergence learning with with continuous-valued units. Specifically, the weight update in contrastive divergence for spiking networks computes the data- and model-driven activities of the visible and hidden layer using the Siegert approximation, and then computes the weight update as usual in RBM training. Let V_{data} be the activity of the visible units driven by the input data (or activity of the hidden layer below). Then the data-driven activity of the hidden layer, given the full weight matrix W connecting the visible and hidden layer, is

$$H_{\text{data}} = \rho_{\text{out}}(V_{\text{data}}, W)\cdot T_{\text{ref}}$$

The model-driven activity of the visible and hidden layers, obtained via Gibbs sampling, is then given as

$$V_{\text{model}} = \rho_{\text{out}}(H_{\text{data}}, W^T)\cdot T_{\text{ref}}, \qquad H_{\text{model}} = \rho_{\text{out}}(V_{\text{model}}, W^T)\cdot T_{\text{ref}}$$

and the weight update Δw is

$$\Delta w = \alpha\cdot(H_{\text{data}}^T V_{\text{data}} - H_{\text{model}}^T V_{\text{model}}), \tag{3.2}$$

where α is the learning rate. Using this formulation for the activation function (the Siegert function) allows reusing state-of-the-art

[5] A. J. F. Siegert. "On the first passage time probability problem". In: *Physical Review* 81.4 (1951), p. 617

[6] F. Jug, M. Cook, and A. Steger. "Recurrent competitive networks can learn locally excitatory topologies". In: *Proceedings of 2012 International Joint Conference on Neural Networks (IJCNN)*. June 2012, pp. 1–8. DOI: 10.1109/IJCNN.2012.6252786

[7] F. Jug, M. Cook, and A. Steger. "Recurrent competitive networks can learn locally excitatory topologies". In: *Proceedings of 2012 International Joint Conference on Neural Networks (IJCNN)*. June 2012, pp. 1–8. DOI: 10.1109/IJCNN.2012.6252786

training tools and methodologies for training spiking networks in a straightforward way. First, the parameters for the target application are chosen (V_{reset}, V_{thresh}, T_{ref}). Next, the corresponding Siegert activation function is given as the activation function for the neurons to transform input currents to spike rates; finally, after training, the parameters and weights are kept unchanged and the units generate Poisson spike trained with rates computed by the Siegert formula (Eq. (3.1)). In [8] it was shown that this results in equivalent spiking implementations of RBMs and DBNs, which perform similarly to conventional networks with the same architecture.

3.2 *Feed-forward Network Conversion*

However, this was shown to closely, but not precisely, approximate the event rates. Certain assumptions that are required to use the Siegert model (for example, Gaussian-distributed weights) were only approximately yet not strictly true in practice. Moreover, purely feedforward models such convolutional neural networks began to outperform generative and probability-based models like Deep Belief Networks. New techniques needed to be developed to port these recent developments in machine learning over to spiking networks. The following text was originally published previously [9,10].

3.2.1 *Motivation for Feed-forward Network Conversion*

Feed-forward deep neural network architectures, such as convolutional neural networks (CNNs) [11] and fully-connected feed-forward neural networks [12], are currently the most successful architectures for natural image classification. They have achieved record-breaking results for problems such as handwriting recognition [13], scene labeling [14], the CIFAR benchmark [15], the ImageNet benchmark [16], and many others. Deep neural network architectures, which are loosely inspired by hierarchies of cortical visual information processing [17], have seen increasing success in recent years due to the availability of more powerful computing hardware, larger datasets, and improved training algorithms, which has enabled the training of much deeper networks, while avoiding problems of overfitting [18]. Despite their successes, the substantial computational cost of training and running deep networks has created a need for specialized hardware acceleration and new computational paradigms to enable the use of deep networks for real-time practical applications.

3.2.2 *Motivation for Deep Spiking Networks*

Algorithms for spiking deep neural networks have also become an increasingly active field of research. This has been driven both by the interest to build more biologically realistic neural network

[8] Peter O'Connor, Daniel Neil, Shih-Chii Liu, Tobi Delbruck, and Michael Pfeiffer. "Real-time classification and sensor fusion with a spiking Deep Belief Network". In: *Frontiers in Neuroscience* 7 (2013)

[9] Peter U Diehl, Daniel Neil, Jonathan Binas, Matthew Cook, Shih-Chii Liu, and Michael Pfeiffer. "Fast-Classifying, High-Accuracy Spiking Deep Networks Through Weight and Threshold Balancing". In: *International Joint Conference on Neural Networks (IJCNN)*. 2015

[10] This conversion work was done in equal part in a collaboration between the author and Peter U. Diehl.

[11] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324

[12] Dan Claudiu Ciresan, Ueli Meier, Luca Maria Gambardella, and Jürgen Schmidhuber. "Deep, big, simple neural nets for handwritten digit recognition". In: *Neural Computation* 22.12 (2010), pp. 3207–3220

[13] Dan Claudiu Ciresan, Ueli Meier, Luca Maria Gambardella, and Jürgen Schmidhuber. "Deep, big, simple neural nets for handwritten digit recognition". In: *Neural Computation* 22.12 (2010), pp. 3207–3220

[14] Clement Farabet, Camille Couprie, Laurent Najman, and Yann LeCun. "Learning hierarchical features for scene labeling". In: *IEEE Trans on Pattern Analysis and Machine Intelligence* 35.8 (2013), pp. 1915–1929

[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Proc. of NIPS*. 2012, pp. 1097–1105

[16] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. "OverFeat: Integrated recognition, localization and detection using convolutional networks". In: *arXiv preprint* 312.6229 (2013)

[17] Kunihiko Fukushima. "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position". In: *Biological Cybernetics* 36.4 (1980), pp. 193–202

[18] Jürgen Schmidhuber. "Deep learning in neural networks: An overview". In: *Neural Networks* 61 (2015), pp. 85–117

models, and by recent improvements and the availability of larger-scale neuromorphic computing platforms, which are optimized for emulating brain-like spike-based computation in dedicated analog or digital hardware [19,20,21,22]. Neuromorphic platforms can be orders of magnitude more efficient in terms of power consumption compared to conventional CPUs or GPUs for running spiking networks, and often permit distributed and asynchronous event-based computation, thereby improving scalability and reducing latencies. Furthermore, event-driven neuromorphic systems focus their computational effort on currently active parts of the network, effectively saving power on the rest of the network. They are therefore attractive as platforms to run large-scale deep neural networks in real-time and potentially support online learning [23,24,25]. These platforms are ideally driven by input from neuromorphic sensors such as silicon retinas [26] or cochleas [27], which create sparse, frame-free, and precisely timed streams of events, with substantially reduced latencies compared to frame-based approaches. Earlier work on spiking deep networks has thus focused on fast classification in event-based vision systems with CNNs and Deep-Belief Networks (DBNs) [28,29,30].

As introduced above, training of spiking deep networks typically does not use spike-based learning rules, but instead starts from a conventional ANN, fully trained with backpropagation, followed by a conversion of the rate-based model into a model consisting of simple spiking neurons. Theory has shown that Spiking Neural Networks (SNNs) are at least as computationally powerful as their analog counterparts [31], but practically it has proven difficult to come up with equivalent solutions. One approach used for example by [32] is to train spiking DBNs by using the Siegert mean-firing-rate approximation of LIF neurons to approximate probabilities during training.Another approach, used in [33], requires tuning of parameters such as leak and refractory period in the spiking network.In both cases, the spiking network suffers from considerable losses in classification accuracy, when compared to a non-spiking network of similar architecture.

Recently, [34] proposed a method for spiking CNN conversion that achieves significantly better performance than previous approaches, by taking the characteristic differences of spiking and non-spiking networks into account. The main challenges are the representation of negative values and biases in spiking neurons, which are avoided by using Rectified Linear Units (ReLUs) during training, and setting all biases to zero. Furthermore, the typical max-pooling operations of CNNs are replaced by spatial linear subsampling. Still, the resulting spiking CNN after conversion suffers from a small loss of performance.

[19] Paul A Merolla, John V Arthur, Rodrigo Alvarez-Icaza, Andrew S Cassidy, Jun Sawada, Filipp Akopyan, Bryan L Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, et al. "A million spiking-neuron integrated circuit with a scalable communication network and interface". In: *Science* 345.6197 (2014), pp. 668–673

[20] Ben Varkey Benjamin, Peiran Gao, Emmett McQuinn, Swadesh Choudhary, Anand R Chandrasekaran, J Bussat, Rodrigo Alvarez-Icaza, John V Arthur, PA Merolla, and Kwabena Boahen. "Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations". In: *Proceedings of the IEEE* 102.5 (2014), pp. 699–716

[21] S.B. Furber, F. Galluppi, S. Temple, and L.A Plana. "The SpiNNaker Project". In: *Proceedings of the IEEE* 102.5 (May 2014), pp. 652–665. ISSN: 0018-9219. DOI: 10.1109/JPROC.2014.2304638

[22] Daniel Neil and S-C Liu. "Minitaur, an event-driven FPGA-based spiking network accelerator". In: *IEEE Trans on Very Large Scale Integration (VLSI) Systems* 22.12 (2014), pp. 2621–2628

[23] G. Indiveri, E. Chicca, and R. Douglas. "A VLSI figurable network of integrate-and-fire neurons with spike-based learning synapses". In: (2004)

[24] X. Jin, A. Rast, F. Galluppi, S. Davies, and S.B. Furber. "Implementing spike-timing-dependent plasticity on SpiNNaker neuromorphic hardware". In: *Neural Networks (IJCNN), The 2010 International Joint Conference on*. IEEE. 2010, pp. 1–8

[25] Peter U Diehl and Matthew Cook. "Efficient implementation of STDP rules on SpiNNaker neuromorphic hardware". In: *2014 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2014, pp. 4288–4295

[26] Patrick Lichtsteiner, Christoph Posch, and Tobi Delbruck. "A 128× 128 120 dB 15 μs latency asynchronous temporal contrast vision sensor". In: *IEEE Journal of Solid-State Circuits* 43.2 (2008), pp. 566–576

[27] Shih-Chii Liu and Tobi Delbruck. "Neuromorphic sensory systems". In: *Current Opinion in Neurobiology* 20.3 (2010), pp. 288–295

[28] R. Serrano-Gotarredona and others. "CAVIAR: A 45k Neuron, 5M Synapse, 12G Connects/s AER Hardware Sensory–Processing– Learning–Actuating System for High-Speed Visual Object Recognition and Tracking". In: *IEEE Trans on Neural Networks* 20.9 (2009), pp. 1417–1438

3.3 Neural Network Architectures for Conversion

3.3.1 ReLU-Based Feed-Forward Neural Networks

In densely connected feed-forward Neural Networks (DenseNets), all neurons in the preceding layer are fully connected to the subsequent layers, with no intra-layer connections. Recent competitive results (e.g. [35]) have renewed the interest in this architecture. Initializing the extremely high-dimensional weight vectors of DenseNets in a good regime that preserves the error gradient, and regularizing the network to prevent overfitting, allows very high performance on standard test sets. The most recent improvements have come with the introduction of the dropout training technique (see section 3.3.3) in combination with ReLUs [36]. ReLUs are a type of nonlinearity which is applied to the weighted sum of inputs, and is described by

$$x_i = \max\left(0, \sum_j w_{ij} x_j\right) \quad , \tag{3.3}$$

where x_i is the activation of unit i, w_{ij} is the weight connecting unit j in the preceding layer to unit i in the current layer, and x_j is the activation of unit j in the preceding layer. By successively updating all the activations of a current layer based on the activations of the previous layer, the input is propagated through the network to activate the output label neurons.

Training proceeds according to standard error backpropagation, successively propagating an error gradient backwards through the layers by computing local derivatives to update individual weights and minimize the error. In these networks, the training process adjusts the randomly-initialized weight matrix describing the connections between the layers to minimize the overall error through stochastic gradient descent. Further details can be found in [37].

3.3.2 Convolutional Neural Networks

CNNs [38] are multi-layered feed-forward architectures in which feature detectors take the form of simple convolution kernels. Typically, a convolutional neural network is composed of alternating layers of convolution and spatial subsampling, with nonlinearities between subsequent iterations. Here, a convolutional layer generates a number of feature maps, which are obtained by convolving patches of the preceding layer with a set of kernels $\{W^k, k = 1 \dots n\}$. The resulting maps $\{x^k, k = 1 \dots n\}$ are given by

$$x^k = f\left(\sum_l W^k * x^l + b^k\right) \quad , \tag{3.4}$$

where f is the neuron's nonlinear activation function, x^l is the activation of the units of the preceding layer's activation map l, the $*$ symbol denotes a 2D valid-region convolution, and b^k is a bias term. As above, ReLUs are used, as described in equation 3.3 as the activation function f. The kernels only respond to a small rectangular patch of inputs, specified by W^k, which are repeated and

[29] Clément Farabet et al. "Comparison between frame-constrained fix-pixel-value and frame-free spiking-dynamic-pixel convNets for visual processing". In: *Frontiers in Neuroscience* 6 (2012)

[30] Peter O'Connor, Daniel Neil, Shih-Chii Liu, Tobi Delbruck, and Michael Pfeiffer. "Real-time classification and sensor fusion with a spiking Deep Belief Network". In: *Frontiers in Neuroscience* 7 (2013)

[31] Wolfgang Maass and Henry Markram. "On the computational power of circuits of spiking neurons". In: *Journal of Computer and System Sciences* 69.4 (2004), pp. 593–616

[32] Peter O'Connor, Daniel Neil, Shih-Chii Liu, Tobi Delbruck, and Michael Pfeiffer. "Real-time classification and sensor fusion with a spiking Deep Belief Network". In: *Frontiers in Neuroscience* 7 (2013)

[33] Jose Pérez-Carrasco and others. "Mapping from Frame-Driven to Frame-Free Event-Driven Vision Systems by Low-Rate Rate Coding and Coincidence Processing–Application to Feedforward ConvNets". In: *IEEE Trans on Pattern Analysis and Machine Intelligence* 35.11 (2013), pp. 2706–2719

[34] Yongqiang Cao, Yang Chen, and Deepak Khosla. "Spiking Deep Convolutional Neural Networks for Energy-Efficient Object Recognition". In: *International Journal of Computer Vision* (2014), pp. 1–13

[35] Dan Claudiu Ciresan, Ueli Meier, Luca Maria Gambardella, and Jürgen Schmidhuber. "Deep, big, simple neural nets for handwritten digit recognition". In: *Neural Computation* 22.12 (2010), pp. 3207–3220

[36] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: A simple way to prevent neural networks from overfitting". In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958

[37] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: A simple way to prevent neural networks from overfitting". In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958

[38] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324

moved over the whole input map. Convolutional layers are often followed by subsampling or pooling layers, whose units combine the responses of multiple feature detectors into one. While many choices exist for pooling layers, an averaging kernel is used here to enhance the portability of this CNN to an SNN. The activation of this averaging layer is identical to equation 3.4, except that the kernel weights W_{ij}^k are fixed to $1/\text{size}(W^k)$, where $\text{size}(W^k)$ is the number of pixels in the kernel.

CNNs reduce the data dimensionality by alternating between convolution and subsampling layers, while producing increasingly abstract features to describe the input. Additionally, the lower number of weights in the CNN compared to fully connected archictectures reduces the problem of overfitting. The output of the CNN is the concatenation of all feature maps in the final layer, which forms the input to a simple fully-connected neural network trained as a classifier. Just like DenseNets, training uses stochastic gradient descent via backprop to adjust the weights in the network, using weight sharing to learn W^k in the convolution layers, together with the weights for the final output layer. Further details can be found in [39].

[39] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324

3.3.3 *Dropout*

Overfitting is a well-known problem of large and deep neural networks. A successful method to avoid this problem is to employ regularizers, such as the recently proposed dropout technique [40]. Dropout randomly disables input units during learning, and thus avoids overspecialization and co-adaptation of hidden units. In this work, dropout is used in the activation function as a mask that randomly disables ReLU activations on a given trial, thereby effectively increasing the overall robustness of the network. A ReLU activation function with dropout is given by

[40] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: A simple way to prevent neural networks from overfitting". In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958

$$x_i = \begin{cases} \max\left(0, \sum_j w_{ij} x_j\right) & \text{with probability } d_r \\ 0 & \text{otherwise}, \end{cases} \tag{3.5}$$

where the variables are as in equation 3.3 with the addition of a dropout rate d_r. At every training iteration, a new random decision is made for each unit. In practice, a d_r value of 0.5 is often used to turn off half of the connections randomly in each training step.

3.4 *Spiking Neural Networks*

3.4.1 *Background*

In a conventional ANN, a whole input vector is presented at one time, and processed layer-by-layer, producing one output value. In an SNN, however, inputs are typically presented as streams of events, and neurons integrate evidence during the presentation, creating spikes to communicate information to subsequent layers, ul-

timately driving firing of output neurons which sum evidence over time. There are significant advantages of this approach: pseudo-simultaneity of input and output can be achieved [41], time-varying inputs can be more efficiently processed [42], and more efficient computation on specialized hardware can be accomplished [43].

The spiking neuron model used for this work is the simple Integrate-and-Fire (IF) model. The evolution of the membrane voltage v_{mem} is given by

$$\frac{dv_{\text{mem}}(t)}{dt} = \sum_i \sum_{s \in S_i} w_i \, \delta(t - s) \quad , \tag{3.6}$$

where w_i is the weight of the ith incoming synapse, $\delta(\cdot)$ is the delta function, and $S_i = \{t_i^0, t_i^1, \ldots\}$ contains the spike-times of the ith presynaptic neuron. If the membrane voltage crosses the spiking threshold v_{thr}, a spike is generated and the membrane voltage is reset to a reset potential v_{res}. In our simulations, this continuous-time description of the IF model is discretized into 1 ms timesteps.

[41] Luis Camunas-Mesa, Carlos Zamarreno-Ramos, Alejandro Linares-Barranco, Antonio J Acosta-Jimenez, Teresa Serrano-Gotarredona, and Bernabé Linares-Barranco. "An event-driven multi-kernel convolution processor module for event-driven vision sensors". In: *IEEE Journal of Solid-State Circuits* 47.2 (2012), pp. 504–517

[42] Peter O'Connor, Daniel Neil, Shih-Chii Liu, Tobi Delbruck, and Michael Pfeiffer. "Real-time classification and sensor fusion with a spiking Deep Belief Network". In: *Frontiers in Neuroscience* 7 (2013)

[43] Paul Merolla, John Arthur, Filipp Akopyan, Nabil Imam, Rajit Manohar, and Dharmendra S Modha. "A digital neurosynaptic core using embedded crossbar memory with 45pJ per spike in 45nm". In: *2011 IEEE Custom Integrated Circuits Conference (CICC)*. 2011, pp. 1–4

3.4.2 *Spiking Network Conversion*

In previous work, a significant performance gap was reported between the state-of-the-art achieved by conventional ANNs and spiking implementations [44]. Here the framework is laid out to facilitate the conversion of deep ANNs to SNNs, and to reduce the performance loss during this conversion. The conversion method used here is an extension of the one suggested in [45], extended to include our novel normalization methods and the analysis of firing rates and thresholds.

Let us first begin with observations about the relationships of ANNs using ReLUs and spiking networks. Firstly, the ReLU can be considered a firing rate approximation of an IF neuron with no refractory period [46], whereby the output of the ReLU is proportional to the number of spikes produced by an IF neuron within a given time window. ReLUs are also advantageous during training as their piecewise constant derivative leads to weight updates of a particularly simple form. Secondly, for classification tasks, only the maximum activation of all units in the output layer is of importance, allowing the overall rate to be scaled by a constant factor. Finally, without a bias to provide an external reference value, the relative scale of the neuron weights to each other and to the threshold of the neuron are the only parameters that matter. This gives rise to the following recipe for converting deep ANNs to SNNs:

1. Use ReLUs for all units of the network.
2. Fix the bias to zero throughout training, and train with back-propagation.
3. Directly map the weights from the ReLU network to a network of IF units.

[44] Peter O'Connor, Daniel Neil, Shih-Chii Liu, Tobi Delbruck, and Michael Pfeiffer. "Real-time classification and sensor fusion with a spiking Deep Belief Network". In: *Frontiers in Neuroscience* 7 (2013); Yongqiang Cao, Yang Chen, and Deepak Khosla. "Spiking Deep Convolutional Neural Networks for Energy-Efficient Object Recognition". In: *International Journal of Computer Vision* (2014), pp. 1–13

[45] Yongqiang Cao, Yang Chen, and Deepak Khosla. "Spiking Deep Convolutional Neural Networks for Energy-Efficient Object Recognition". In: *International Journal of Computer Vision* (2014), pp. 1–13

[46] Yongqiang Cao, Yang Chen, and Deepak Khosla. "Spiking Deep Convolutional Neural Networks for Energy-Efficient Object Recognition". In: *International Journal of Computer Vision* (2014), pp. 1–13

4. Use weight normalization (see section 3.4.3) to obtain near-lossless accuracy and faster convergence.

These suggestions work for both the case of fully-connected networks and CNNs. Once the ReLUs in the artificial neural network after training have been replaced by IF neurons, a loss of performance for a fixed simulation duration can come from three factors:

1. The unit did not receive sufficient input to cross its threshold, meaning its rate is lower than it should be;
2. The unit received so much input that the ReLU model predicts more than one output spike per timestep. This can happen either because there are too many input spikes in one timestep or if some of the input weights are higher than the neuron threshold.
3. Due to the probabilistic nature of the spiking input, it can happen that a set of spikes over- or under-activate a specific feature set due to non-uniformity of the spike trains.

Reducing the simulation timestep can help to reduce the number of input spikes per timestep, and increasing the simulation duration will help to avoid insufficient activation. However, all factors can be addressed by finding the right balance of spiking thresholds, input weights and input firing rates. Specifically, high spiking thresholds (or low input weights) decrease the error due to the over-activation and non-ideal spike trains, while increasing errors due to the under-activation factor and vice-versa. Note that only the ratio of spiking threshold to input weights determines the amount of integrated evidence until a spike is fired but not their individual values. Instead of hand-tuning the parameters, a more rigorous approach is here presented towards adjusting the network weights (and thereby the ratio of spiking threshold to input weights), i.e. to calculate rescaling factors for the weights which reduce the errors due to the three causes described above.

3.4.3 Weight Normalization

A key contribution of this work is a novel weight normalization procedure that puts the network into a regime where the above problematic factors are avoided.

Two possible ways to normalize the network weights are presented here to ensure that activations are sufficiently small to prevent the ReLU from overestimating output activations. The safest, most conservative method is to consider all possible positive activations that could occur as input to a layer, and rescale all the weights by that maximum possible positive input. If the maximum positive input can only cause one spike, then the network will never need to produce more than one spike at once from the same neuron. By doing so, the resulting spiking networks become robust to arbitrarily high input rates and completely eliminate losses due to too many

Algorithm 3: Model-Based Normalization

```
Rescale by theoretically maximum possible input
for layer in layers:
    max_pos_input = 0
    # Find maximum input for this layer
    for neuron in layer.neurons:
        input_sum = 0
        for input_wt in neuron.input_wts:
            input_sum += max(0, input_wt)
        max_pos_input = max(max_pos_input, input_sum)
    # Rescale all weights
    for neuron in layer.neurons:
        for input_wt in neuron.input_wts:
            input_wt = input_wt / max_pos_input
```

inputs. Unfortunately, this means that evidence integration in order to generate a spike might require much more time. If a high classification performance is required and longer sampling times are acceptable, this is the preferred method of finding the right weight scaling. This approach, outlined in algorithm 3, is referred to as *model-based normalization* because it requires only knowledge of the network weights.

Alternatively, the training set can be used to estimate typical activations within the network, rather than assuming the worst-case scenario of maximum positive activation. In our experiments, it was observed that this scaling factor is much less conservative. It preserves nearly all the accuracy but requires dramatically less evidence integration time. For this approach, after training a ReLU network, the training set is propagated through the neural network and the ReLU activations are stored. Then, the weights are normalized according to the maximum possible activation within the training set, so that this case would emit only a single spike. Additionally, this normalization requires taking into account the maximum single input weight as well, since otherwise a single spike could carry so much weight that the receiving neuron would need to spike multiple times within one timestep. While this is not a strong guarantee that performance can be maintained on the test set, the training set should be representative of the test set and results show this approach to be highly effective. This normalization method is especially suitable if both short latencies and high accuracy are required since a practically good tradeoff between those conflicting goals is found. Pseudocode for this approach is shown in algorithm 4 and is referred to as *data-based normalization*, since the weights are scaled according to actual activations of the network in response to data.

3.5 Experimental Setup

3.5.1 Dataset

Due to its ubiquity in machine learning, the MNIST dataset of handwritten digits was chosen for this investigation. The training set consists of 60,000 individual handwritten digits collected from postal codes, each labeled 0-9 for the individual 28x28 pixel

Algorithm 4: Data-Based Normalization

```
Rescale by example input
previous_factor = 1
for layer in layers:
    max_wt = 0
    max_act = 0
    for neuron in layer.neurons:
        for input_wt in neuron.input_wts:
            max_wt = max(max_wt, input_wt)
        max_act = max(max_act, neuron.output_act)
    scale_factor = max(max_wt, max_act)
    applied_factor = scale_factor / previous_factor
    # Rescale all weights
    for neuron in layer.neurons:
        for input_wt in neuron.input_wts:
            input_wt = input_wt / applied_factor
    previous_factor = scale_factor
```

grayscale images. The test set consists of 10,000 digits. The highest reported accuracy on this task using a single network and without extending the data set is 99.55% and was achieved using maxout networks [47]; the highest reported accuracy of a spiking implementation prior to this work is 98.30% [48] which was achieved using spiking CNNs.

3.5.2 *Architectures*

The code used to train and convert the networks in this paper is a modification of the Matlab DeepLearnToolbox [49] and can be found online.[50] Two main architectures were used in this work. First, to prove the efficacy of these networks for a straightforward typical neural network, a four layer fully-connected neural network was trained. Described in terms of the number of neurons in the network, this 784-1200-1200-10 network has two hidden layers of size 1200 units, with all neurons fully connected between the layers. Five networks were trained using a fixed learning rate of 1, momentum of 0.5, a batchsize of 100, 50 epochs of training, 50% dropout, and weights randomly initialized uniformly between -0.1 and 0.1. After training, the best-performing fully-connected neural network achieves a classification accuracy of 99.87% on the MNIST training set and 98.68% on the MNIST test set.

The second architecture is a 28x28-12c5-2s-64c5-2s-100 CNN. The input image is 28x28, followed by 12 convolutional kernels of size 5x5, followed by a 2x2 averaging subsampling window. This convolution process is repeated in a second stage with 64 maps of size 5x5, followed by a 2x2 averaging of the network. These final features are vectorized and fully connected to a 10-node output layer, where each of the 10 nodes represents one of the ten digit classes. The training process used a fixed learning rate of 1, a batchsize of 50, no momentum, 50% dropout of the kernels, zero bias, and 50 epochs of training. No distortions beyong the original data set are used. The resulting CNN achieves 99.19% training accuracy and 99.14% test accuracy.

The best-performing ReLU network from each of the training methods described above was then selected and the weights were

[47] Ian J. Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. "Maxout Networks". In: *ICML*. 2013

[48] D. Garbin, O. Bichler, E. Vianello, Q. Rafhay, C. Gamrat, L. Perniola, G. Ghibaudo, and B. DeSalvo. "Variability-tolerant Convolutional Neural Network for Pattern Recognition Applications based on OxRAM Synapses". In: *IEEE International Electron Devices Meeting (IEDM)* (2014), pp. 1–13

[49] R. B. Palm. "Prediction as a candidate for learning deep hierarchical models of data". MA thesis. 2012

[50] http://github.com/dannyneil/spiking_relu_conversion

transferred directly to a spiking IF network. A grid search of input rates (25, 50, 100, 200, 400, 1000 Hz) and thresholds (0.25, 0.5, 1, 2, 4, 10, 20) was then performed to determine the spiking networks with the best classification performance. This performance was compared to that of the data- and model-normalized networks of default threshold.

For the DenseNet, the model-based normalization scaled down the weights in each layer significantly: each layer's weights were multiplied according to algorithm 3, downscaling the layer weights by factors of 0.08 and 0.045. Model-based weight normalization was not applied to the output layer for either DenseNets or CNNs. Unlike the aggressive decrease of the weights due to model-based normalization, the data-based normalization (algorithm 4) scaled the weights by factors of 0.37, 1.25, and 0.8, only adjusting the network slightly but making it more robust to high input rates. In the CNN the scaling factors of the weights of the convolutional layers were 0.1657 and 0.1238 for the model-based normalization. Using the data-based normalization, the scaling factors 0.1657, 1.0021 and 1.19 were applied to the convolutional layers and the output layer, respectively. Note that the output layer weights are increased due to too little activation for the training set.

3.5.3 *Spiking Input*

The intensity values of the MNIST images were normalized to values between 0 and 1. Based on those intensity values, Poisson distributed spike trains were generated for each image pixel with firing rates proportional to the pixel's intensity value. For further details, see [51].

[51] Peter O'Connor, Daniel Neil, Shih-Chii Liu, Tobi Delbruck, and Michael Pfeiffer. "Real-time classification and sensor fusion with a spiking Deep Belief Network". In: *Frontiers in Neuroscience* 7 (2013)

3.6 *Results*

The overall effectiveness of converting custom ReLU networks to SNNs is summarized in Table 3.1. The first network in each section is the original trained ReLU network; its performance is the target performance of the spiking networks. Next, is the best-performing spiking IF network after a grid search of parameters, followed by the data-normalized network with the default threshold; and the model-normalized network. Note that the data-normalized network shows nearly the same performance as the original ReLU network without choosing hyperparameters or sweeping parameters for ideal performance. To obtain these results, the spike-based networks were simulated for 0.5 s for each input image. However, in practice, comparable performance can be achieved already after tens of milliseconds of simulated time, as discussed in section 3.6.3.

3.6.1 *Conversion and Parameter Choices*

The activations presented in Fig. 3.1 describe example responses from the different layers for different parameters of the DenseNet.

Network Type	Input Rate	Thr.	Accuracy
CNNs			
ReLU Rate-Based	–	–	99.14%
IF Network	1000 Hz	20.0	99.12%
Data-Norm. Net	400 Hz	1.0	99.10%
Model-Norm. Net	1000 Hz	1.0	99.11%
FC Networks			
ReLU Rate-Based	–	–	98.68%
IF Network	200 Hz	4.0	98.48%
Data-Norm. Net	1000 Hz	1.0	98.64%
Model-Norm. Net	1000 Hz	1.0	98.61%

Table 3.1: Comparison of classification accuracy for different network architectures and conversion mechanisms.

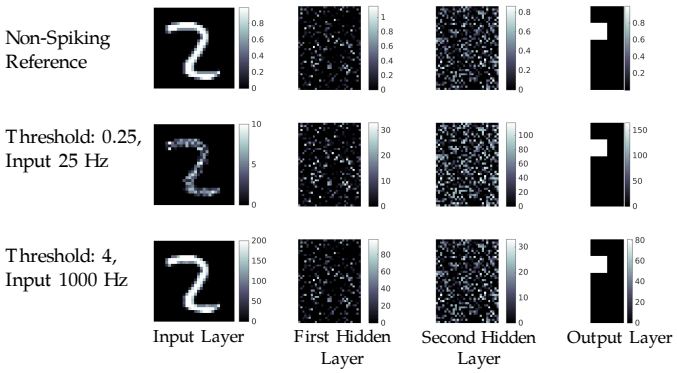


Figure 3.1: Comparison of activations between ReLU-based fully-connected network and non-normalized spiking network variants with different thresholds and input rates. The figure shows the accumulated spike count over 200 ms of simulation time. Ideally, the images in the bottom two rows should resemble a scaled version of the top row. Images shown here are scaled individually due to the unbounded upper range of the ReLU.

The top row shows responses from the layers of the ReLU-based network in response to the input. Note that each image in this plot is individually scaled, but the relative activations of the neurons within the map should ideally match across all rows. While the overall activation structure is well preserved, small differences occur mostly when neurons fire at lower rates.

3.6.2 *Accuracy*

Figure 3.2 shows the classification error and the number of spikes in the network (without input spikes) of the spiking CNN (upper plots) and the spiking fully-connected Network (lower plots) for a range of input firing rates and firing thresholds of the IF neurons. All performances are averaged over five simulations and all spike numbers are averaged over two simulations, using the same network but different input Poisson spike trains for each run. The highest performance of the spiking CNN was 99.12% for a first layer IF threshold of 20 and an input rate of 1000 Hz (upper left plot). The best performance of the spiking fully-connected network was 98.48% which was achieved using a threshold of 4 and a 200 Hz input rate, see lower left 2D plot. Generally, there is a trade-off between increasing the threshold to integrate more spikes before propagating the detection of a feature, and decreasing the threshold to reduce the sampling time necessary to produce a sufficient num-

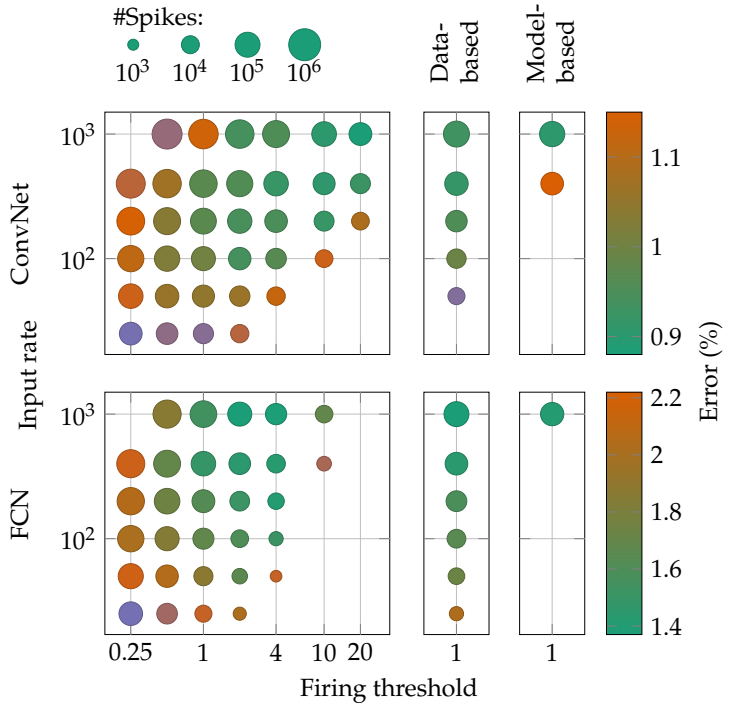


Figure 3.2: Classification performance and number of spikes produced for different architectures as a function of the input rate and the firing threshold. Upper panels show results for CNNs, lower panels for DenseNet. The color of each circle represents the mean accuracy on the MNIST test set (averaged over 5 trials), using an integration time of 0.5s (500 timesteps) for every input example. The size of the circle corresponds to the average number of spikes generated by the whole network per example presentation. The panels on the right show the same data for the normalized networks, whereby the threshold was fixed at 1 for all experiments. Parameter sets that led to test errors greater than 1.15 (ConvNet) or 2.2 (DenseNet), respectively, are not displayed.

ber of spikes to minimize the error due to the discretization of the transmitted messages. For low thresholds, higher performances are achieved using intermediate input rates, whereas for high thresholds, high input rates give better results. The number of spikes generated within the network (including spikes generated at the output layer) increases for higher input spiking rates and decreases for higher spiking thresholds. Surprisingly, the number of spikes generated within the fully-connected network and the CNN are comparable, although the fully-connected network uses about 60% more synapses than the CNN.

The results for the data-normalized and the model-normalized forms of both the spiking fully-connected network and the spiking CNN are shown on the four rightmost panels of Fig. 3.2. For both network types, the data-normalized networks show very high accuracies over a broad range of input firing rates. In contrast, model-normalized networks show a good performance only for high input rates for the spiking CNN. The reason for this can be seen in Fig. 3.3, which shows the classification error as a function of evidence integration time for the data-normalized networks, the model-normalized networks and the spiking networks with the best parameter set of the 2D plot. Due to the increased thresholds in each layer of the model-normalized network, the sampling time has to be increased to converge to a solution. On the other hand, both data-normalized networks converge faster than the best corresponding networks found in the grid search. This shows that data-based weight normalization is an effective method to obtain fast and accurate spiking deep networks.

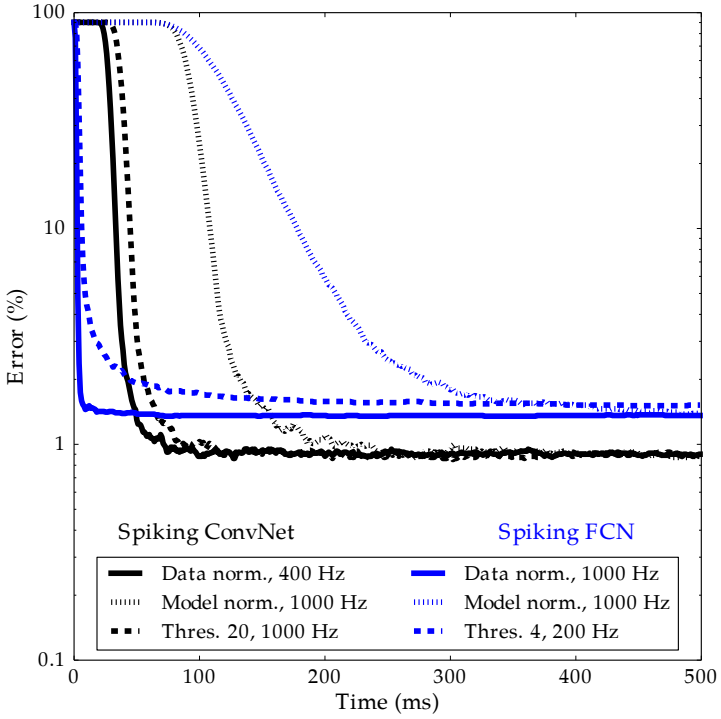


Figure 3.3: Classification error over time. Black curves show results for CNNs and blue curves show the results for fully-connected networks. Solid lines denote the error of data-normalized networks, dashed lines denote the error of model-normalized networks. Dotted lines denote the error for the best parameter set found from the 2D grid (figure 3.2). All networks except the model-normalized ones show very low error before 100 ms. Fully-connected data-normalized networks are close to their peak accuracy after only 6 ms (1.74% error).

3.6.3 *Convergence Time*

One of the reasons to use SNNs is their configurability. If high accuracy is desired, high spiking thresholds help to improve the accuracy; if short latencies are important, low firing thresholds ensure responses after only a few input spikes. The upper plot in Fig. 3.4 shows the number of unclassified examples over time, i.e. the time until the first output spike is produced, and the lower plot shows the classification error of the CNN using only the first output spike. A first layer threshold of 0.25 leads to very short output latencies (on the order of a few milliseconds) whereas a first layer threshold of 20 leads to greater latencies but more than 98% precision after the first spike. Note, however, that the limited precision in the case of a threshold of 0.25 is not ultimately problematic as extended execution time will yield more spikes with correct outputs.

3.7 *Conclusion*

This chapter presents a methodology for converting traditional neural networks to SNNs while maintaining high accuracy, and, with the introduced method of normalization, reduced evidence integration time to obtain the same accuracy. While previous investigations have examined the conversion of traditional neural networks to SNNs [52,53,54], here the focus lies on improving the converted network by adjusting the parameters of the spiking neurons. In particular, this chapter investigated typical sources of performance loss in SNNs and presented recipes for how to best address them.

[52] Peter O'Connor, Daniel Neil, Shih-Chii Liu, Tobi Delbruck, and Michael Pfeiffer. "Real-time classification and sensor fusion with a spiking Deep Belief Network". In: *Frontiers in Neuroscience* 7 (2013)

[53] Jose Pérez-Carrasco and others. "Mapping from Frame-Driven to Frame-Free Event-Driven Vision Systems by Low-Rate Rate Coding and Coincidence Processing–Application to Feedforward ConvNets". In: *IEEE Trans on Pattern Analysis and Machine Intelligence* 35.11 (2013), pp. 2706–2719

[54] Yongqiang Cao, Yang Chen, and Deepak Khosla. "Spiking Deep Convolutional Neural Networks for Energy-Efficient Object Recognition". In: *International Journal of Computer Vision* (2014), pp. 1–13

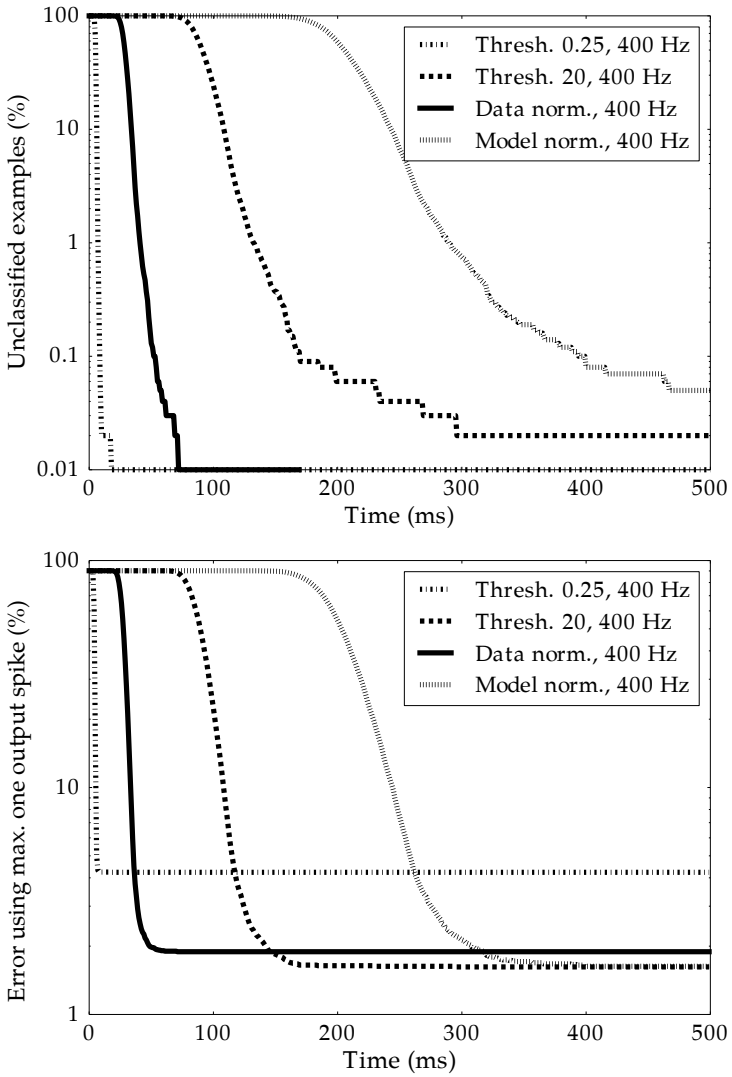


Figure 3.4: Time to first output spike and performance based on the first output spike. All 10,000 MNIST test examples were presented to the spiking CNN for 0.5s. The upper graph shows the percentage of examples for which none of the output neurons has fired as a function of time. The lower graph shows the error rate of the network, using only the first output spike to determine the class label.

Although the proposed *model-based* weight normalization led to a considerable slowdown, both the CNN and the DenseNet were still able to achieve high performance. The *data-based* normalization on the other hand improved latencies while assuring almost no loss due to conversion. This speedup is partly due to neurons in the deeper hidden layers being less activated. Therefore the data-based normalization actually *increases* the weights of those layers, leading to reduced latencies compared to just normalizing the first layer. This property of the data-based normalization will become especially important for converting state-of-the-art networks for challenging real-world tasks, where it is common to use more than ten hidden layers [55].

This chapter presents a significant step forward for spiking neural networks, finally closing the accuracy gap between state-of-the-art machine learning convolutional neural networks and spiking networks. Importantly, it maintains the key advantages of the event-driven sensors outlined in Chapter 1:

- Sparseness: the model successfully operates on sparse data, only

[55] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Proc. of NIPS*. 2012, pp. 1097–1105

processing nonzero inputs.

- Latency-accuracy and computation-accuracy tradeoffs: the conversion process maintains the ability to integrate for a longer timer period, incurring more computational cost and a greater consumption of time, in order to achieve better results, or to provide a lower-accuracy answer more quickly.
- Pseudo-simultaneity and low latency: the final spiking model exhibits pseudo-simultaneity, emitting output spikes during the continued presentation of the input in an ongoing, continuous manner.

With a novel, high-quality method to convert state-of-the-art machine learning networks into spiking networks, it is now possible to study optimizations that are unique to spiking networks and unavailable to traditional machine learning.

4
Unique Optimizations for Event-based Deep Networks

4.1 What new opportunities can be afforded?

The development of algorithms in Chapter 3 enabled spiking networks to achieve nearly state-of-the-art of machine learning accuracy on datasets while maintaining the advantages of event-driven inputs. Now, these spiking networks can be explored for the kinds of advantages that exist only in spiking neural networks, the advantages of event-driven network outlined in Chapter 1. Importantly, computation and latency are significantly more fixed in a standard deep neural network architecture than in an event-driven one. Computation is determined by the architecture of the network alone in a standard deep neural network, not as a variable cost that depends on input data, a confidence threshold, and the amount of time the network has run as it is in spiking networks. Latency is even more difficult to decrease in a standard deep neural network; the network produces no result at all until it successfully produces a complete result, while event-driven implementations softly integrate between no result and and asymptotically-reached final result. Event-driven deep networks have unique advantages. Perhaps learning algorithms can explore these unique advantages for novel optimizations?

The text appearing in Sections 4.1-4.4 is derived from previous text [1]. Section 4.2 covers the background and the novel algorithms that explore these advantages.. Section 4.3 covers the performance of these algorithms on a standard task, and Section 4.4 concludes this work and discusses the significance of these findings. Section 4.5 examines the advances so far and motivates the question the concern the remainder of this thesis.

[1] Daniel Neil, Michael Pfeiffer, and Shih-Chii Liu. "Learning to be Efficient: Algorithms for Training Low-Latency, Low-Compute Deep Spiking Neural Networks". In: *ACM Symposium on Applied Computing*. Vol. 31. 2016

While frame-based ANNs maintain a constant amount of computing steps to calculate an output, no matter what the input, it is a unique property of SNNs to provide several ways to control the amount of computation during classification. For example, the firing rate of input neurons can be varied, and weights and thresholds within the SNN can be adapted to produce higher or lower spike rates. However, as was shown in the previous chapter, the classification accuracy depends on these parameters, and high rates and

long integration times might be required to match the performance of a traditional ANN with a SNN.

Traditional machine learning focuses more or less exclusively on the question of improving the accuracy of a classifier, because run times of a fixed architecture can be assumed to be constant. However, there are many applications where fast and efficient inference is equally important, which motivates an approach to train SNNs in such a way that good classification accuracy, defined by a fixed performance level, is reached as fast and with as little computation as possible. Here, a variety of different approaches are evaluated that take *execution latency* and *computational cost* into account during training, allowing the network to best balance accuracy against computational effort. Networks can be encouraged to compute more efficiently either by punishing high firing rates and redundant representations, or by training feature detectors that react early after having seen only a small number of spikes from their respective inputs. Here all tested approaches yield Deep SNNs that perform at the desired level of accuracy of 98% on the MNIST benchmark, while dramatically reducing the latency and the number of compute steps compared to an ANN of identical size. This suggests that Deep SNNs are ideally suited to solve difficult classification tasks in real-time and at limited power budgets.

4.2 Methodology

4.2.1 Network Architecture and Dataset

For all experiments in this work, the network architecture was a 784-1200-1200-10 fully-connected feed-forward neural network, which was suggested in [2]. Networks were initially trained as ANNs composed of ReLUs, using a learning rate α of 0.01, momentum of 0.1, and weights initialized uniformly between $[-0.1, 0.1]$; this is referred to as "default" training in the remainder of this work. Unless otherwise specified, dropout was set to 0.5. A total of 522 networks were trained on the MNIST benchmark for handwritten digit classification [3], using a training set of 60,000 and a test set of 10,000 28x28 gray-level images.

[2] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. "Improving neural networks by preventing co-adaptation of feature detectors". In: *arXiv preprint arXiv:1207.0580* (2012)

[3] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324

4.2.2 SNN Conversion and Normalization

The goal of this work is to achieve efficient classification with SNNs, while maintaining parity of accuracy with traditional ANNs. Following the approach introduced in [4] and in the prior chapter, a standard (here, also referred to as rate-based) ANN is trained first as described above, and then converted into a SNN, where spike rates approximate activations within the ANN. This is done by using the weights of the ReLU network directly as the connection weights of an equivalent SNN composed of (non-leaky) IF neurons. In all experiments, the neurons are trained without bias, in order to save computation. The approaches for increasing the efficiency of

[4] Peter U Diehl, Daniel Neil, Jonathan Binas, Matthew Cook, Shih-Chii Liu, and Michael Pfeiffer. "Fast-Classifying, High-Accuracy Spiking Deep Networks Through Weight and Threshold Balancing". In: *International Joint Conference on Neural Networks (IJCNN)*. 2015

the network described below are either applied directly during the training, or, in some cases when the efficiency algorithm showed a tendency to decrease the accuracy in the process of decreasing computation, the network was fine-tuned after normal training.

Weight normalization for SNNs, previously introduced for convolutional and fully connected networks in [5], produces spiking networks that achieve equal classification performance to their rate-based equivalents. Here, these spiking networks are investigated for reducing the amount of computation. The normalization process starts from a previously trained ANN, which is converted into an SNN as described above. Finally, the weights of each layer are rescaled by a constant factor, determined by the data-normalization method described previously [6]. This constant factor is estimated from the training set so that a maximally-activated neuron will spike exactly once per timestep. Before this constant factor is applied, the original networks could sometimes have neurons momentarily activated beyond their spiking threshold with sufficiently high weights. This leads to inaccuracies, as each neuron is only able to communicate a single spike per timestep, regardless of how much the activation exceeds the threshold; this results in the loss of the extra activation due to the discretization of a single spike. Alternatively, in other networks one might find that the maximum activation in response to standard input is far less than the spiking threshold, and the neuron thus requires an unnecessarily high number of spikes to begin producing events that can be picked up by downstream neurons. A more detailed description of this data-based normalization can be found in [7].

For all networks and optimization techniques presented here, data-normalization was separately examined to study its contribution towards increasing computational efficiency.

[5] Peter U Diehl, Daniel Neil, Jonathan Binas, Matthew Cook, Shih-Chii Liu, and Michael Pfeiffer. "Fast-Classifying, High-Accuracy Spiking Deep Networks Through Weight and Threshold Balancing". In: *International Joint Conference on Neural Networks (IJCNN)*. 2015

[6] Peter U Diehl, Daniel Neil, Jonathan Binas, Matthew Cook, Shih-Chii Liu, and Michael Pfeiffer. "Fast-Classifying, High-Accuracy Spiking Deep Networks Through Weight and Threshold Balancing". In: *International Joint Conference on Neural Networks (IJCNN)*. 2015

[7] Peter U Diehl, Daniel Neil, Jonathan Binas, Matthew Cook, Shih-Chii Liu, and Michael Pfeiffer. "Fast-Classifying, High-Accuracy Spiking Deep Networks Through Weight and Threshold Balancing". In: *International Joint Conference on Neural Networks (IJCNN)*. 2015

4.2.3 *Evaluation Criteria*

An SNN classifier was considered successful when the classification accuracy on the test set reached 98%, a level which can nowadays be reached by most deep ANNs, within the number of operations required by a frame-based ANN. Since performance typically improves as evidence accumulates, the minimum number of input spikes per digit (which are converted into spike trains as in [8]) such that the overall accuracy on the test set reaches 98% are computed. This corresponds to input latency. Furthermore, the amount of computation can be computed as the total number of operations per digit to reach this performance level. In spiking networks an "operation" is defined as the addition of a synaptic weight to its neuron's membrane potential.

[8] Peter U Diehl, Daniel Neil, Jonathan Binas, Matthew Cook, Shih-Chii Liu, and Michael Pfeiffer. "Fast-Classifying, High-Accuracy Spiking Deep Networks Through Weight and Threshold Balancing". In: *International Joint Conference on Neural Networks (IJCNN)*. 2015

4.2.4 *Methods for Reducing Firing Rates*

The first class of algorithms to train efficient networks are those that aim to decrease the number of spikes within a network. Re-

ducing the number of spikes needed for the network decreases computation because each spike triggers further computation in the downstream neurons, in particular in fully-connected networks. Networks that generate fewer spikes will therefore have large savings in the overall computation.

Sparse Coding Sparse coding aims to represent the data using a small subset of the available basis functions at a given time. Thus, lower-compute spiking networks can make use of sparsity to achieve an encoding of inputs with fewer active neurons and therefore lower overall firing rates. Sparsity can be enforced by adding a regularization term L_{sparse} to the overall cost function, which penalizes deviations from a target firing rate s_{targ} in order to encourage learning of a sparse weight matrix [9]. The implementation here derives from [10], in which the penalty term is computed from the vector $\mathbf{y}$ of neuron activations, and the deviations from s_{targ} are calculated per component with cost factor s_{cost}:

$$L_{sparse} = s_{cost} \cdot \|\mathbf{y} - s_{targ} \cdot \mathbf{1}\| \tag{4.1}$$

[9] Christopher Poultney, Sumit Chopra, Yann L Cun, et al. "Efficient learning of sparse representations with an energy-based model". In: *Advances in neural information processing systems*. 2006, pp. 1137–1144

[10] R. B. Palm. "Prediction as a candidate for learning deep hierarchical models of data". MA thesis. 2012

Three networks for each combination of sparsity constraints were trained, using sparsity costs $s_{cost} \in \{0.1, 0.01, 0.001, 0.0001\}$ and target rates $s_{targ} \in \{0.2, 0.05, 0.01\}$. The networks were initially trained without sparsity for 20 epochs, after which training enforcing sparsity was continued for e_2 secondary training epochs, where $e_2 \in \{5, 50, 50\}$.

L2 Cost on Activation Another method of decreasing the number of spikes is to add a cost function that directly takes into account the predicted number of spikes. When using the conversion technique, the activation of a ReLU neuron in the ANN directly represents the expected firing rate of the neuron in the SNN. A cost function which penalizes high activations therefore decreases the expected number of spikes in the network. In this work, a modified L2-norm of the following form is used:

$$L_{act}(\mathbf{y}) = c_{act} \cdot \sum_i I(y_i > c_{min}) \cdot y_i^2 \tag{4.2}$$

Three networks at each combination of activation penalty $c_{act} \in \{0.1, 0.01, 0.001, 0.0001\}$ and $c_{min} \in \{0.5, 1.0, 1.5, 2.0\}$ were trained for epochs $e \in \{5, 20, 50\}$ after 2 epochs of pre-training to initialize weights to an approximately correct regime.

4.2.5 *Methods for Rapid Classification*

The second category of algorithms are those that produce accurate classifications more quickly. Since the Integrate-and-Fire neurons of the SNN approximate the continuous activation of a ReLU neuron by their firing rate, the goal of the following approaches is to make neurons, especially in higher layers, reach their steady-state firing

rates more quickly. A shorter runtime implies fewer spikes, and thus less computation.

Dropout Dropout [11] has been used very successfully as a regularization technique for large ANNs. In brief, the dropout algorithm sets each neuron's activation to zero with probability p during the forward computation. At very high dropout rates, the network is forced to pattern-complete from a minimal amount of input. For example, if $p = 0.9$, each subsequent layer is forced to classify correctly with only 10% of normal inputs. This concept is transferred to SNNs, where at any given point in time many neurons will not be active. Dropout in SNNs thus means that the network is encouraged to classify after very few input spikes. Five networks are trained for dropout probabilities $p \in \{0.0, 0.4, 0.5, 0.6, 0.7, 0.8\}$.

[11] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: A simple way to prevent neural networks from overfitting". In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958

Dropout Learning Schedule Because extremely high dropout could make training more difficult and cause a loss of accuracy, here an alternative strategy is proposed: after training the network normally for 50 epochs, training is continued over e_2 epochs while the dropout rate p is gradually increased to p_{final}. The training schedule of an example input (digit "2") can be seen in Fig. 4.1, where the entire digit is presented throughout the first phase, then parts of the image are dropped with a gradually increasing rate in the second phase. While Fig. 4.1 shows the dropout of the input, all layers of the network similarly have the same dropout level. Three networks for each combination of $p_{final} \in \{0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$ and additional training epochs $e_2 \in \{20, 50, 80\}$ were trained.

Since this technique can be performed on an already-trained network, it can be effectively used in combination with existing networks.

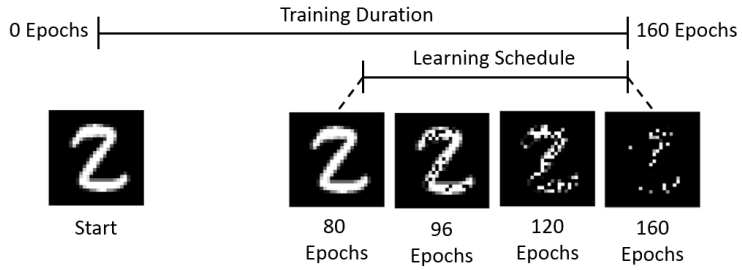


Figure 4.1: Diagram of an example dropout learning schedule with 160 epochs. The first 80 epochs use zero dropout, but the rate of dropout is gradually increased from 0% to 80% over the last 80 epochs.

Stacked Auto-Encoder with Zero Masking Similar in motivation to the variants of dropout introduced above, a Stacked Auto-Encoder (SAE) trained with high zero-masking, i.e. replacing input pixels with zero, will learn effective ways to restore a signal even given very little input. Since SAEs are trained with a cost function that measures how well each layer can restore its own input, a high

zero-masking rate of e.g. $p = 0.9$ requires the network to learn to reconstruct the input signal from only 10% of the input signal. While the standard approach for ANNs is to use an SAE to extract the signal from corrupt or noisy inputs, in the domain of SNNs this amounts to generating predictions of the external representation. From the first input spike, the SAE attempts to "undo" the zero-masking caused by the inputs that have not yet arrived. For each additional input spike, the challenge gets easier as the effective zero-masking of time is gradually undone and a more complete signal is provided. SAEs that have been trained with high zero-masking may restore the signal with very little information, which allows them to begin producing outputs very quickly. Training networks as SAEs is followed by a discriminative training with classification labels for e epochs. Three networks for every combination of zero-masking rate $p \in \{0.1, 0.3, 0.5, 0.8, 0.85, 0.95, 0.99\}$ and training epochs $e \in \{20, 50, 80\}$ were trained.

4.3 *Results*

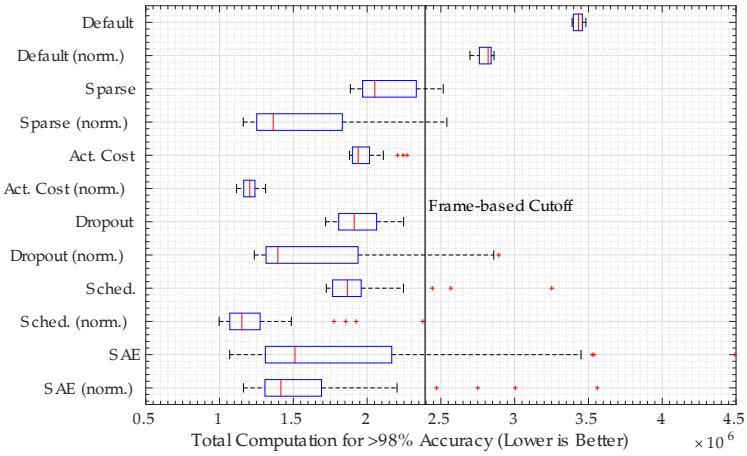


Figure 4.2: Boxplot indicating amount of computation for SNNs using different optimization approaches. This boxplot indicates minimum, first quartile, median (red line), third quartile, and maximum, with outliers shown as red stars. The majority of the optimization methods lie to the left of the black vertical line, indicating they require less computation than a frame-based ANN to achieve the same 98% classification accuracy. The best result shown here achieves the target accuracy in less than 42% of the computational operations required for an ANN.

As described in Sec. 4.2.3, the different approaches are evaluated by quantifying the number of input spikes and the amount of computation necessary to reach 98% accuracy on MNIST. Fig. 4.2 summarizes the computational cost of running a network trained according to these approaches. Ideally, networks should be located to the left, which would imply that 98% classification accuracy can be achieved with only a few operations within the network. The black line in Fig. 4.2 shows the number of operations necessary to arrive at an answer for the frame-based ANN, and nearly all of the examined SNNs require fewer operations. Furthermore, in comparison to the baseline results (labeled "Default,"), nearly all optimization algorithms offer a substantial improvement. Networks that do not achieve 98% accuracy are not shown; certain parameter configurations such as extremely high dropout or unbalanced

cost on the activation prevented these networks from achieving the target accuracy.

It can also be observed that weight normalization decreases computation for most networks. In Fig. 4.2, normalized networks from the same optimization algorithm had a tendency to shift the results to the left, implying a decrease of the total amount of computation in the networks. In general, the normalized networks were both faster (lower latency) and more efficient (fewer operations) than their unnormalized counterparts (see Table 4.1), with the exception of the SAEs. These networks had weights so large that normalization had a tendency to decrease the weights, thus requiring further latency to achieve the same activation.

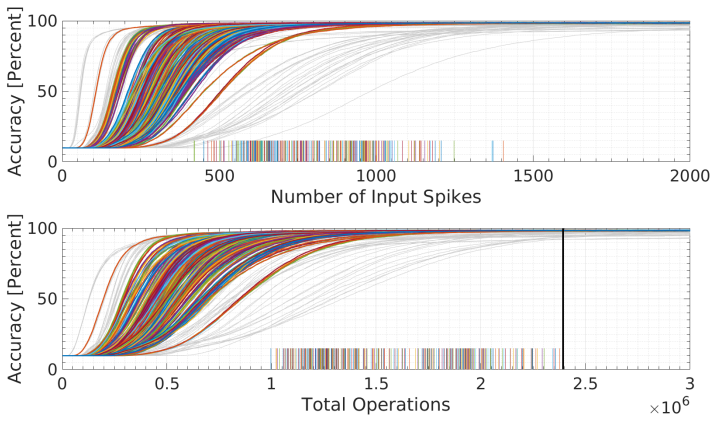


Figure 4.3: Dependency of accuracy on the number of input spikes and total operations for trained SNNs using different optimization approaches. The top figure depicts the accuracy versus latency while the bottom shows accuracy versus computation. Each line shows the accuracy curve for one of the 522 networks. Curves for networks that achieve 98% accuracy within the compute constraint are plotted in different (but arbitrary) colors, and the remaining networks are plotted in light gray. In both plots, a colored vertical tick mark on the horizontal axis is drawn to indicate the point at which a network passes 98% accuracy. In the bottom figure, the black vertical line indicates the amount of computation required for a frame-based ANN.

Figure 4.3 shows the accuracy of all 522 tested networks over time, i.e. either as a function of the number of input spikes (top), or the number of total operations (bottom). Since evidence is accumulating over time, these curves are typically monotonically increasing. The thin light gray curves indicate networks that did not achieve 98% accuracy within the compute constraint, which is indicated as a black vertical line. For those networks that did achieve 98% accuracy, a colored vertical line on the horizontal axis indicates the point at which the network crossed 98%, and the corresponding accuracy curve is plotted in color. One can see that the most efficient networks achieved classification with 0.997 MOps, compared to the fixed computation costs of the frame-based ANN requiring 2.39 MOps. This amounts to a reduction of operations by more than 58%.

Out of all the different networks, the network with the shortest latency to reach 98% accuracy is the SAE, which reaches the desired performance after only 445 input spikes. As mentioned above, the unnormalized SAE networks outperformed the normalized SAE networks. This is because the weights were so large that they were decreased by the normalization process, rather than increased, requiring more spikes to achieve the same level of accuracy. The fastest networks, which are the leftmost curves in Fig. 4.3, corre-

spond to the networks trained with extremely high dropout rates of $p = 0.70$ and $p = 0.80$. However, they were not able to achieve greater than 98% accuracy and so are shown in grey. In fact, even rate-based ANNs before conversion were not able to achieve accuracies above 98.10% with such high dropout rates, so they were excluded from further analysis.

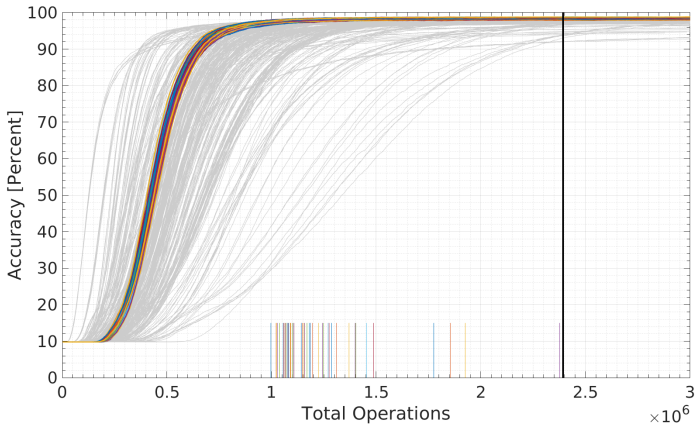


Figure 4.4: Same as Fig. 4.3, but highlighting only the results for the 54 SNNs trained with a Dropout Learning Schedule in color, with the remaining SNNs in light gray. One can see the remarkable similarity of learning results for different parameter settings.

In order to compare the influence of different parameter settings for one approach, a set of performance curves can be shown for a given algorithm, in this case the dropout learning schedule (Sec. 4.2.5) in Fig. 4.4. Highlighted in bright colors are the curves corresponding to all curves for all networks that use this approach and achieved greater than 98% accuracy in fewer computes than a frame-based ANN. These are 46 of the 54 parameter combinations tested, and the networks have been weight-normalized. Their performance is tightly aligned, despite the large variations in the tested parameters, indicating robustness to the parameters of the dropout learning schedule. Moreover, these networks achieved the overall lowest compute cost of 0.997 MOps, while simultaneously having among the lowest latency, producing accurate classification after only 602 input spikes.

A summary of the performance results for the different optimization methods, together with suggested parameter ranges for each method can be found in Table 4.1. It shows that in general all methods outperform the default case of an unprocessed SNN by a wide margin. The best methods reduce computation by 70% of Ops, and the latency by almost 75%. Among the methods tested, an advantage for networks trained with dropout learning schedule can be found in the normalized case, and SAE in the unnormalized case. With the exception of SAE, in general weight-normalization is advantageous for all methods.

In order to analyze qualitative differences in the features learned by different learning approaches, each row in Fig. 4.5 shows ten randomly-selected features (i.e. weight vectors) learned by the first hidden layer connecting to the input layer. As expected, the first

Method	Parameters	Unnormalized		Normalized	
		Ops	Latency	Ops	Latency
Default	—	3.43	1768	2.80	1781
Sparse Coding	$s_{cost} = 0.0001$ to achieve $s_{targ} = 0.01$	2.00	1031	1.22	631
Activation Cost	Cost of $c_{act} = 0.01$ above $c_{min} = 2.0$	1.92	952	1.17	602
Dropout	Dropout of 50%	1.98	1028	1.27	641
Dropout Learning Sched.	$p_{final} = 0.90$ after $e_2 = 50$ epochs	1.79	931	**1.04**	**602**
SAE	$p = 0.80$ for $e = 50$ epochs	**1.17**	**445**	1.25	788

Table 4.1: Summary of Results: Comparison of the number of operations (measured in millions of adds) and latency (measured as the number of input spikes) necessary to reach 98% accuracy for different optimization approaches. Networks with unnormalized and normalized weights are compared.

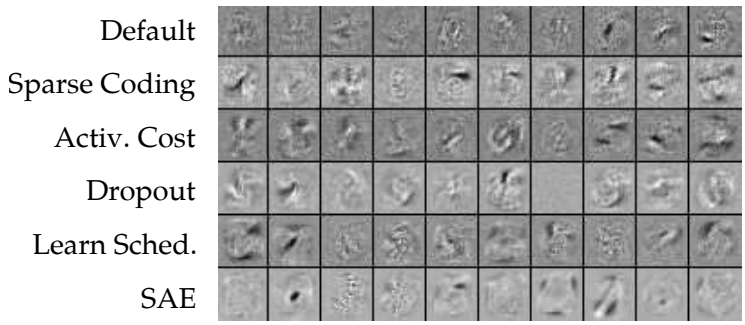


Figure 4.5: Examples of features learned in the first hidden layer with different optimization approaches. 10 features were selected randomly, and are displayed with normalized gray levels. Note that, similar to previous studies, Gabor-like and stroke-like features of the MNIST digits appear for all approaches.

layer learns Gabor-like and stroke-like filters in all cases similar to those that emerge in all state-of-the-art methods [12], thus showing no major differences between the weights learned by different approaches.

4.4 *Discussion*

[12] Daniel D Lee and H Sebastian Seung. "Learning the parts of objects by non-negative matrix factorization". In: *Nature* 401.6755 (1999), pp. 788–791

The purpose of this work is to demonstrate the power and efficiency of spiking neural networks. These networks were able to achieve equivalent classification performance as their rate-based equivalents, and often did so in far fewer operations and with a shorter latency. Due to the efficiency of a spike-based implementation, the input can be correctly classified before a frame-based approach could even read in all the necessary pixels to compute.

Here, different methods were examined for incorporating efficiency into the training process of neural networks, thereby allowing networks to *learn to be efficient*. Overall, nearly all methods and parameter sets yielded much improved results in terms of efficiency, while maintaining classification accuracy. Even more encouraging, several different training methods converge to approximately the same fundamental minima of computation as well as latency (around 1.5 MOps, in Fig. 4.2), which is a phenomena that warrants further study.

The fastest accurate classification was achieved using unnormalized SAEs, which reached 98% accuracy after only 420 spikes. A network without any optimization required on average 1753 spikes,

which results in a latency more than four times as long. The most efficient network in terms of operations, a normalized network trained with a dropout learning schedule, needs only 1.04 MOps compared to the standard default training of 3.43 MOps, amounting to almost 70% reduction.

While no algorithm clearly wins, and further research can demonstrate how these results extend to other tasks, the newly introduced dropout learning schedule algorithm is perhaps the best to use in practice. It is compatible with existing learned architectures, straightforward to implement, and yields extremely low-compute and low-latency networks. Moreover, when used with existing networks, the overall accuracy can be monitored during the learning schedule and training can be terminated early or late, depending on how much accuracy loss is acceptable.

IMPORTANTLY, the presented SNNs communicate all necessary values with a single binary event, and thus do not rely on a costly multiplier implementation in hardware. Thus, if hardware costs are considered, optimized SNNs might be even more advantageous, since they can be implemented using simple adders, and are thus very amenable to a simple, and low-power optimized hardware implementation.

4.5 What Questions Should Be Addressed Next?

At this moment, it is useful summarize the prior accomplishments of this thesis. Several hardware implementations of event-driven deep neural networks have been designed, yielding efficient event-driven network implementations (Chapter 2). Novel methods of transferring state-of-the-art deep learning results to event-driven networks have been developed, which allow maintaining high accuracy even when run on extremely efficient platforms (Chapter 3). Finally, in this chapter, new optimizations have been developed which exploit the unique abilities of an event-driven network, allowing both latency and computation itself to be optimized by stochastic gradient descent.

HOWEVER, the benchmark problems investigated here - image recognition tasks using MNIST, for example - are artificial tasks ported from machine learning in order to provide a common base of comparison, and the methods introduced to address these challenges with spiking neural networks have significant drawbacks. Namely, in order to achieve the highest accuracy levels, all the neurons in the deep neural network have infinite memory and no decay. This allows the input spikes, which are assumed to be drawn from a stationary underlying distribution, to be accumulated without loss of information to decay.

In turn, this requires the input firing rates to be drawn from a *stationary distribution*. While images can be easily modeled as a

stationary distribution, with pixel intensity proportional to a firing rate, most real-world phenomena are not stationary over time. The conversion methods simply do not support non-stationary input distributions. The fundamental tenet of the conversion, explored in Chapter 3, is that the accumulated number of spikes from a neuron forms a scaled firing rate, and that this firing rate in turn represents the analog value of a standard deep neural network neuron. The networks lack an understanding of time as they lack time constants, computing only on the time-average of the summed input, making it impossible for a neuron to determine the difference between different *firing patterns* that have the same average *firing rate*.

This is actually quite a large problem, as the event-based sensors are inherently dynamic. They do not produce a rate-based approximation of a scene, either visual or acoustic, but rather a measure of its changes. The SNNs introduced so far lack a way of learning to understand the meaning of these changes, as the inputs evolve over time.

So which problems should be addressed next? Clearly, while MNIST and CIFAR are excellent problems on which to prototype learning algorithms, they are considered relatively simple by the machine learning community. Should the next investigation focus on substantially more difficult datasets, extending the methods introduced in Chapter 3 to state-of-the-art architecture and the most challenging machine learning problems?

Instead, it is the belief of the author that a more fundamental question ought to be addressed first: how to deal with non-stationary, event-based inputs. Fundamentally, the goal of this thesis has not yet been accomplished by the novel algorithms introduced so far. While these algorithms permit computation on event-based inputs that preserve the advantages of the event-based sensors described in Chapter 1, the networks only operate on a very artificial form of an event-based "sensor": the software model that converts stationary images to Poisson spike trains. On real event-based inputs, as described above, these networks lack the ability to discern the spatiotemporal patterns that characterize event-based outputs.

An algorithm is needed which can natively learn to process the complex spatio-temporal surfaces generated by event-based sensors.

5
Developing a Model to Directly Learn from Event-based Data

5.1 Introduction

Previous chapters have introduced methods for converting static neural networks, which lack a representation of time, into spiking neural networks. As discussed in the conclusion of Chapter 4, while the spiking neural networks accumulate evidence over time, the algorithms for these spiking networks introduced so far operate on static, unchanging inputs. That is, these spiking networks sorely lack an ability to deal with dynamic inputs. Instead, it could make sense to pair dynamic, time-evolving sensors with a neural network architecture that has a native understanding of time.

The obvious choices for this model are RNN architectures, as these models equip neural networks with memories. In recent years, RNNs have achieved ever greater success in a variety of application domains, including natural language processing [1,2], speech recognition [3], and attention-based models for mulitmodal processing [4,5]. Their popularity has also grown as larger datasets, more powerful computing resources, and better training algorithms have made these models more effective and easier to use. Pioneering advances such as Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) [6,7], which equip these models with gates, have enabled these models to learn realistically complex and long sequences. However, in light of the goal to use an RNN with an event-driven sensor, modern RNNs are discrete, timestepped models which are rather incompatible with the continuous-time event-driven sensors. While some previous approaches realized the resulting limitations of pursuing only timestepped recurrent neural networks [8,9,10] and looked towards continuous-time dynamical system approaches for RNNs, most modern RNN implementations use discrete timesteps.

This chapter introduces one of the main contributions of this thesis, the Phased LSTM model, which equips modern, state-of-the-art recurrent neural network models with the ability to process continuous-time signals.

This has important implications in real-world tasks such as au-

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate". In: *arXiv preprint arXiv:1409.0473* (2014)

[2] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. "Recurrent neural network based language model." In: *Interspeech* 2 (2010), p. 3

[3] Alex Graves, Abdel-Rahman Mohamed, and Geoffrey Hinton. "Speech recognition with deep recurrent neural networks". In: *2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2013, pp. 6645–6649

[4] Kyunghyun Cho, Aaron Courville, and Yoshua Bengio. "Describing multimedia content using attention-based encoder-decoder networks". In: *IEEE Transactions on Multimedia* 17.11 (2015), pp. 1875–1886

[5] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard Zemel, and Yoshua Bengio. "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention". In: *International Conference on Machine Learning*. 2015

[6] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural Computation* 9.8 (1997), pp. 1735–1780

[7] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. "Learning phrase representations using RNN encoder-decoder for statistical machine translation". In: *arXiv preprint arXiv:1406.1078* (2014)

tonomous driving and robotics, as the need to integrate input from a variety of sensors with differing sampling rates is an important but largely unexplored problem within deep neural networks. The variety of sensors that enrich a platform - for example, auditory, visual, gyroscopic, distance-measuring - are each likely to have their own sampling rate. As a result, a standard discrete-time RNN would be forced to operate at the smallest time step merely to accommodate the sensors with a high sampling frequency (resulting in unnecessarily higher computation and power consumption), or instead to sacrifice response-time latency to run at a slower rate.

Instead, a model which can natively make use of the continuous time of event-based sensors can process inputs with very low latencies and accurate timing, preserving the advantages laid out in Chapter 1. Excitingly, Phased LSTM also is able to give back to the machine learning community as it is able to succeed in a variety of standard machine learning tasks where previous approaches have failed. This validates the hypothesis in Chapter 1 that the data manipulations typically performed (discrete timesteps, significant normalization, clean and regular data) do indeed constrain the models that are designed, and that forcing researchers to confront the challenges can yield advantageous models that were otherwise overlooked.

[10] Gert Cauwenberghs. "An analog VLSI recurrent neural network learning a continuous-time trajectory". In: *IEEE Transactions on Neural Networks* 7.2 (1996), pp. 346–361

The text in Section 5.2-5.4 originally appeared previously [11]. Section 5.2 introduces the formulation of the Phased LSTM model. Section 5.3 shows the results of this model when applied to a variety of tasks designed to stress its capabilities. Finally, Section 5.4 concludes this chapter with the summary of the model and its implications, and Section 5.5 discusses the model in the context of this thesis.

[11] Daniel Neil, Michael Pfeiffer, and Shih-Chii Liu. "Phased LSTM: Accelerating Recurrent Network Training for Long or Event-based Sequences". In: *Advances in Neural Information Processing Systems*. 2016, pp. 3882–3890

5.2 *Model Description*

Long short-term memory (LSTM) units [12] (Fig. 5.1) are an important ingredient for modern deep RNN architectures. First, the definition of the update equations in the commonly-used version from [13]:

[12] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural Computation* 9.8 (1997), pp. 1735–1780

[13] Alex Graves. "Generating sequences with recurrent neural networks". In: *arXiv preprint arXiv:1308.0850* (2013)

$$i_t = \sigma_i(x_t W_{xi} + h_{t-1} W_{hi} + w_{ci} \odot c_{t-1} + b_i) \tag{5.1}$$
$$f_t = \sigma_f(x_t W_{xf} + h_{t-1} W_{hf} + w_{cf} \odot c_{t-1} + b_f) \tag{5.2}$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \sigma_c(x_t W_{xc} + h_{t-1} W_{hc} + b_c) \tag{5.3}$$
$$o_t = \sigma_o(x_t W_{xo} + h_{t-1} W_{ho} + w_{co} \odot c_t + b_o) \tag{5.4}$$
$$h_t = o_t \odot \sigma_h(c_t) \tag{5.5}$$

The main difference to classical RNNs is the use of the gating functions i_t, f_t, o_t, which represent the *input*, *forget*, and *output* gate at time t respectively. c_t is the cell activation vector, whereas x_t and h_t represent the input feature vector and the hidden output vector respectively. The gates use the typical sigmoidal nonlinearities σ_i, σ_f, σ_o and tanh nonlinearities σ_c, and σ_h with weight parameters

W_{hi}, W_{hf}, W_{ho}, W_{xi}, W_{xf}, and W_{xo}, which connect the different inputs and gates with the memory cells and outputs, as well as biases b_i, b_f, and b_o. The cell state c_t itself is updated with a fraction of the previous cell state that is controlled by f_t, and a new input state created from the element-wise (Hadamard) product, denoted by $\odot$, of i_t and the output of the cell state nonlinearity σ_c. Optional *peephole* [14] connection weights w_{ci}, w_{cf}, w_{co} further influence the operation of the input, forget, and output gates.

[14] Felix A Gers and Jürgen Schmidhuber. "Recurrent nets that time and count". In: *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks (IJCNN)*. vol. 3. IEEE. 2000, pp. 189–194

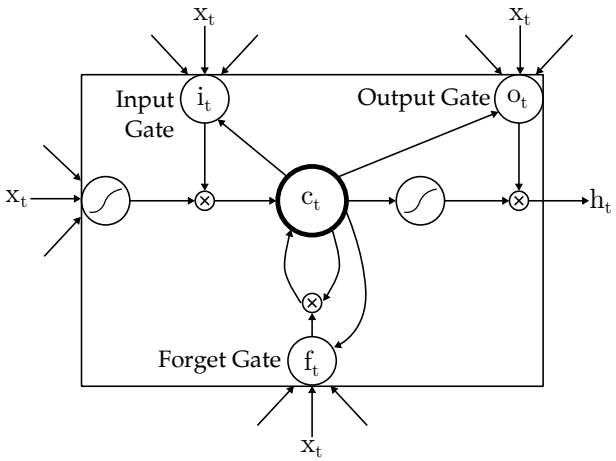


Figure 5.1: Model architecture of the standard LSTM model.

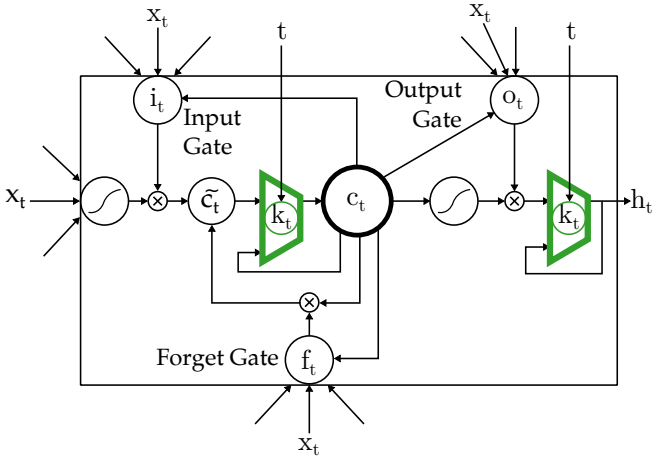


Figure 5.2: Model architecture, Phased LSTM model, with time gate k_t controlled by timestamp t. In the Phased LSTM formulation, the cell value c_t and the hidden output h_t can only be updated during an "open" phase; otherwise, the previous values are maintained.

The Phased LSTM model extends the LSTM model by adding a new *time gate*, k_t (Fig. 5.2(b)). The opening and closing of this gate is controlled by an independent rhythmic oscillation specified by three parameters; updates to the cell state c_t and h_t are permitted only when the gate is open. The first parameter, τ, controls the real-time period of the oscillation. The second, r_{on}, controls the ratio of the duration of the "open" phase to the full period. The third, s, controls the phase shift of the oscillation to each Phased LSTM cell.

All parameters can be learned during the training process. Though other variants are possible, here is proposed a particularly successful linearized formulation of the time gate, with analogy to

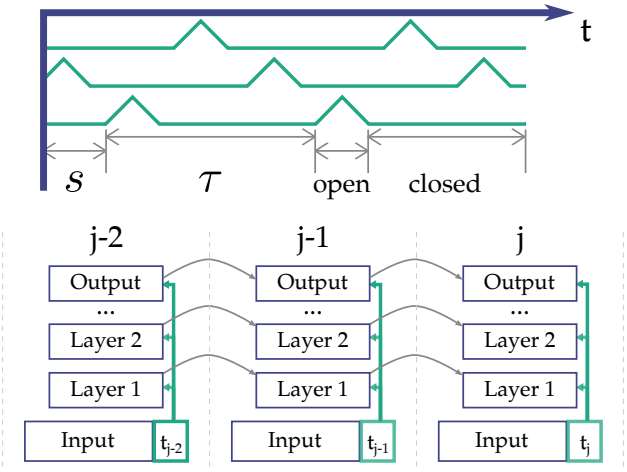


Figure 5.3: Diagram of Phased LSTM behaviour. The rhythmic oscillations to the time gates of 3 different neurons; the period τ and the phase shift s is shown for the lowest neuron. The parameter r_{on} is the ratio of the open period to the total period τ. Bottom: Note that in a multilayer scenario, the timestamp is distributed to all layers which are updated at the same time point.

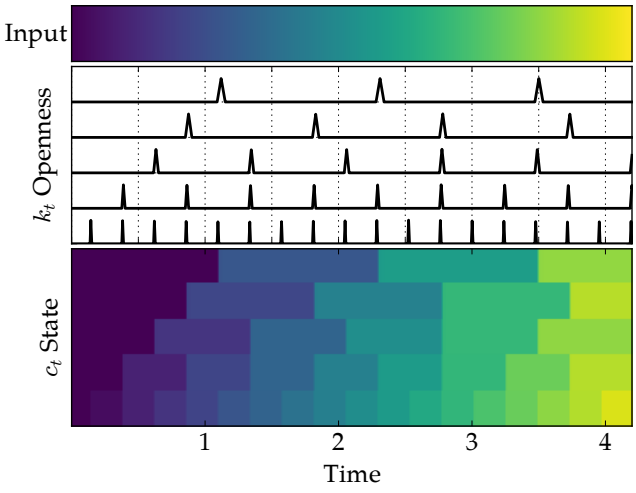


Figure 5.4: Illustration of Phased LSTM operation. A simple linearly increasing function is used as an input. The time gate k_t of each neuron has a different τ, identical phase shift s, and an open ratio r_{on} of 0.05. Note that the input (top panel) flows through the time gate k_t (middle panel) to be held as the new cell state c_t (bottom panel) only when k_t is open.

the rectified linear unit that propagates gradients well:

$$\phi_t = \frac{(t-s) \bmod \tau}{\tau}, \qquad k_t = \begin{cases} \frac{2\phi_t}{r_{on}}, & \text{if } \phi_t < \frac{1}{2} r_{on} \\ 2 - \frac{2\phi_t}{r_{on}}, & \text{if } \frac{1}{2} r_{on} < \phi_t < r_{on} \\ \alpha \phi_t, & \text{otherwise} \end{cases} \tag{5.6}$$

ϕ_t is an auxiliary variable, which represents the phase inside the rhythmic cycle. The gate k_t has three phases (see Fig. 5.3): in the first two phases, the "openness" of the gate rises from 0 to 1 (first phase) and drops from 1 to 0 (second phase). During the third phase, the gate is closed and the previous cell state is maintained. The leak with rate α is active in the closed phase, and plays a similar role as the leak in a parametric "leaky" rectified linear unit [15] by propagating important gradient information even when the gate is closed. Note that the linear slopes of k_t during the open phases of the time gate allow effective transmission of error gradients.

In contrast to traditional RNNs, and even sparser variants of RNNs [16], updates in Phased LSTM can optionally be performed at irregularly sampled time points t_j. This allows the RNNs to work

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". In: *The IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 1026–1034

[16] Jan Koutnik, Klaus Greff, Faustino Gomez, and Juergen Schmidhuber. "A clockwork RNN". in: *arXiv preprint arXiv:1402.3511* (2014)

with event-driven, asynchronously sampled input data. The shorthand notation $c_j = c_{t_j}$ is used for cell states at time t_j (analogously for other gates and units), and let c_{j-1} denote the state at the previous update time t_{j-1}. Then, the regular LSTM cell update equations for c_j and h_j (from Eq. 3 and Eq. 5) can be rewritten using proposed cell updates $\widetilde{c_j}$ and $\widetilde{h_j}$ mediated by the time gate k_j:

$$\widetilde{c_j} = f_j \odot c_{j-1} + i_j \odot \sigma_c(x_j W_{xc} + h_{j-1} W_{hc} + b_c) \tag{5.7}$$

$$c_j = k_j \odot \widetilde{c_j} + (1 - k_j) \odot c_{j-1} \tag{5.8}$$

$$\widetilde{h_j} = o_j \odot \sigma_h(\widetilde{c_j}) \tag{5.9}$$

$$h_j = k_j \odot \widetilde{h_j} + (1 - k_j) \odot h_{j-1} \tag{5.10}$$

A schematic of Phased LSTM with its parameters can be found in Fig. 5.3, accompanied by an illustration of the relationship between the time, the input, the time gate k_t, and the state c_t in Fig. 5.4.

One key advantage of this Phased LSTM formulation lies in the rate of memory decay. For the simple task of keeping an initial memory state c_0 as long as possible without receiving additional inputs (i.e. $i_j = 0$ at all time steps t_j), a standard LSTM with a nearly fully-opened forget gate (i.e. $f_j = 1 - \epsilon$) after n update steps would contain

$$c_n = f_n \odot c_{n-1} = (1 - \epsilon) \odot (f_{n-1} \odot c_{n-2}) = \ldots = (1 - \epsilon)^n \odot c_0 \tag{5.11}$$

This means the memory for $\epsilon < 1$ decays exponentially with every time step. Conversely, the Phased LSTM state only decays during the open periods of the time gate, but maintains a perfect memory during its closed phase, i.e. $c_j = c_{j-\Delta}$ if $k_t = 0$ for $t_{j-\Delta} \leq t \leq t_j$. Thus, during a single oscillation period of length τ, the units only update during a duration of $r_{on} \cdot \tau$, which will result in substantially fewer than n update steps. Because of this cyclic memory, Phased LSTM can have much longer and adjustable memory length via the parameter τ.

The oscillations impose sparse updates of the units, therefore substantially decreasing the total number of updates during network operation.

During training, this sparseness ensures that the gradient is required to backpropagate through fewer updating timesteps, allowing an undecayed gradient to be backpropagated through time and allowing faster learning convergence. Similar to the shielding of the cell state c_t (and its gradient) by the input gates and forget gates of the LSTM, the time gate prevents external inputs and time steps from dispersing and mixing the gradient of the cell state.

5.3 *Results*

In the following sections, the advantages of the Phased LSTM model are investigated in a variety of scenarios that require either precise timing of updates or learning from a long sequence.

For all the results presented here, the networks were trained with Adam [17] set to default learning rate parameters, using Theano [18] with Lasagne [19]. Unless otherwise specified, the leak rate was set to $\alpha = 0.001$ during training and $\alpha = 0$ during test. The phase shift, s, for each neuron was uniformly chosen from the interval $[0, \tau]$. The parameters τ and s were learned during training, while the open ratio r_{on} was fixed at 0.05 and not adjusted during training, except in the first task to demonstrate that the model can train successfully while learning all parameters.

[17] Diederik Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014)

[18] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. "Theano: a CPU and GPU math expression compiler". In: *Proceedings of the Python for scientific computing conference (SciPy)*. Vol. 4. 2010, p. 3

5.3.1 *Frequency Discrimination Task*

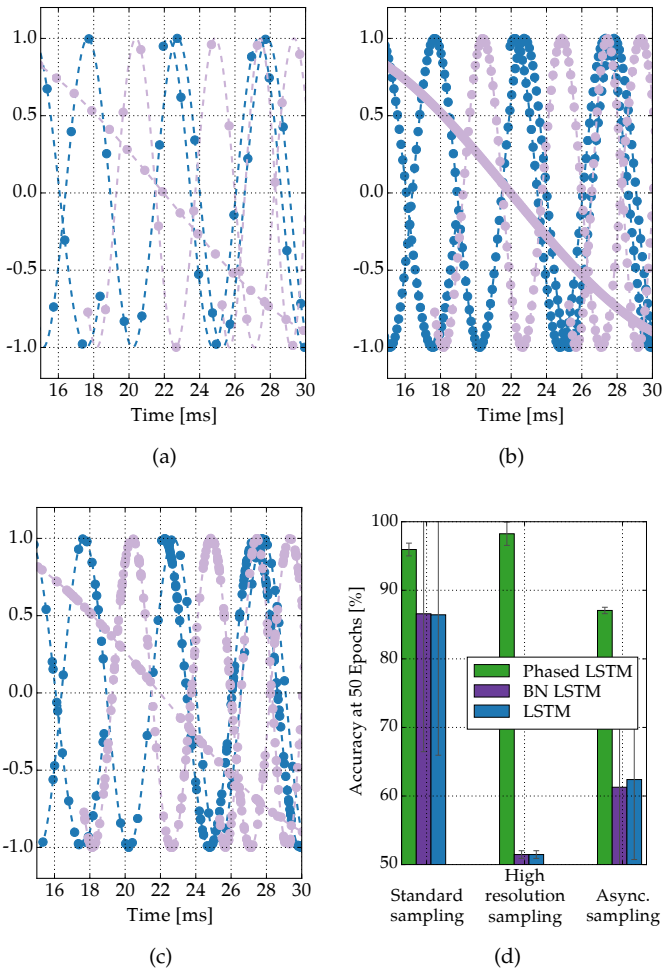


Figure 5.5: Frequency discrimination task. The network is trained to discriminate waves of different frequency sets (shown in blue and gray); every circle is an input point. **(a)** Standard condition: the data is regularly sampled every 1 ms. **(b)** High resolution sampling condition: new input points are gathered every 0.1ms. **(c)** Asynchronous sampling condition: new input points are presented at intervals of 0.02 ms to 10 ms. **(d)** The accuracy of Phased LSTM under the three sampling conditions is maintained, but the accuracy of the BN-LSTM and standard LSTM drops significantly in the sampling conditions (b) and (c). Error bars indicate standard deviation over 5 runs.

In this first experiment, the network is trained to distinguish two classes of sine waves from different frequency sets: those with a period in a target range $T \sim \mathcal{U}(5,6)$, and those outside the range, i.e. $T \sim \{\mathcal{U}(1,5) \cup \mathcal{U}(6,100)\}$, using $\mathcal{U}(a,b)$ for the uniform distribution on the interval (a,b). This task illustrates the advantages of Phased LSTM, since it involves a periodic stimulus and requires fine timing discrimination. The inputs are presented as pairs $\langle y,\ t \rangle$, where y is the amplitude and t the timestamp of the sample from the input sine wave.

[19] Sander Dieleman et al. *Lasagne: First release.* Aug. 2015. DOI: 10.5281/zenodo.27878. URL: http://dx.doi.org/10.5281/zenodo.27878

Figure 5.5 illustrates the task: the blue curves must be separated from the lighter curves based on the samples shown as circles. Three conditions for sampling the input signals are evaluated: In the standard condition (Fig. (a)), the sine waves are regularly sampled every 1 ms; in the oversampled condition (Fig. (b)), the sine waves are regularly sampled every 0.1 ms, resulting in ten times as many data points. Finally, in the asynchronously sampled condition (Fig. (c)), samples are collected at asynchronous times over the duration of the input. Additionally, the sine waves have a uniformly drawn random phase shift from all possible shifts, random numbers of samples drawn from $\mathcal{U}(15, 125)$, a random duration drawn from $\mathcal{U}(15, 125)$, and a start time drawn from $\mathcal{U}(0, 125 - \text{duration})$. The number of samples in the asynchronous and standard sampling condition is equal. The classes were approximately balanced, yielding a 50% chance success rate.

Single-layer RNNs are trained on this data, each repeated with five random initial seeds. The Phased LSTM configuration can be compared to regular LSTM, and Batch-normalized (BN) LSTM which has found success in certain applications [20]. For the regular LSTM and the BN-LSTM, the timestamp is used as an additional input feature dimension; for the Phased LSTM, the time input controls the time gates k_t. The architecture consists of 2-110-2 neurons for the LSTM and BN-LSTM, and 1-110-2 for the Phased LSTM. The oscillation periods of the Phased LSTMs are drawn uniformly in the exponential space to give a wide variety of applicable frequencies, i.e., $\tau \sim \exp(\mathcal{U}(0,\ 3))$. All other parameters match between models where applicable. The default LSTM parameters are given in the Lasagne Theano implementation, and were kept for LSTM, BN-LSTM, and Phased LSTM. Appropriate gate biasing was investigated but did not resolve the discrepancies between the models.

All three networks excel under standard sampling conditions as expected, as seen in Fig. (d) (left). However, for the same number of epochs, increasing the data sampling by a factor of ten has devastating effects for both LSTM and BN-LSTM, dropping their accuracy down to near chance (Fig. (d), middle). Presumably, if given enough training iterations, their accuracies would return to the normal baseline. However, for the oversampled condition, Phased LSTM actually *increases* in accuracy, as it receives more information about the underlying waveform. Finally, if the updates are not evenly spaced and are instead sampled at asynchronous times, even when controlled to have the same number of points as the standard sampling condition, it appears to make the problem rather challenging for traditional state-of-the-art models (Fig. (d), right). However, the Phased LSTM has no difficulty with the asynchronously sampled data, because the time gates k_t do not need regular updates and can be correctly sampled at any continuous time within the period.

The previous task is extended by training the same RNN architectures on signals composed of two sine waves. The goal is to dis-

[20] Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, et al. "Deep speech: Scaling up end-to-end speech recognition". In: *arXiv preprint arXiv:1412.5567* (2014)

tinguish signals composed of sine waves with periods $T_1 \sim \mathcal{U}(5,6)$ and $T_2 \sim \mathcal{U}(13,15)$, each with independent phase, from signals composed of sine waves with periods $T_1 \sim \{\mathcal{U}(1,5) \cup \mathcal{U}(6,100)\}$ and $T_2 \sim \{\mathcal{U}(1,13) \cup \mathcal{U}(15,100)\}$, again with independent phase. Despite being significantly more challenging, Fig. 5.6 demonstrates how quickly the Phased LSTM converges to the correct solution compared to the standard approaches, using exactly the same parameters. Additionally, the Phased LSTM appears to exhibit very low variance during training.

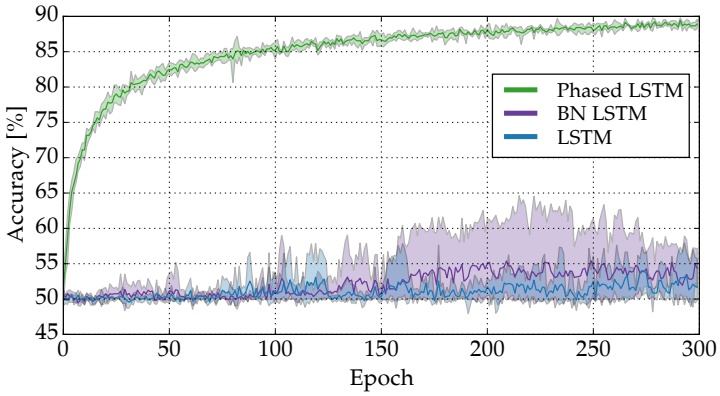


Figure 5.6: Accuracy during training for the superimposed frequencies task. The Phased LSTM outperforms both LSTM and BN-LSTM while exhibiting lower variance. Shading shows maximum and minimum over 5 runs, while dark lines indicate the mean.

5.3.2 *Adding Task*

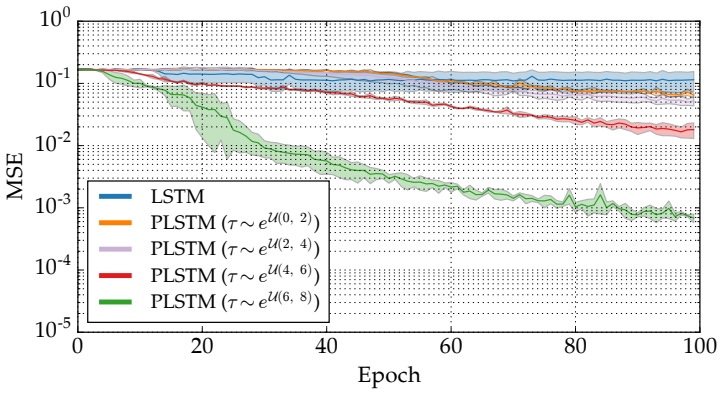


Figure 5.7: Mean-squared error over training on the addition task, with an input length of 500. Note that longer periods accelerate learning convergence.

To investigate how introducing time gates helps learning when long memory is required, an original LSTM task called the adding task [21] is revisited here. In this task, a sequence of random numbers is presented along with an indicator input stream. When there is a 0 in the indicator input stream, the presented value should be ignored; a 1 indicates that the value should be added. At the end of presentation the network produces a sum of all indicated values. Unlike the previous tasks, there is no inherent periodicity in the input, and it is one of the original tasks that LSTM was designed to solve well. This would seem to work against the advantages of Phased LSTM, but using a longer period for the time gate k_t

[21] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural Computation* 9.8 (1997), pp. 1735–1780

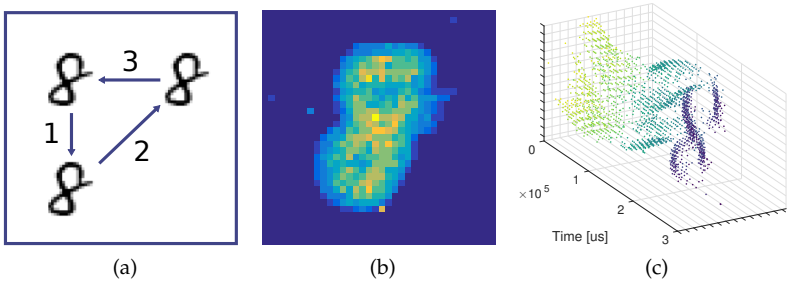
could allow more effective training as a unit opens only a for a few timesteps during training.

In this task, a sequence of numbers (of length 490 to 510) was drawn from $\mathcal{U}(-0.5,\ 0.5)$. Two numbers in this stream of numbers are marked for addition: one from the first 10% of numbers (drawn with uniform probability) and one in the last half (drawn with uniform probability), producing a model of a long and noisy stream of data with only few significant points. Importantly, this should challenge the Phased LSTM model because there is no inherent periodicity and every timestep could contain the important marked points.

The same network architecture is used as before. The period τ was drawn uniformly in the exponential domain, comparing four sampling intervals $\exp(\mathcal{U}(0,2)), \exp(\mathcal{U}(2,4)), \exp(\mathcal{U}(4,6))$, and $\exp(\mathcal{U}(6,8))$. Note that despite different τ values, the total number of LSTM updates remains approximately the same, since the overall sparseness is set by r_{on}. However, a longer period τ provides a longer jump through the past timesteps for the gradient during backpropagation-through-time.

Moreover, whether the model can learn longer sequences more effectively when longer periods are used can be investigated. Indeed, by varying the period τ, the results in Fig. 5.7 show longer τ accelerates training of the network to learn much longer sequences faster.

5.3.3 *N-MNIST Event-Based Visual Recognition*



(a) (b) (c)

Figure 5.8: N-MNIST experiment. (**a**) Sketch of digit movement seen by the image sensor. (**b**) Frame-based representation of an '8' digit from the N-MNIST dataset obtained by integrating all input spikes for each pixel. (**c**) Spatio-temporal representation of the digit, presented in three saccades as in (a). Note that this representation shows the digit more clearly than the blurred frame-based one.

To test performance on real-world asynchronously sampled data, the publicly-available N-MNIST [22] dataset for neuromorphic vision is used. The recordings come from an event-based vision sensor that is sensitive to local temporal contrast changes [23]. An event is generated from a pixel when its local contrast change exceeds a threshold. Every event is encoded as a 4-tuple $\langle x,\ y,\ p,\ t \rangle$ with position x, y of the pixel, a polarity bit p (indicating a contrast increase or decrease), and a timestamp t indicating the time when the event is generated. The recordings consist of events generated by the vision sensor while the sensor undergoes three saccadic movements facing a static digit from the MNIST dataset (Fig. 5.8a). An example of the event responses can be seen in Fig. 5.8c).

[22] G. Orchard, A. Jayawant, G. Cohen, and N. Thakor. "Converting static image datasets to spiking neuromorphic datasets using saccades". In: *Frontiers in Neuroscience* 9 (2015), p. 437. DOI: 10.3389/fnins.2015.00437. URL: http://journal.frontiersin.org/article/10.3389/fnins.2015.00437

[23] Christoph Posch, Teresa Serrano-Gotarredona, Bernabe Linares-Barranco, and Tobi Delbruck. "Retinomorphic event-based vision sensors: bioinspired cameras with spiking outputs". In: *Proc. of the IEEE* 102.10 (2014), pp. 1470–1484

In previous work using event-based input data [24,25], the timing information was sometimes removed and instead a frame-based representation was generated by computing the pixel-wise event-rate over some time period (as shown in Fig. 5.8(b)). Note that the spatio-temporal surface of events in Fig. 5.8(c) reveals details of the digit much more clearly than in the blurred frame-based representation.The Phased LSTM allows us to operate directly on such spatio-temporal event streams.

Table 5.1 summarizes classification results for three different network types: a CNN trained on frame-based representations of N-MNIST digits and two RNNs, a BN-LSTM and a Phased LSTM, trained directly on the event streams. Regular LSTM is not shown, as it was found to perform worse. The CNN was comprised of three alternating layers of 8 kernels of 5x5 convolution with a leaky ReLU nonlinearity and 2x2 max-pooling, which were then fully-connected to 256 neurons, and finally fully-connected to the 10 output classes. The event pixel address was used to produce a 40-dimensional embedding via a learned embedding matrix [26], and combined with the polarity to produce the input. Therefore, the network architecture was 41-110-10 for the Phased LSTM and 42-110-10 for the BN-LSTM, with the time given as an extra input dimension to the BN-LSTM.

Table 5.1 shows that Phased LSTM trains faster than alternative models and achieves much higher accuracy with a lower variance even within the first epoch of training. A factor ρ is further defined which represents the probability that an event is included, i.e. $\rho = 1.0$ means all events are included. The RNN models are trained with $\rho = 0.75$, and again the Phased LSTM achieves slightly higher performance than the BN-LSTM model. When testing with $\rho = 0.4$ (fewer events) and $\rho = 1.0$ (more events) without retraining, both RNN models perform well and greatly outperform the CNN. This is because the accumulated statistics of the frame-based input to the CNN change drastically when the overall spike rates are altered. The Phased LSTM RNNs seem to have learned a stable spatio-temporal surface on the input and are only slightly altered by sampling it more or less frequently.

Finally, as each neuron of the Phased LSTM only updates about 5% of the time, on average, 159 updates are needed in comparison to the 3153 updates needed per neuron of the BN-LSTM, leading to an approximate twenty-fold reduction in run time compute cost. It is also worth noting that these results form a new state-of-the-art accuracy for this dataset [27,28].

5.3.4 *Visual-Auditory Sensor Fusion for Lip Reading*

Finally, the use of Phased LSTM can be demonstrated on a task involving sensors with different sampling rates. Few RNN models ever attempt to merge sensors of different input frequencies, although the sampling rates can vary substantially. For this task,

[24] Daniel Neil and Shih-Chii Liu. "Effective Sensor Fusion with Event-Based Sensors and Deep Network Architectures". In: *IEEE Int. Symposium on Circuits and Systems (ISCAS)*. 2016, pp. 2282–2285

[25] Peter O'Connor, Daniel Neil, Shih-Chii Liu, Tobi Delbruck, and Michael Pfeiffer. "Real-time classification and sensor fusion with a spiking Deep Belief Network". In: *Frontiers in Neuroscience* 7 (2013)

[26] Sander Dieleman et al. *Lasagne: First release.* Aug. 2015. DOI: 10.5281/zenodo.27878. URL: http://dx.doi.org/10.5281/zenodo.27878

[27] G. Orchard, A. Jayawant, G. Cohen, and N. Thakor. "Converting static image datasets to spiking neuromorphic datasets using saccades". In: *Frontiers in Neuroscience* 9 (2015), p. 437. DOI: 10.3389/fnins.2015.00437. URL: http://journal.frontiersin.org/article/10.3389/fnins.2015.00437

[28] Gregory Kevin Cohen, Garrick Orchard, Sio Hoi Ieng, Jonathan Tapson, Ryad Benjamin Benosman, and André van Schaik. "Skimming Digits: Neuromorphic Classification of Spike-Encoded Images". In: *Frontiers in Neuroscience* 10.184 (2016). DOI: 10.3389/fnins.2016.00184

	CNN	BN-LSTM	Phased LSTM ($\tau = 100$ms)
Accuracy at Epoch 1	73.81% ± 3.5	40.87% ± 13.3	**90.32% ± 2.3**
Train/test $\rho = 0.75$	95.02% ± 0.3	96.93% ± 0.12	**97.28% ± 0.1**
Test with $\rho = 0.4$	90.67% ± 0.3	94.79% ± 0.03	**95.11% ± 0.2**
Test with $\rho = 1.0$	94.99% ± 0.3	96.55% ± 0.63	**97.27% ± 0.1**
LSTM Updates	–	3153 per neuron	**159 ± 2.8 per neuron**

Table 5.1: Accuracy on N-MNIST.

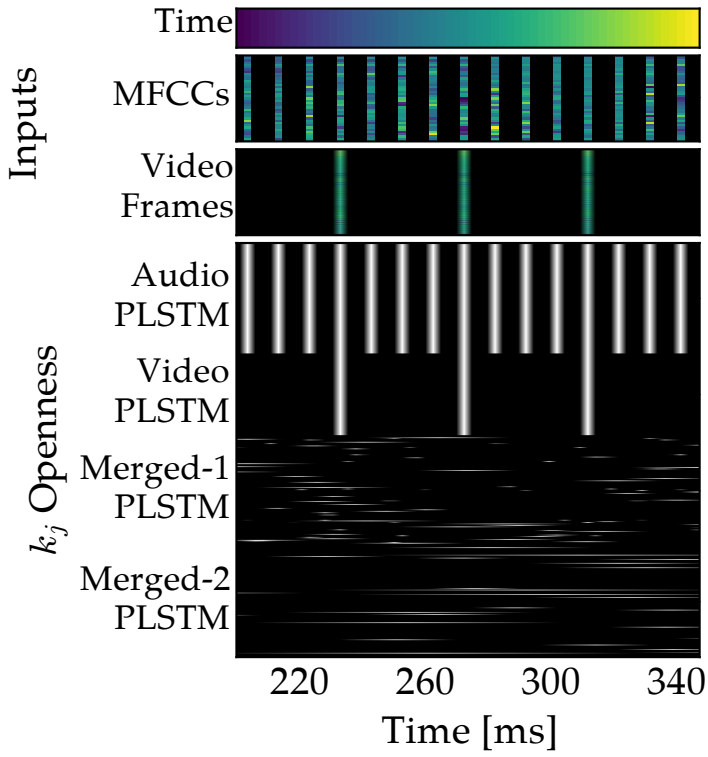


Figure 5.9: Inputs and openness of time gates for the lip reading experiment. Note that the 25fps video frame rate is a multiple of the audio input frequency (100 Hz). Phased LSTM timing parameters are configured to align to the sampling time of their inputs.

the GRID dataset [29] is used. This corpus contains video and audio of 30 speakers each uttering 1000 sentences composed of a fixed grammar and a constrained vocabulary of 51 words. The data was randomly divided into a 90%/10% train-test set. An OpenCV [30] implementation of a face detector was used on the video stream to extract the face which was then resized to grayscale 48x48 pixels. The goal here is to obtain a model that can use audio alone, video alone, or both inputs to robustly classify the sentence. However, since the audio alone is sufficient to achieve greater than 99% accuracy, sensor modalities were randomly masked to zero during training to encourage robustness towards sensory noise and loss.

The network architecture first separately processes video and audio data before merging them in two RNN layers that receive both modalities. The video stream uses three alternating layers of 16 kernels of 5x5 convolution and 2x2 subsampling to reduce the input of 1x48x48 to 16x2x2, which is then used as the input to 110 recurrent units. The audio stream connects the 39-dimensional MFCCs (13 MFCCs with first and second derivatives) to 150 recurrent units. Both streams converge into the *Merged-1* layer with 250 recurrent

[29] Martin Cooke, Jon Barker, Stuart Cunningham, and Xu Shao. "An audio-visual corpus for speech perception and automatic speech recognition". In: *The Journal of the Acoustical Society of America* 120.5 (2006), pp. 2421–2424

[30] Itseez. *Open Source Computer Vision Library*. https://github.com/itseez/opencv. 2015

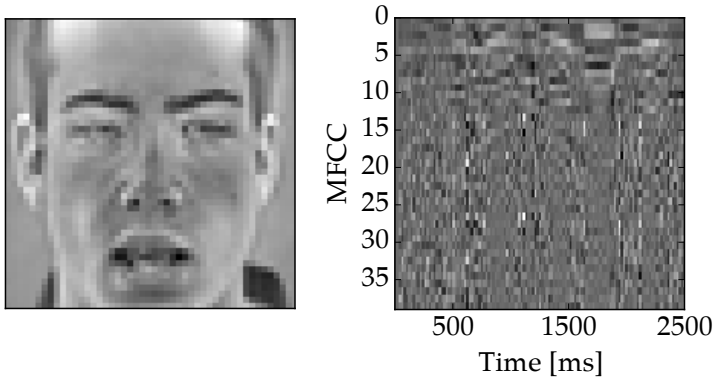


Figure 5.10: Example input of video (top) and audio (bottom).

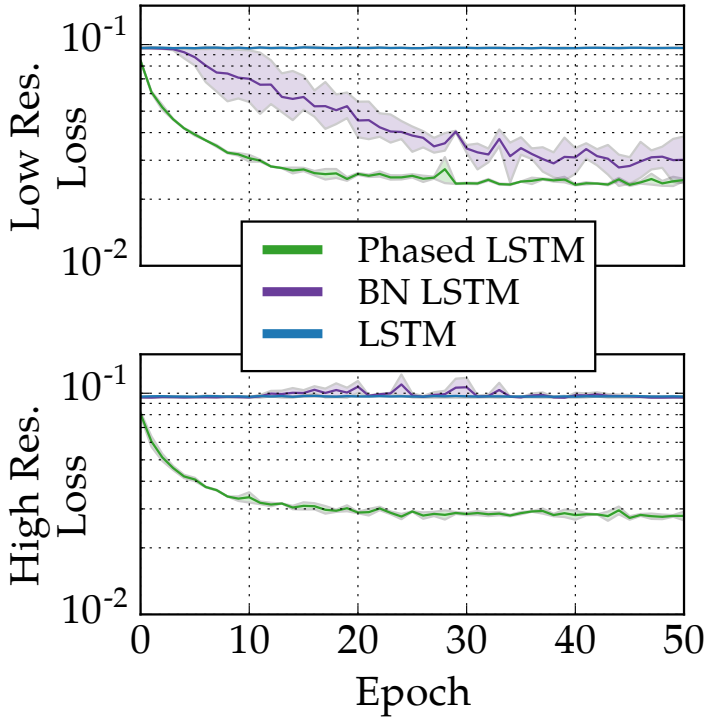


Figure 5.11: Test loss using the video stream alone. Video frame rate is 40ms. Top: low resolution condition, Mel-Frequency Cepstral Coefficientss (MFCCs) computed every 40ms with a network update every 40 ms; Bottom: high resolution condition, MFCCs every 10 ms with a network update every 10 ms.

units, and is connected to a second hidden layer with 250 recurrent units named *Merged-2*. The output of the Merged-2 layer is fully-connected to 51 output nodes, which represent the vocabulary of GRID. For the Phased LSTM network, all recurrent units are Phased LSTM units.

In the audio and video Phased LSTM layers, the open periods of the time gates are manually aligned to the sampling times of the inputs and disable learning of the τ and s parameters (see Fig. 5.9). This prevents presenting zeros or artificial interpolations to the network when data is not present. In the merged layers, however, the parameters of the time gate are learned, with the period τ of the first merged layer drawn from $\mathcal{U}(10,\ 1000)$ and the second from $\mathcal{U}(500,\ 3000)$. Fig. 5.10 shows a visualization of one frame of video and the complete duration of an audio sample.

During evaluation, all networks achieve greater than 98% accu-

racy on audio-only and combined audio-video inputs. However, video-only evaluation with an audio-video capable network proved the most challenging, so the results in Fig. 5.11 focus on these results (though result rankings are representative of all conditions). Two differently-sampled versions of the data were used: In the first "low resolution" version (Fig. 5.11, top), the sampling rate of the MFCCs was matched to the sampling rate of the 25 fps video. In the second "high-resolution" condition, the sampling rate was set to the more common value of 100 Hz sampling frequency (Fig. 5.11, bottom and shown in Fig. 5.9). The higher audio sampling rate did not increase accuracy, but allows for a faster latency (10ms instead of 40ms). The Phased LSTM again converges substantially faster than both LSTM and batch-normalized LSTM. The peak accuracy of 81.15% compares favorably against lipreading-focused state-of-the-art approaches [31] while avoiding manually-crafted features.

[31] Michael Wand, Jan Koutník, and Jürgen Schmidhuber. "Lipreading with Long Short-Term Memory". In: *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2016, pp. 6115–6119

5.4 Conclusion

The Phased LSTM has many surprising advantages. With its rhythmic periodicity, it acts like a learnable, gated Fourier transform on its input, permitting very fine timing discrimination. Alternatively, the rhythmic periodicity can be viewed as a kind of persistent dropout that preserves state [32], enhancing model diversity. The rhythmic inactivation can even be viewed as a shortcut to the past for gradient backpropagation, accelerating training. The presented results support these interpretations, demonstrating the ability to discriminate rhythmic signals and to learn long memory traces. Importantly, in all experiments, Phased LSTM converges more quickly and theoretically requires only 5% of the computes at runtime, while often improving in accuracy compared to standard LSTM. The presented methods can also easily be extended to GRUs [33], and it is likely that even simpler models, such as ones that use a square-wave-like oscillation, will perform well, thereby making even more efficient and encouraging alternative Phased LSTM formulations. An inspiration for using oscillations in recurrent networks comes from computational neuroscience [34], where rhythms have been shown to play important roles for synchronization and plasticity [35]. Phased LSTMs were not designed as biologically plausible models, but may help explain some of the advantages and robustness of learning in large spiking recurrent networks.

[32] Stanislau Semeniuta, Aliaksei Severyn, and Erhardt Barth. "Recurrent Dropout without Memory Loss". In: *arXiv* arXiv:1603.05118 (2016)

[33] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. "Learning phrase representations using RNN encoder-decoder for statistical machine translation". In: *arXiv preprint arXiv:1406.1078* (2014)

[34] György Buzsaki. *Rhythms of the Brain*. Oxford University Press, 2006

[35] Bernhard Nessler, Michael Pfeiffer, Lars Buesing, and Wolfgang Maass. "Bayesian computation emerges in generic cortical microcircuits through spike-timing-dependent plasticity". In: *PLoS Comput Biol* 9.4 (2013), e1003037

5.5 Discussion

This model presents a large step forward towards applying state-of-the-art models to event-driven data, as it can learn the complex spatio-temporal structure of event-based signals while operating in continuous time and preserving the advantages outlined in Chapter 1. The sparseness Phased LSTM exhibits is sparseness in space and sparseness in time, as only a small percentage of neurons are

active at any given point, which matches it well with the sparseness in space and time of the sensor. Moreover, drawing inspiration from the hardware design experiments in Chapter 2, the sparseness is predictable, as it obeys a simple cyclic formulation, which skirts the bottleneck of large, unpredictable memory access.

Phased LSTM also maintains the latency-accuracy and computation-accuracy tradeoffs at both the runtime point and at the architecture design point. A stationary representation, like an image producing spikes, can be continually improved over time through additional sequential computation (i.e., more timepoints processed by Phased LSTM). Moreover, like standard neural networks, more Phased LSTM neurons can be added to gain additional computation at the cost of more computation and higher latency, but Phased LSTM also permits a parameter tweak (Phased LSTM's r_{on}) to increase per-timestep computation without changing the number of neurons. This creates multiple levels of flexibility for computation.

Finally, as the model produces a new state for every input that arrives, assuming that at least a percentage of neurons are on, Phased LSTM exhibits pseudo-simultaneity. For every input timepoint, in continuous time, a partial computation of the output state is performed, permitting the network to be used in situations requiring extremely low latency.

Phased LSTM, however, has parameters and design choices that seem arbitrary. A systematic analysis of its advantages, where these advantages arise from, and whether the model can be simplified is important to determine whether further effort should be dedicated to this model, and whether it can be advantageous for future event-driven implementations.

6
Determining the Efficacy of a New Architecture

6.1 Introduction

The introduction of the Phased LSTM model laid out in Chapter 5 raises many new questions about how precisely this new model works, and whether it is the simplest possible implementation that yields the necessary advantages: efficacy on continuous-time signals, strong memory persistence across memory timescales, and predictable sparse computation patterns.

The purpose of this chapter is to investigate these questions, organized into several parts. Section 6.2 introduces the new models that are designed to stress particular aspects of the Phased LSTM model. Section 6.3 discusses the measure of merit that will be used when discussing these models. Section 6.4 showcases the results of the models on a variety of artificial and real-world tasks. Finally, Section 6.6 discusses the conclusions that can be drawn from these results. The text in Sections 6.2-6.6 will appear as part of a separate publication [1].

[1] Daniel Neil and Shih-Chii Liu. "Expanded Working Memory Enhances Phased LSTM". 2017

6.2 Models

6.2.1 LSTM

Gated models such as LSTM units [2] and GRU units [3] are important for modern deep RNN models. First, the update equations from their commonly-used version in [4]:

$$i_t = \sigma_i(x_t W_{xi} + h_{t-1} W_{hi} + w_{ci} \odot c_{t-1} + b_i) \tag{6.1}$$
$$f_t = \sigma_f(x_t W_{xf} + h_{t-1} W_{hf} + w_{cf} \odot c_{t-1} + b_f) \tag{6.2}$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \sigma_c(x_t W_{xc} + h_{t-1} W_{hc} + b_c) \tag{6.3}$$
$$o_t = \sigma_o(x_t W_{xo} + h_{t-1} W_{ho} + w_{co} \odot c_t + b_o) \tag{6.4}$$
$$h_t = o_t \odot \sigma_h(c_t) \tag{6.5}$$

These equations differ from classical RNNs with their use of gating functions, which are continuously-valued mixing factors which range between 0 and 1. Linear mixing preserves the gradient better than classical RNNs over multiple timesteps, as the neuron is able to control to what extent it updates its internal state with respect

[2] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural Computation* 9.8 (1997), pp. 1735–1780

[3] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. "Learning phrase representations using RNN encoder-decoder for statistical machine translation". In: *arXiv preprint arXiv:1406.1078* (2014)

[4] Alex Graves. "Generating sequences with recurrent neural networks". In: *arXiv preprint arXiv:1308.0850* (2013)

to the current state and inputs. These gate activation vectors, i_t, f_t, o_t, represent the input, forget, and output gates respectively. The value each neuron stores is the internal cell activation vector c_t, with x_t and h_t representing the input vector and the hidden state vector respectively. To constrain the gates to lie between 0 and 1, the nonlinearity σ_i, σ_f, σ_o for the input, forget, and output gates use the sigmoidal nonlinear transformation $y = 1/(1+e^{-x})$. Each gate has a weight parameter for the input x and the hidden state h, resulting W_{xi} and W_{hi}, W_{xf} and W_{hf}, W_{xo} and W_{ho} for the input, forget, and hidden gates. Each also has a bias, b_i, b_f, and b_o. The cell state c_t is updated with a linear interpolation using the elementwise (Hadamard) product $\odot$ of the previous cell state, controlled by f_t, and a new cell state controlled by i_t. The cell state is then further transformed by the output gate o_t to produce a new hidden state h_t. Optional peephole connections [5] w_{ci}, w_{cf} and w_{co} use the cell state c_t to further influence the input, forget, and output gates.

[5] Felix A Gers and Jürgen Schmidhuber. "Recurrent nets that time and count". In: *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks (IJCNN)*. vol. 3. IEEE. 2000, pp. 189–194

6.2.2 *Phased LSTM*

The Phased LSTM model, introduced in [6], extends the LSTM model with a novel time gate k_t. The time gate controls the operation of the neuron; as it acts multiplicatively, when the value of the time gate is closed the neuron performs no updates, and when its value lies close to one it updates as a standard LSTM cell. The opening and closing is a periodic oscillation controlled by three parameters. The first parameter, the period τ, controls the duration of an entire cyclical open and close period. The second, the shift s, applies an offset that adjusts the phase shift of the oscillation. The third, r_{on}, is the ratio of open duration to total period. With an r_{on} of 1, the Phased LSTM would always update, and with an r_{on} of 0, the Phased LSTM would always remain closed and never update. To encourage long memory and sparsity, r_{on} is typically set to about 10%.

[6] Daniel Neil, Michael Pfeiffer, and Shih-Chii Liu. "Phased LSTM: Accelerating Recurrent Network Training for Long or Event-based Sequences". In: *Advances in Neural Information Processing Systems*. 2016, pp. 3882–3890

The openness of the time gate k_t can be calculated thusly:

$$\phi_{i,t} = \frac{(t - s_i) \bmod \tau_i}{\tau_i} \tag{6.6}$$

$$k_{i,t} = \begin{cases} \dfrac{2\phi_{i,t}}{r_{on,i}}, & \text{if } \phi_{i,t} < \dfrac{1}{2} r_{on,i} \\ 2 - \dfrac{2\phi_{i,t}}{r_{on,i}}, & \text{if } \dfrac{1}{2} r_{on,i} < \phi_{i,t} < r_{on,i} \\ \alpha\phi_{i,t}, & \text{otherwise} \end{cases} \tag{6.7}$$

The neuron index i has be added to the parameters for clarity to demonstrate which parameters and variables are neuron-specific ($\phi_{i,t}$, $k_{i,t}$, s_i, τ_i, $r_{on,i}$) and which are global (t, α). The variable $\phi_{i,t}$ is an auxiliary variable which represents the percentage of the phase within the rhythmic cycle, ranging from 0% to 100%. The operation of the gate is specified in three piecewise phases: a rising opening

phase (during the first half of r_{on}), a falling opening phase (during the second half of r_{on}) and an off phase. During the off phase, note that there is a leak α, with analogy to the leaky rectified linear unit [7], which during training propagates important gradient information even while the gate is closed. After training, $\alpha = 0$ so as to require no update of the neuron at all during the off-phase. Furthermore, note that the linear slopes of the rising and falling phase have a constant gradient, preserving the strong gradient information that allows ReLUs to train so well.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". In: *The IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 1026–1034

Advantageously for Phased LSTM (PLSTM), the oscillation is defined at all continuous time points t, allowing irregularly-spaced time points t_j to be used within this RNN framework. As arbitrary time points can be used, Phased LSTM can natively work with event-driven, asynchronously-sampled input data. Here, the shorthand notation $c_j = c_{t_j}$ for cell states c at time t_j is used, with the other gates following analogously. Similarly, $c_{j-1} = c_{t_{j-1}}$ for previously-sampled times in continuous time. Then, the equations for the standard LSTM updates are rewritten, with a proposed cell update $\widetilde{c_j}$ and hidden state update $\widetilde{h_j}$ controlled by the time gate k_j:

$$
\begin{aligned}
i_j &= \sigma_i(x_j W_{xi} + h_{j-1} W_{hi} + w_{ci} \odot c_{j-1} + b_i) \\
f_j &= \sigma_f(x_j W_{xf} + h_{j-1} W_{hf} + w_{cf} \odot c_{j-1} + b_f) \\
\widetilde{c_j} &= f_j \odot c_{j-1} + i_j \odot \sigma_c(x_j W_{xc} + h_{j-1} W_{hc} + b_c) \\
c_j &= k_j \odot \widetilde{c_j} + (1 - k_j) \odot c_{j-1} \\
o_j &= \sigma_o(x_t W_{xo} + h_{j-1} W_{ho} + w_{co} \odot \widetilde{c_j} + b_o) \\
\widetilde{h_j} &= o_j \odot \sigma_h(\widetilde{c_j}) \\
h_j &= k_j \odot \widetilde{h_j} + (1 - k_j) \odot h_{j-1}
\end{aligned}
$$

Though this formulation appears to require more updates, it offers substantial speedups as a large proportion of the neurons are skipped in a timestep at runtime. For further information, refer to the formulation of Phased LSTM in [8].

[8] Daniel Neil, Michael Pfeiffer, and Shih-Chii Liu. "Phased LSTM: Accelerating Recurrent Network Training for Long or Event-based Sequences". In: *Advances in Neural Information Processing Systems*. 2016, pp. 3882–3890

A note on gradient flow is important here, because this technical detail can result in a variety of different behaviours. As the output is dependent on the opening phase, and the opening phase is dependent on the mod function, it is worth clarifying the derivative. Defining mod thusly:

$$
y = \operatorname{mod}(t, \tau) = t - \left\lfloor \frac{t}{\tau} \right\rfloor \tau \tag{6.8}
$$

(6.9)

The partial derivatives can be specified:

$$\frac{\partial y}{\partial t} = \frac{\partial t}{\partial t} - \tau \frac{\partial}{\partial t}\left(\left\lfloor \frac{t}{\tau} \right\rfloor\right) - \left\lfloor \frac{t}{\tau} \right\rfloor \frac{\partial}{\partial t}(\tau) \tag{6.10}$$

$$= 1 - 0 - 0 \tag{6.11}$$

$$\frac{\partial y}{\partial \tau} = \frac{\partial}{\partial \tau} t - \tau \frac{\partial}{\partial \tau}\left(\left\lfloor \frac{t}{\tau} \right\rfloor\right) - \left\lfloor \frac{t}{\tau} \right\rfloor \frac{\partial}{\partial \tau}(\tau) \tag{6.12}$$

$$= 0 - 0 - \left\lfloor \frac{t}{\tau} \right\rfloor \tag{6.13}$$

$$\frac{\partial y}{\partial t} = 1, \qquad \frac{\partial y}{\partial \tau} = -\left\lfloor \frac{t}{\tau} \right\rfloor, \qquad \frac{t}{\tau} \notin \mathcal{Z} \tag{6.14}$$

Note that the derivative of floor is always zero, as the function is flat everywhere except at points where it performs a step change and thus has an undefined derivative. Here, the restriction $\frac{t}{\tau} \notin \mathcal{Z}$ is disregarded as it is unlikely to precisely happen in a floating-point system and the derivative is mathematically correct everywhere else. Applying a minute amount of jitter would also resolve the issue.

The formulation of the derivative given here, however, raises a point: the gradient with respect to the period increases over time, and thus is not translationally invariant in time. If timesteps are used with large text corpora, the end of the corpus will more strongly influence the period than the beginning; for other applications, adding an offset in time or changing to Unix timestamps will also change the result. In practice, the effect has not yet caused problems, but an argument can be made to replace $\frac{\partial y}{\partial \tau} := -1$ for more consistent behaviour.

6.2.3 *Joint*

Phased LSTM provides a unique tradeoff between computation within a timestep and memory over time. By extending τ and decreasing r_{on}, more neuron states represent older moments in the input history. From the previous work in [9], it is clear that PLSTM excels at long-term memory. However for many tasks, including natural language processing, short-term time dependencies may be as important as long-term memories. The extended off-period of PLSTM neurons leaves neurons unable to respond for long stretches to short-term fluctuations in input that can be of great importance. To address this deficiency, a simple hybrid PLSTM model is developed in which a majority of neurons are PLSTM neurons which maintain long-term context, while a percentage of regular LSTM neurons are reserved to compute on shorter timescales and consistently update every time step.

A joint PLSTM-LSTM neural model can be implemented by

[9] Daniel Neil, Michael Pfeiffer, and Shih-Chii Liu. "Phased LSTM: Accelerating Recurrent Network Training for Long or Event-based Sequences". In: *Advances in Neural Information Processing Systems*. 2016, pp. 3882–3890

forcing the k_t gate fully open for the last percentage of neurons:

$$\phi_{i,t} = \frac{(t - s_i) \bmod \tau_i}{\tau_i} \tag{6.15}$$

$$\widetilde{k}_{i,t} = \begin{cases} \dfrac{2\phi_{i,t}}{r_{on,i}}, & \text{if } \phi_{i,t} < \dfrac{1}{2} r_{on,i} \\ 2 - \dfrac{2\phi_{i,t}}{r_{on,i}}, & \text{if } \dfrac{1}{2} r_{on,i} < \phi_{i,t} < r_{on,i} \\ \alpha\phi_{i,t}, & \text{otherwise} \end{cases} \tag{6.16}$$

$$k_{i,t} = \begin{cases} \widetilde{k}_{i,t} & \text{if } i/n < \Theta \\ 1 & \text{otherwise} \end{cases} \tag{6.17}$$

As above, the helper variable $\phi_{i,t}$ is introduced which represents the cycle position of neuron i of n neurons at time t, using phase shift s_i, period τ_i, on ratio $r_{on,i}$ and time-gate $k_{i,t}$. Here the notation $\widetilde{k}_{i,t}$ is introduced for the proposed time gate state, which is set to 1 (fully open), if the ratio of neuron index to total neurons i/n exceeds the PLSTM ratio of Θ.

6.2.4 *Random-Dropout LSTM: No periodicity*

A possible source of the advantages seen in PLSTM is the sparse opening pattern of the neurons. Neurons are exposed to fewer timesteps, allowing a more powerful gradient to be preserved through training and decreasing the number of effective timesteps to which a neuron is exposed. Unfortunately, it is not *a priori* clear whether the advantage lies with the sparsity or the periodicity. To determine which is the dominant factor, a neuron which has identical wake-sleep sparsity can be designed, yet without any learnable periodicity:

$$k_{i,t} = \begin{cases} 1, & \text{if } p_i \sim \mathcal{U}(0,1) < \Theta \\ 0, & \text{otherwise} \end{cases} \tag{6.18}$$

The neuron time gate $k_{i,t}$ for neuron i at time t is set to 1 if a random variable p_i drawn from the uniform distribution between 0 and 1, is less than the sparsity threshold Θ. Therefore the mean occupancy of the time gate, which is the mean number of on-states, will match the mean occupancy of a Phased LSTM model if $r_{on} = \Theta$. The success of the random dropout model (RndDrp) with r_{on} and Θ matched would imply that the *sparseness* is a driving factor in the advantages found for Phased LSTM. However, poor performance of the Random Dropout LSTM model implies that indeed the *periodicity* of the wake-sleep cycles have importance.

6.2.5 *Square-wave Phased LSTM: No edge gradients*

While the three-phase model with rising-, falling-, and off-phases does preserve the gradients along all points, it is not clear such a complex model is necessary. A simpler implementation would be a square wave oscillation, in which there are only two phases, an

on- and off-phase. Concretely, the time gate k_t from the standard PLSTM formulation can be replaced with a simplified formulation:

$$\phi_{i,t} = \frac{(t - s_i) \bmod \tau_i}{\tau_i}, \qquad k_{i,t} = \begin{cases} 1, & \text{if } \phi_{i,t} < r_{on,i} \\ \alpha\phi_{i,t}, & \text{otherwise} \end{cases} \tag{6.19}$$

During the open phase, the time gate is fully open without the rising and falling scaling associated with Phased LSTM. Unfortunately, no gradient information to the period τ is then provided during the open phase; however, during the off phase, the leak still allows learning of the period. The square-wave model therefore can test the importance of learning during the period τ during the open phase. If its performance exceeds the original PLSTM model, then either the reduction of the parameters needed or the flat open period provides an advantage over PLSTM.

6.2.6 *Cyclic LSTM: Fixed phase relationships*

Similarly, if individual periodicity of the PLSTM opening periods are the primary driver of the advantages, then using a single parameter to represent the period of a PLSTM neuron suffices to create a periodic oscillation. Each neuron has a fixed phase relationship to the other neurons, and a single timestep of opening allows just one parameter, the period τ to be used:

$$k_{i,t} = \begin{cases} 1, & \text{if round}(t \bmod \tau_i) = 0 \\ \alpha|\phi_{i,t} - 0.5|, & \text{otherwise} \end{cases} \tag{6.20}$$

Here the off-period is centered at half the phase, $\phi_t - 0.5$, in order to direct the gradient towards the first timestep. The off leak α is set as above to allow propagation of gradient information during the off phase. If the cyclic model succeeds, it provides further evidence that the primary advantage lies in the periodicity; if it does not nearly match the performance of PLSTM, then the parameters that encode the relationship between neurons, the phase shift s and the on duration ratio r_{on} must learn useful re-alignments of neuron activation patterns. Note that the period τ remains trainable in the formulation in Eq. 6.20.

6.2.7 *Refractory LSTM: Forced longest memories*

Finally, as an even more extreme possibility, even the phase relationship could be unnecessary, and only the long memory of the PLSTM neurons are required. This possibility can be tested by introducing a refractory end time $r_{i,t}$ to track the time when a neuron can emerge from an off, or refractory, period:

$$k_{i,t} = \begin{cases} 1, & \text{if } t > r_{i,t} \\ 0, & \text{otherwise} \end{cases} \tag{6.21}$$

$$r_{i,t+1} = \begin{cases} t + \tau_i, & \text{if } t > r_{i,t} \\ r_{i,t} & \text{otherwise} \end{cases} \tag{6.22}$$

The time gate $k_{i,t}$ thus is open as soon as the current time t exceeds the refractory end time $r_{i,t}$ for neuron i. Once it is updated, it re-enters a refractory state in which it remains fixed until reopening after the refractory period τ. The refractory model ensures that memories of the past are maintained and not overwritten, and also eliminates any dependency between neurons; waiting sufficiently long before delivering an input can bring all neurons out of the off-phase, while immediately then providing any inputs will place all neurons into a refractory phase. If this model succeeds, then the predominant factor in Phased LSTM's success must be its distant memory of events, which the refractory model maximizes.

6.3 *Measures*

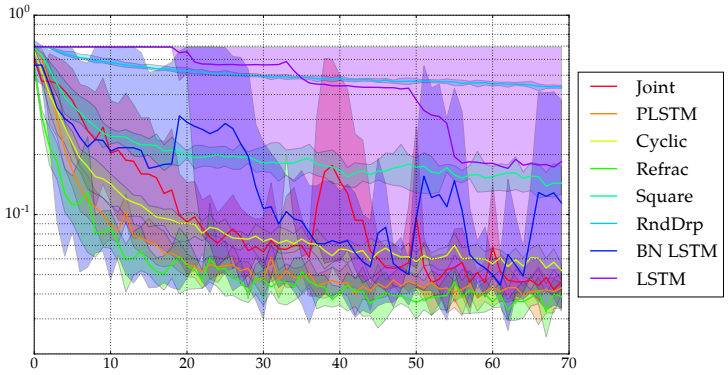


Figure 6.1: Error for the various models on the standard sampling condition of the first frequency task. The mean is displayed as a dark line, with semi-transparent maximum and minimum shaded around the line. Note the diversity of model behaviours over a fixed period of training epochs.

Comparison between neuron model types requires a measure that incorporates a variety of effects. As can be seen in Fig. 6.1, the behaviour of these models differs considerably. These error curves correspond to the standard sampling task introduced in [10], and consist of five runs from each model. Note that some models quickly converge, then drop back to chance, only to converge again; others exhibit very tight variance but high error; others have errors that vary across orders of magnitude. Models' ability to rapidly decrease the overall error, remain converged, and achieve acceptable final performance all varies quite considerably, necessitating a single measure that can encapsulate the result of these effects.

[10] Daniel Neil, Michael Pfeiffer, and Shih-Chii Liu. "Phased LSTM: Accelerating Recurrent Network Training for Long or Event-based Sequences". In: *Advances in Neural Information Processing Systems*. 2016, pp. 3882–3890

In this work, the *Integral of Error (IOE)* will be used as a primary error measure:

$$IOE = \sum_{n=1}^{N} \mathcal{L}(x, y) \tag{6.23}$$

The integral of error is the sum of all losses $\mathcal{L}$ for minibatches indexed $n = 1 \dots N$, composed of inputs x and targets y. Unlike final performance as a criteria, it incorporates the difficulty a model has in achieving high performance, penalizing models that take substantially more epochs to reach the same performance level. Unlike a time-to-convergence measure which records the number of epochs required to arrive at a satisfactory performance level, it

can always be defined and calculated, regardless of whether the model has converged. Additionally, the integral of error naturally incorporates errors that arise from models that are fragile and have difficulty maintaining convergence, increasing sharply in error after convergence or oscillating in performance. In addition to being easy to calculate, as it is simply the sum of all batchwise errors, useful statistics like variance and skewness of errors can be calculated for further information.

As the integral of error has arbitrary scaling and is dependent on the run length, it will only be used to compare between models. Furthermore, the resulting integral of error will be multiplicatively normalized here to yield useful comparisons against a reference run, e.g., the standard Phased LSTM model.

6.4 Experiments

This section will introduce a battery of experiments designed to stress different components of the models on a variety of artificial and real-world benchmarks.

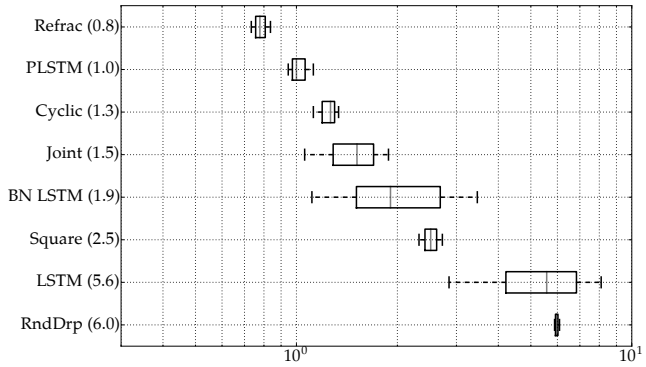
6.4.1 Frequency Analysis

The first experiment revisits the initial experiment of Phased LSTM [11]. The frequency discrimination task is particularly useful for elucidating the different axes of advantages that PLSTM offers, as data sequence length and sampling rate can be probed independently. In the task, a series of points $< y, t >$ are presented to the model arising from a sin wave of unknown period and phase shift, and the model must determine whether the points correspond to a sin wave with a period from a particular range from those with a period faster or slower than the target range. Classes are balanced, yielding a 50% chance rate.

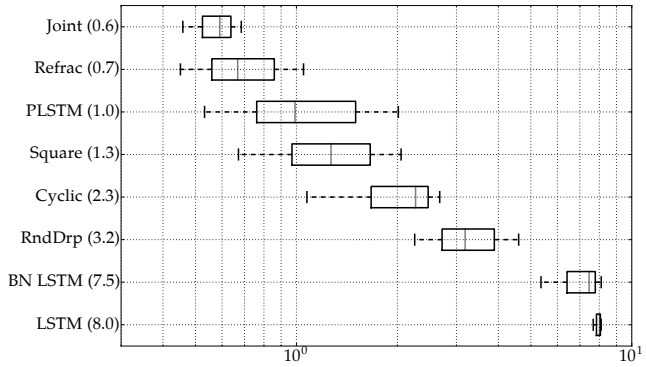
[11] Daniel Neil, Michael Pfeiffer, and Shih-Chii Liu. "Phased LSTM: Accelerating Recurrent Network Training for Long or Event-based Sequences". In: *Advances in Neural Information Processing Systems*. 2016, pp. 3882–3890

The models are tested under three conditions. The first (Fig. (a)), which corresponds to the dominant method in which RNNs are currently used, presents on average 70 datapoints at regularly-spaced timestep intervals. All models succeed at the task, and the error is primarily determined by how quickly the model is able to converge to a lower error. In the second (Fig. (b)), the sampling rate is increased by a factor of ten, leading to input sequences an order of magnitude longer. This test particularly probes a model's ability to quickly learn long sequences. Finally, in the third (Fig. (c)), the models receive the same number of input points as in the first but instead are sampled asynchronously. This test probes a model's ability to learn and generalize data which occurs in continuous time (asynchronous samples).

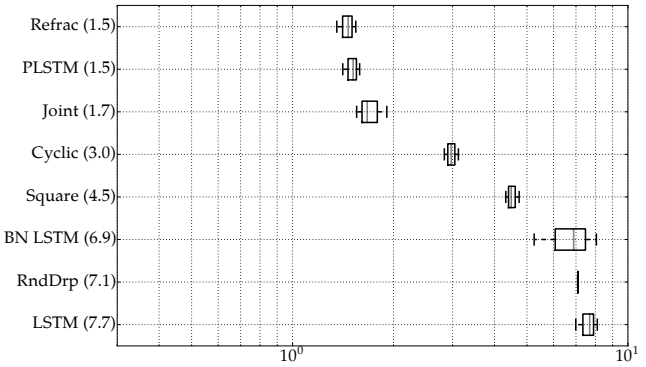
Note that in all figures, the integral of error measure has been set such that the error of PLSTM on the standard sampling condition is 1 to ease comparisons. For example, Fig. (c) shows that asynchronous sampling increases the IOE by 50% compared to hav-



(a)



(b)



(c)

Figure 6.2: Frequency discrimination task IOE. **(a)** Standard condition: the data is regularly sampled every 1 ms. **(b)** High resolution sampling condition: new input points are gathered every 0.1ms. **(c)** Asynchronous sampling condition: new input points are presented at intervals of 0.02 ms to 10 ms. Boxplots indicate the mean in gray, minimum and maximum with whisker lines, and the box extends to the lower and upper quartile of the results. IOE has been normalized such that the IOE of PLSTM on the standard task is 1.

ing evenly-stepped timesteps. In general, these results lay out the conclusion that will be echoed in the subsequent experiments: in general, PLSTM is among the best-performing models. The joint model, with long-term memory from PLSTM and short-term memory from LSTM, is nearly as performant and in some cases superior. Standard models without augmentation to aid in managing these sequences, such as batch-normalized LSTM and standard LSTM, tended to perform worse.

Combining Longer and Shorter-term Memory The joint model composed of both Phased LSTM (longer-term memory) and LSTM (shorter-term memory) does indeed perform acceptably well. Surprisingly, for the very long input sequences of the fast sampling rate in Fig. (b), it is the top performer, exceeding PLSTM's perfor-

mance and actually converging faster for the long sequence than it did for the short sequence, implying better sample complexity than the other models.

Sparsity vs. Periodicity The random dropout, cyclical, and refractory models allow investigating whether the advantages of PLSTM arise from sparsity or periodicity. Surprisingly, the random dropout model (RndDrp) matching the sparsity of PLSTM consistently performed among the worst for each input type, usually performing better than LSTM but not other models. The cyclical and refractory models, designed to test whether periodicity and the resulting long memory is important, often performed among the best for each category. Both taken together suggest that the *periodicity* is the dominant factor that allows PLSTM to be as performant as it is.

Alternative formulations The cyclical, refractory, and square models suggest possible alternative formulations for PLSTM with reductions in parameters and complexity. Indeed, the refractory model, which consistently maintains the longest neuron memories as they cannot be overwritten until after the refractory period, even exceeds the performance of PLSTM on all three tasks. The periodic model, perhaps because multiple subsequent inputs can both resolve to $\text{round}(t \bmod \tau_i) = 0$ and overwrite the earlier, does not perform as well. The square model, which removes the gradient information to the period during the on-phase but has a simpler activation profile, performs surprisingly poorly. Despite being as periodic as the cyclical model, maintaining timing learnable parameters, and not having to un-distort inputs manipulated by the rising and falling open phase of PLSTM, the model performs on average worse than the alternatives. Therefore, multiple sources of gradient information to the period appear to be important, at least in this task.

6.4.2 *Speaker Identification*

The MOCHA-TIMIT dataset was used for a real-world speaker identification task. Three speakers (two male, one female) were recorded speaking 460 sentences, which were then split into a 90%, 10% train/test split. Each sentence was transformed into 23 filterbank features from Kaldi [12], padded per batch to match in length, and used as the input to a network of 110 recurrent neurons of the given model type, followed by a fully-connected layer of 3 softmax output neurons which represent the probability of each target class. For models that require a time input, an input step index ranging $t \in (0, L)$ for a sequence of length L was used.

Five independent seeds were used to produce the results found in Fig. 6.3. The MOCHA task is used as an example of a real-world challenge without an inherent periodic nature, in which the sequences are moderately long (averaging around 390 timesteps). Most models achieve perfect performance by the end of training.

[12] Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian, Petr Schwarz, et al. "The Kaldi speech recognition toolkit". In: *IEEE 2011 workshop on automatic speech recognition and understanding*. EPFL-CONF-192584. IEEE Signal Processing Society. 2011

Note that in this simplistic and synchronous but real-world example, most PLSTM variants perform about equally well with the exception of the sparsity-matching random dropout (RndDrp) model. PLSTM, cyclical, joint, refractory, and even the square model to some extent all exhibit similar integral of errors. They do, however, perform better than standard models despite a synchronous and consistent timestep.

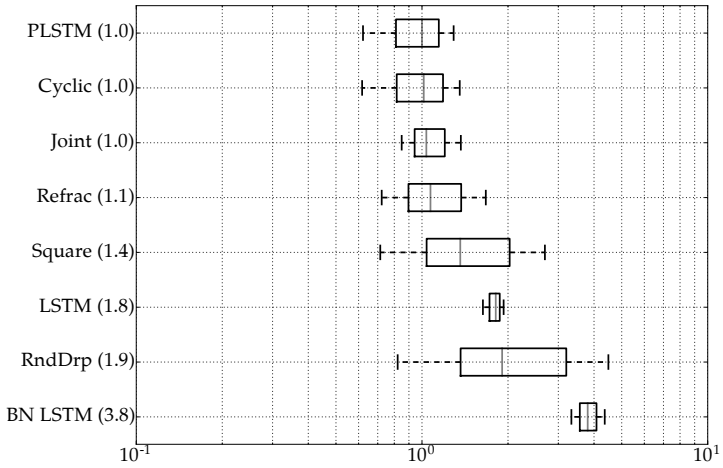


Figure 6.3: Integral of Error on MOCHA-TIMIT speaker identification task. Errors (in parantheses) are normalized such that PLSTM is 1. Note that most PLSTM variants perform approximately equally, with the notable exception of RndDrp, while standard LSTM and batch-normalized LSTM have more difficulty.

6.4.3 *Natural Language Processing*

One area that could be both fruitful and challenging for Phased LSTM is natural language processing. The long memory of PLSTM can greatly aid it in processing long documents, or interpreting documents processed at fine (e.g., character) resolution. However, the success of n-grams indicate the surprisingly important short-term context as well. Phased LSTM suffers under short-term context, as many fewer of its neurons are reliably available every timestep, and for now at least, training does not seem to yield solutions in which the neurons are consistently on 100% of the time. However, the joint model is perfect for merging the shorter-term memory of LSTM with the longer, trainable longer-term memory of PLSTM, so high performance would be expected of the joint model on Natural Language Processing (NLP) tasks.

As a difficult long-context task, we choose the enwiki8 Hutter 100MB Wikipedia encoding example [13]. The Hutter task consists of creating a model that predicts the next byte in the Wikipedia dataset, measuring the mean log-probability of the dataset. As the goal of this task is to compare models, smaller models than are used to achieve state-of-the-art results [14] are used here. The network is composed of a 30-dimensional embedding layer, three layers of 400 units, and a dense layer that connects to a softmax of 256 possible choices for the output byte. For models that require a time input, an input step index ranging $t \in (0, L)$ for a sequence of length L was used.

[13] Marcus Hutter. "The human knowledge compression contest". In: *http://prize.hutter1.net* (2012)

[14] Julian Georg Zilly, Rupesh Kumar Srivastava, Jan Koutník, and Jürgen Schmidhuber. "Recurrent highway networks". In: *arXiv preprint arXiv:1607.03474* (2016)

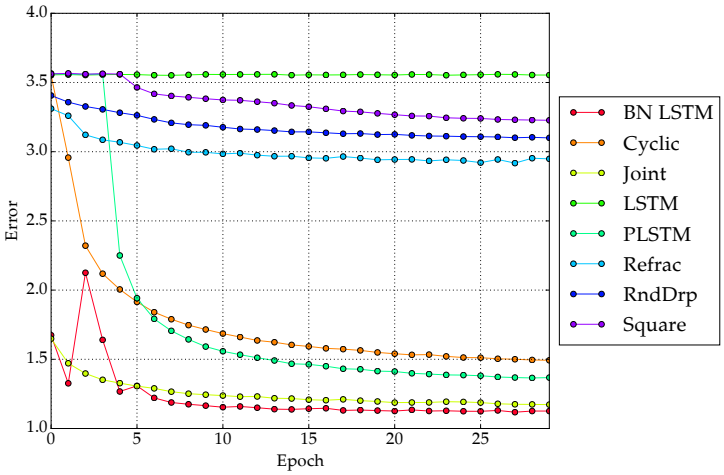


Figure 6.4: Validation error (nats) on the enwiki8 100MB Wikipedia dump. Due to the long computation time (approx. 1 hour/epoch on a GTX 980Ti), only a single run of each model is shown as a qualitative comparison.

The results can be found in Fig. 6.4. Many of the trends mentioned before, such as the lower performance of LSTM, the RndDrp model, and the square-wave model, are again borne out here. Surprisingly, batch-normalized LSTM does quite well here, after an initial period of instability. As expected, the joint model outperforms all other Phased LSTM models. The high performances of all the phased models are surprising, considering that only approximately 1/20th of the neurons are being used per timestep, compared to a standard recurrent model (with $r_{on} = 0.05$). This suggests that substantial redundancy exists in the standard recurrent models, and that perhaps dramatically scaling up the number of neurons (e.g., by 20x to match) in the periodic models could yield large performance improvements. Furthermore, the LSTM portion of the joint model could be similarly improved by applying batch normalization to its inputs.

6.5 *PLSTM Parameter Importance*

The Phased LSTM model contains timing parameters which have not yet been systematically examined. The purpose of this section is to study the effect of learning on the r_{on}, τ, and s parameters through denying learning or systematically corrupting these parameters after training.

6.5.1 *Presence & Absence of Learning*

A natural question that arises is the relative importance of learning the timing parameters. While previous work has similarly investigated sparse-in-time gated neural network models, prior models lacked the trainable timing parameters that PLSTM offers. By disabling updates to the timing parameters during training, the performance of PLSTM with and without learning can be elucidated.

The results can be found in Fig. 6.5. Solid lines correspond to the standard PLSTM with learning for all parameters, while dashed lines correspond to learning with training parameters disabled on

the frequency discrimination task. In the high-resolution sampling condition, little difference exists between learning and not learning the timing parameters (Fig. 6.5, green). However, convergence and final error is improved for the standard sampling condition (Fig. 6.5, purple) and a very large improvement exists for the asynchronous sampling condition (Fig. 6.5, blue). Indeed, training the parameters of Phased LSTM is important to yield good results for at least certain tasks.

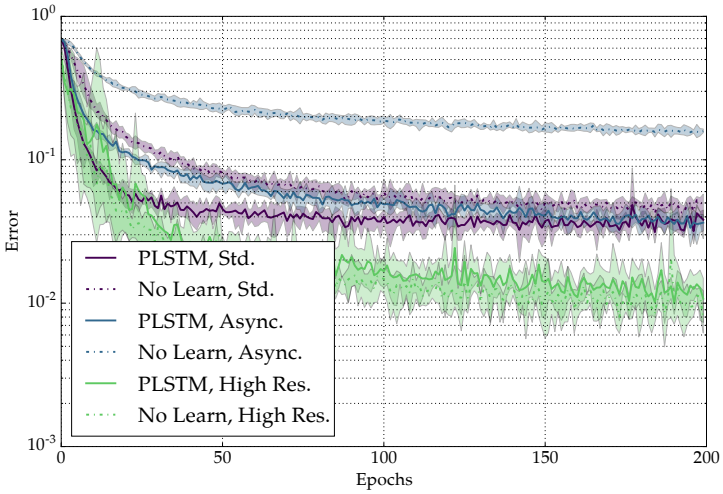


Figure 6.5: Comparison between learning (solid lines) and not learning (dashed lines) the timing parameters of Phased LSTM, under the three conditions of the frequency task. Minimum and maximum error for each epoch is shown in the shaded area, with the mean shown in a thick solid lines. Note a significant advantage to learning the timing parameters can be found for certain tasks.

6.5.2 *Parameter Ablation*

Finally, the relative importance for each parameter type can be investigated by selectively corrupting those parameters of the model. The parameter ablation process consists of the following procedure. First, a neural network model that has completed training (from Sec. 6.4.1) is loaded. Then, a random given percentage of neurons are selected (e.g., 10%), and their parameters of the chosen parameter type (e.g., periods) are permuted within the group. The random percentage is swept from 0% to 100%, and this is repeated 10 times for each model, of which there are five different random initializations.

The results of the parameter ablation can be found in Fig. 6.6. As the classes have been balanced, the chance rate is 50%. Error bars are the standard deviations for each point. As can be seen, the parameters are used to varying extents in different tasks, as the colored curves differ depending on the task. Permuting the periods randomly most greatly affects the accuracy of the system, where even at 50% permuted periods the models hover above chance for all tasks. The shifts are the next most sensitive parameter, surprisingly, and exhibit greater task-dependent results as the shift appears to most greatly affect the standard sampling condition. Finally, the r_{on} parameter appears substantially less important; even if the r_{on} values are completely shuffled within the network, the network does substantially better than chance.

One explanation for these results lies in the distribution of these parameters. Periods are exponentially distributed over many magnitudes, while the r_{on} parameters were all initialized to the same initial value and their final values remain close. Therefore, shuffling the similar parameters results in more similar results than shuffling parameters that differ more strongly. However, this methodology does reinforce earlier results that a specified-duration open period (given by r_{on}) is not as necessary a parameter to the model. The cyclic and refractory models both have a single on-step followed by a long series of off-steps, and yet perform quite competitively with PLSTM.

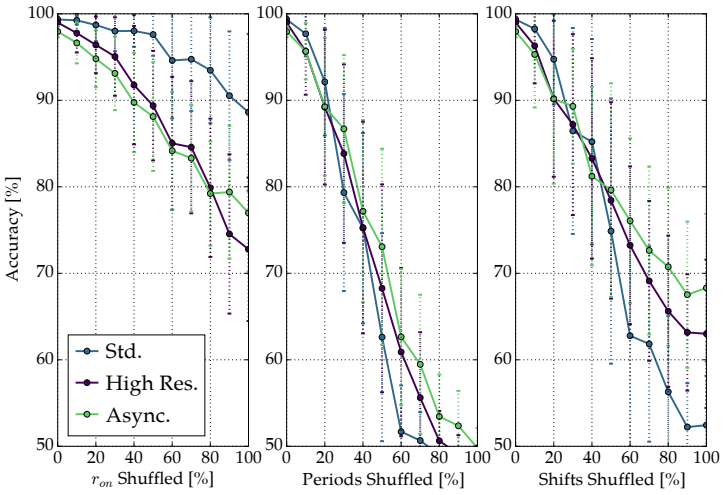


Figure 6.6: Parameter ablation for frequency tasks. Each plot demonstrates the shuffling of a particular parameter, after training. Across the horizontal axis, the percentage of parameters shuffled; across the vertical axis, the resulting decrease in accuracy. Error bars are standard deviations calculated from 5 different models, each with ten random shuffled parameter subsets at each point. Note the larger drop in accuracy for the period compared to the other two parameters; note also task-specific parameter sensitivity with the r_{on} more sensitive to parameter ablation in the high resolution and asynchronous sampling conditions.

6.6 *Conclusion & Discussion*

For many applications, and with inspiration from biology, combining shorter-term and longer-term memories is advantageous in order to extract temporal patterns across a wide range of temporal scales. Currently, even with the advantages offered by gating in LSTMs and GRUs, training for long sequences does not converge rapidly. Conversely, Phased LSTM, which trades off per-timestep computation in favor of longer memories, is optimized for longer-term memories and suffers when there are significant short-timestep interactions. In this work, we present the joint model that combines both of these systems to obtain higher performance across a wide variety of models. ‘ The Phased LSTM formulation succeeds well, as does the Joint formulation which combines the shorter and longer-term memory. The refractory model succeeds very well, emphasizing the importance of mixing in distant memories with the current state. Conversely, the random dropout model worked surprisingly poorly, emphasizing that the sparsity is not the main factor in the advantages of Phased LSTM, but rather the distant memory of the past, with predictable timing. This result, that distant memory is extremely important, is reinforced by the success of the cyclic and refractory models.

As many of these models have periodic open-close patterns, they perform a frequency decomposition on the input. This could perhaps underlie the advantage of WaveNet-style models that rely on dilated convolutions as well, which similarly do a frequency sampling of their input. While that makes intuitive sense for certain tasks like audio processing, it is perhaps surprising that this same principle works across the range of problem presented here, including textual analysis and visual analysis. Moreover, these models are used throughout the hierarchy, so they not only do a frequency decomposition of their inputs, but also of the intermediate neuron activations in response to this input. Future work can further explore whether a more explicit model of this timing analysis could yield better representations, using e.g., a differentiable frequency transform connected directly to a neuron model.

In the process of evaluating these models, it is quite clear that traditional models have far more computational capacity than is necessary. That is, with only a small fraction of neurons performing any computation at a given time, the networks are able to achieve new state-of-the-art accuracy levels. Part of the intuition for why this might be is that the amount of computation is roughly fixed per timestep in normal models, and the number of neurons is set to maximize performance - that is, the amount of computation is set by the maximum ever needed. On interesting and uninteresting timesteps alike, the network expends the same amount of computational effort which is likely to be the most ever needed. This principle is similar to the relationship between frame-based and event-based sensors, in which (traditionally) a full frame is read out regardless of the amount of new or relevant information in that image; in contrast, an event-based sensor only triggers computation when useful changes happen.

Could, perhaps, the principles of event-based *sensing* can be extended to event-based *computation* where computation is only triggered when sufficiently interesting computations happen? The following chapter will explore this idea.

7
Extending the Principle of Event-based Sensors to Computation

7.1 Introduction

The extremely computationally-sparse Phased LSTM model suggests that significant redundances exist in the computation of recurrent neural networks. One very straightforward optimization would be to skip unnecessary computations in a recurrent neural network if the activation has changed by an insignificant amount. Moreover, if this constraint included in training, it could allow the network to become robust to skipped computations, and to signal extra necessary computation through large recurrent state computation.

This chapter extends the principles of event-based sensing to traditional neural network models, skipping computations that arise from unchanging inputs. This is derived mathematically and extended empirically, and offers computational speedups in tasks that range from 6x-100x increases, which are significant savings for traditional models, and again emphasizing the importance of looking to apply principles from event-based sensing to state-of-the-art machine learning models.

Section 7.3 introduces the delta network concept for basic matrix-vector operations. Section 7.4 extends the formulation for a GRU RNN. Section 7.5 proposes an approximation method using a finite threshold for the deltas that offers greater speedups. Section 7.6 describes new training methods to optimally train Delta Networks. Section 7.7 shows the results of accuracy against speedup for three example tasks. The results are summarized in the final Section 7.8.

The text in Section 7.3-7.8 is in submission for another publication [1].

[1] Daniel Neil, Jun Haeng Lee, Tobi Delbruck, and Shih-Chii Liu. "Delta Networks for Optimized Recurrent Network Computation". In: *arXiv preprint arXiv:1612.05571* (2016)

7.2 Motivation

RNNs require many matrix-vector multiplications per layer to calculate the updates of neuron activations over time. RNNs also require a large weight memory storage that is expensive to allocate to on-chip static random access memory. In a 45nm technology, the

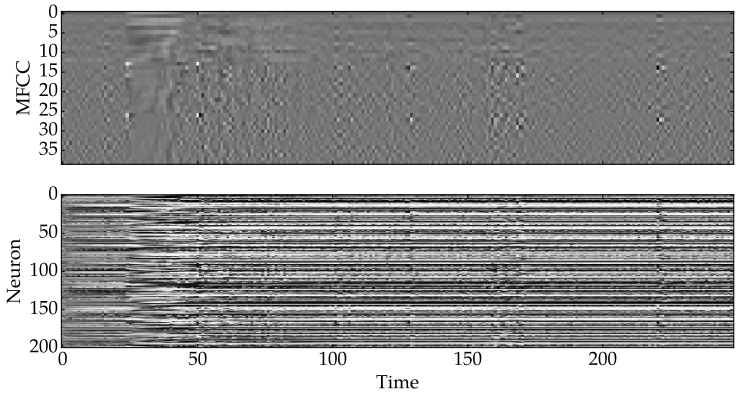


Figure 7.1: Stability in RNN activations over time. The top figure shows the continually-changing MFCC features for a spoken digit from the TIDIGITS dataset; the bottom figure shows the corresponding neural network output activations in response to these features. Note the slow evolution of the network states over timesteps.

energy cost of an off-chip dynamic 32-bit random access memory (SDRAM) access is about 2nJ and the energy for a 32-bit integer multiply is about 3pJ, so memory access is about 700 times more expensive than arithmetic [2]. Architectures can benefit from minimizing the use of this external memory. Previous work has focused on a variety of algorithmic optimizations for reducing compute and memory access requirements for deep neural networks. These methods include reduced precision for hardware optimization [3,4,5,6,7]; weight encoding, pruning, and compression [8,9]; and architectural optimizations [10,11,12]. However these studies did not consider the temporal properties of the data. Natural inputs to a neural network tend to have a high degree of temporal autocorrelation, resulting in slowly-changing network states. This slow-changing activation feature is also seen within the computation of RNNs processing audio inputs, for example, speech (Fig. 7.1).

Delta networks, as introduced here, exploit the temporal stability of both the input stream and the associated neural representation to reduce memory access and computation without loss of accuracy. By caching neuron activations, computations can be skipped where inputs change by a small amount from the previous update. Because each neuron that is not updated will save fetches of entire columns of several weight matrices, determining which neurons need to be updated offers significant speedups.

7.3 *Delta Network Formulation*

The purpose of a delta network is to transform a dense matrix-vector multiplication (for example, a weight matrix and a state vector) into a sparse matrix-vector multiplication followed by a full addition. This transformation leads to savings on both operations (actual multiplications) and more importantly memory accesses (weight fetches). Fig. 7.2 illustrates the savings due to a sparse multiplicative vector. Zeros are shown with white, while non-zero matrix and vector values are shown in black. Note the multiplicative effect of sparsity in the weight matrix and sparsity in the delta vector. In this example, 20% occupancy of the weight matrix and 20% occupancy of the Δ vector requires fetching and computing

[2] M. Horowitz. "1.1 Computing's energy problem (and what we can do about it)". In: *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*. Feb. 2014, pp. 10–14. DOI: 10.1109/ISSCC.2014.6757323

[3] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. "Binaryconnect: Training deep neural networks with binary weights during propagations". In: *Advances in Neural Information Processing Systems*. 2015, pp. 3123–3131

[4] Evangelos Stromatias, Daniel Neil, Michael Pfeiffer, Francesco Galluppi, Steve B Furber, and Shih-Chii Liu. "Robustness of spiking Deep Belief Networks to noise and reduced bit precision of neuro-inspired hardware platforms". In: *Frontiers in Neuroscience* 9 (2015)

[5] Matthieu Courbariaux and Yoshua Bengio. "Binarynet: Training deep neural networks with weights and activations constrained to+ 1 or-1". In: *arXiv preprint arXiv:1602.02830* (2016)

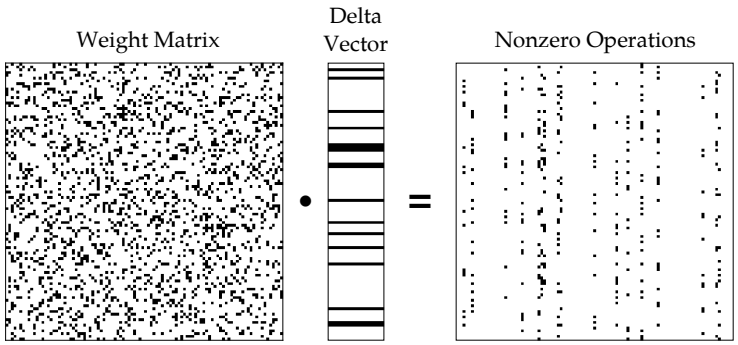


Figure 7.2: Illustration of saved matrix-vector computation using delta networks with sparse delta vectors and weight matrices.

only 4% of the original operations.

To illustrate the delta network methodology, consider a general matrix-vector multiplication of the form

$$r = Wx \tag{7.1}$$

that uses n^2 compute operations [13], $n^2 + n$ reads and n writes for a W matrix of size $n \times n$ and a vector x of size n. Now consider multiple matrix-vector multiplications for a long input vector sequence x_t indexed by $t = 1, 2, \ldots$. The result r_t can be calculated recursively as:

$$r_t = W\Delta + r_{t-1}, \tag{7.2}$$

where $\Delta = x_t - x_{t-1}$ and r_{t-1} is the result obtained from the previous calculation; if stored, the compute cost of r_{t-1} is zero as it can be fetched from the previous timestep. Trivially, $x_0 = 0$ and $r_0 = 0$. It is clear that

$$\begin{aligned} r_t &= W(x_t - x_{t-1}) + r_{t-1} && (7.3)\\ &= W(x_t - x_{t-1}) + W(x_{t-1} - x_{t-2}) + \ldots + r_0 && (7.4)\\ &= Wx_t && (7.5) \end{aligned}$$

Thus this formulation, which uses the difference between two subsequent timesteps and referred to as the *delta network* formulation, can be seen to produce exactly the same result as the original matrix-vector multiplication.

7.3.1 Theoretical Cost Calculation

To illustrate the savings if Δ from (7.2) is sparse, let us begin defining o_c to be the *occupancy* of a vector, that is, the percentage of nonzero elements in the vector.

Consider the compute cost for r_t; it consists of the total cost for calculating Δ (n operations for a vector of size n), adding in the stored previous result r_{t-1} (n operations), and performing the sparse matrix multiply $W\Delta$ ($o_c \cdot n^2$ operations for a W of size $n \times n$ and a sparse Δ vector of occupancy ratio o_c). Similarly, the memory cost for calculating r_t requires fetching $o_c \cdot n^2$ weights for W, $2n$ values for Δ, n values for r_{t-1} and writing out the n values for r_t.

[6] Steven K Esser, Paul A Merolla, John V Arthur, Andrew S Cassidy, Rathinakumar Appuswamy, Alexander Andreopoulos, David J Berg, Jeffrey L McKinstry, Timothy Melano, Davis R Barch, et al. "Convolutional Networks for Fast, Energy-Efficient Neuromorphic Computing". In: *Proceedings of the National Academy of Sciences* 113.41 (2016), pp. 11441–11446

[7] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. "Xnor-net: Imagenet classification using binary convolutional neural networks". In: *European Conference on Computer Vision*. Springer, 2016, pp. 525–542

[8] Song Han, Huizi Mao, and William J Dally. "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding". In: *CoRR, abs/1510.00149* 2 (2015)

[9] Song Han, Junlong Kang, Huizi Mao, Yiming Hu, Xin Li, Yubin Li, Dongliang Xie, Hong Luo, Song Yao, Yu Wang, et al. "ESE: Efficient Speech Recognition Engine with Compressed LSTM on FPGA". in: *FPGA 2017; NIPS 2016 EMDNN workshop*. 2016

[10] Forrest N Iandola, Matthew W Moskewicz, Khalid Ashraf, Song Han, William J Dally, and Kurt Keutzer. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 1MB model size". In: *arXiv preprint arXiv:1602.07360* (2016)

[11] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going deeper with convolutions". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 1–9

[12] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. "Deep networks with stochastic depth". In: *European Conference on Computer Vision*. Springer. 2016, pp. 646–661

Overall, the compute cost for the standard formulation ($C_{\text{comp,dense}}$) and the new delta formulation ($C_{\text{comp,sparse}}$) will be:

$$C_{\text{comp,dense}} = n^2 \tag{7.6}$$
$$C_{\text{comp,sparse}} = o_c \cdot n^2 + 2n \tag{7.7}$$

while the memory access costs for both the standard ($C_{\text{mem,dense}}$) and delta networks ($C_{\text{mem,sparse}}$) can be seen from inspection as:

$$C_{\text{mem,dense}} = n^2 + n \tag{7.8}$$
$$C_{\text{mem,sparse}} = o_c \cdot n^2 + 4n \tag{7.9}$$

Thus, the arithmetic intensity (ratio of arithmetic to memory access costs) as $n \to \infty$ is 1 for both the standard and delta network methods. This means that every arithmetic operation requires a memory access, unfortunately placing computational accelerators at a disadvantage. However, if a sparse occupancy o_c of Δ is assumed, then the decrease in computes and memory accesses due to storing the previous state will result in a speedup of:

$$C_{\text{dense}}/C_{\text{sparse}} \approx n^2/(o_c \cdot n^2) = (1/o_c) \tag{7.10}$$

For example, if $o_c = 10\%$, then the theoretical speedup will be 10X. Note that this speedup is determined by the occupancy in each computed $\Delta = x_t - x_{t-1}$, implying that this sparsity is determined by the data stream. Specifically, the regularity with which values stay exactly the same between x_t and x_{t-1}, or as demonstrated later, within a certain absolute value called the threshold, determines the speedup. In a neural network, x can represents inputs, intermediate activation values, or outputs of the network. If x changes slowly between subsequent timesteps then the input values x_t and x_{t-1} will be highly redundant, leading to a low occupancy o_c and a correspondingly increased speedup.

[13] In this paper, a "compute" operation is either a multiply, an add, or a multiply-accumulate. The costs of these operations are similar, particularly when compared to the cost of an off-chip memory operation. See (Horowitz, "1.1 Computing's energy problem (and what we can do about it)") for a simple comparison of energy costs of compute and memory operations.

7.4 Delta Network GRU

In GRUs, the matrix-vector multiplication operation that can be replaced with a delta network operation appears several times, shown in bold below. This GRU formulation is from [14]:

$$r_t = \sigma_r(\mathbf{W_{xr}x_t} + \mathbf{W_{hr}h_{t-1}} + b_r) \tag{7.11}$$
$$u_t = \sigma_u(\mathbf{W_{xu}x_t} + \mathbf{W_{hu}h_{t-1}} + b_u) \tag{7.12}$$
$$c_t = \sigma_c(\mathbf{W_{xc}x_t} + r_t \odot (\mathbf{W_{hc}}h_{t-1}) + b_c) \tag{7.13}$$
$$h_t = (1 - u_t) \odot h_{t-1} + u_t \odot c_t \tag{7.14}$$

Here r, u, c and h are reset and update gates, candidate activation, and activation vectors, respectively, typically a few hundred elements long. The σ functions are nonlinear logistic sigmoids that saturate at 0 and 1. The $\odot$ signifies element-wise multiplication. Each term in bold can be replaced with the delta update defined in

[14] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. "Empirical evaluation of gated recurrent neural networks on sequence modeling". In: *arXiv preprint arXiv:1412.3555* (2014)

(7.2), forming:

$$\begin{aligned}
\Delta_x &= x_t - x_{t-1} && (7.15)\\
\Delta_h &= h_{t-1} - h_{t-2} && (7.16)\\
r_t &= \sigma_r(W_{xr}\Delta_x + z_{xr} + W_{hr}\Delta_h + z_{hr} + b_r) && (7.17)\\
u_t &= \sigma_u(W_{xu}\Delta_x + z_{xu} + W_{hu}\Delta_h + z_{hu} + b_u) && (7.18)\\
c_t &= \sigma_c(W_{xc}\Delta_x + z_{xc} + b_c \\
&\quad r_t \odot (W_{hc}\Delta_h + z_{hc})) && (7.19)\\
h_t &= (1 - u_t) \odot h_{t-1} + u_t \odot c_t && (7.20)
\end{aligned}$$

where the values z_{xr}, z_{xu}, z_{xc}, z_{hr}, z_{hu}, z_{hc} are recursively defined as the the stored result of the previous computation for the input or hidden state, i.e.:

$$z_{xr} := \qquad z_{xr,t-1} = \qquad W_{xr}(x_{t-1} - x_{t-2}) + z_{xr,t-2} \qquad (7.21)$$

The above operation can be applied for the other five values z_{xu}, z_{xc}, z_{hr}, z_{hu}, z_{hc}. The initial condition at time x_0 is $z_0 := 0$. Also, many of the additive terms in the equations above, including the stored full-rank pre-activation states as well as the biases, can be merged into single values resulting into four stored memory values (M_r, M_u, M_{xc}, and M_{hr}) for the three gates:

$$M_{t-1} := z_{x,t-1} + z_{h,t-1} + b \qquad (7.22)$$

Finally, in accordance with the above definitions of the initial state, the memories M are initialized at their corresponding biases, i.e., $M_{r,0} = b_r$, $M_{u,0} = b_u$, $M_{xc,0} = b_c$, and $M_{hr,0} = 0$, resulting in the following full formulation of the delta network GRU:

$$\begin{aligned}
\Delta_x &= x_t - x_{t-1} && (7.23)\\
\Delta_h &= h_{t-1} - h_{t-2} && (7.24)\\
M_{r,t} &:= W_{xr}\Delta_x + W_{hr}\Delta_h + M_{r,t-1} && (7.25)\\
M_{u,t} &:= W_{xu}\Delta_x + W_{hu}\Delta_h + M_{u,t-1} && (7.26)\\
M_{xc,t} &:= W_{xc}\Delta_x + M_{xc,t-1} && (7.27)\\
M_{hc,t} &:= W_{hc}\Delta_h + M_{hc,t-1} && (7.28)\\
r_t &= \sigma_r(M_{r,t}) && (7.29)\\
u_t &= \sigma_u(M_{u,t}) && (7.30)\\
c_t &= \sigma_c(M_{xc,t} + r_t \odot M_{hc,t}) && (7.31)\\
h_t &= (1 - u_t) \odot h_{t-1} + u_t \odot c_t && (7.32)
\end{aligned}$$

7.5 *Delta Network Approximations*

The formulations described in Secs. 7.3 and 7.4 are designed to give precisely the same answer as the original computation in the network. However, a more aggressive approach can be taken in the update, inspired by recent studies that have shown the possibility of greatly reducing weight precision in neural networks

without giving up accuracy [15,16]. Instead of skipping a vector-multiplication computation if a change in the activation $\Delta = 0$, a vector-multiplication can be skipped if a value of Δ is smaller than the threshold (i.e $|\Delta_{i,t}| < \Theta$, where Θ is a chosen threshold value for a state i at time t). That is, if a neuron's hidden-state M activation has changed by less than Θ since it was last memorized, the neuron output will not be propagated, i.e., its Δ value is set to zero for that update. Using this threshold, the network will not produce precisely the same result at each update, but will produce a result which is approximately correct. Moreover, the use of a threshold substantially increases activation sparsity.

Importantly, if a non-zero threshold is used with a naive delta change propagation, errors can accumulate over multiple time steps through state drift. For example, if the input value x_t increases by nearly Θ on every time step, no change will ever be triggered despite an accumulated significant change in activation, causing a large drift in error. Therefore, in our implementation, the memory records the last value causing an above-threshold change, not the difference since the last time step.

More formally, the states $\hat{x}_{i,t-1}$ and $\hat{h}_{j,t-1}$ are introduced. These states store the $i-$th input and the hidden state of the $j-$th neurons, respectively, at their last *change*. The current input $x_{i,t}$ and state $h_{j,t}$ will be compared against these values to determine the Δ. Then the $\hat{x}_{i,t-1}$ and $\hat{h}_{j,t-1}$ values will only be updated if the threshold Θ is crossed. The equations are shown below for $\hat{x}_{i,t-1}$ with similar equations for $\hat{h}_{j,t-1}$:

$$\hat{x}_{i,t-1} = \begin{cases} x_{i,t-1} & \text{if } |x_{i,t} - \hat{x}_{i,t-1}| > \Theta \\ \hat{x}_{i,t-2} & \text{otherwise} \end{cases} \tag{7.33}$$

$$\Delta x_{i,t} = \begin{cases} x_{i,t} - \hat{x}_{i,t-1} & \text{if } |x_{i,t} - \hat{x}_{i,t-1}| > \Theta \\ 0 & \text{otherwise} \end{cases} \tag{7.34}$$

That is, when calculating the input delta vector $\Delta x_{i,t}$ comprised of each element i at time t, the difference between two values are used: the current value of the input $x_{i,t}$, and the value the last time the delta vector was nonzero $\hat{x}_{i,t-1}$. Furthermore, if the delta change is less than Θ, then the delta change is set to zero, producing a small approximation error that will be corrected when a sufficiently large change produces a nonzero update. The same formulation is used for the hidden state delta vector $\Delta h_{j,t}$.

This input approximation does not guarantee that the output error is bounded by the same threshold Θ. As supported by previous studies demonstrating that GRUs can be arbitrarily sensitive to input perturbations [17], the per-timestep error can grow with the input error. While thresholding should offer greater sparsity, the accumulation of these approximations could result in a diverging output error, therefore motivating the experiments in Sec. 7.7.1 that examine the effect of approximation on trajectory evolution.

[15] Evangelos Stromatias, Daniel Neil, Michael Pfeiffer, Francesco Galluppi, Steve B Furber, and Shih-Chii Liu. "Robustness of spiking Deep Belief Networks to noise and reduced bit precision of neuro-inspired hardware platforms". In: *Frontiers in Neuroscience* 9 (2015)

[16] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. "Low precision arithmetic for deep learning". In: *arXiv preprint arXiv:1412.7024* (2014)

[17] Thomas Laurent and James von Brecht. "A recurrent neural network without chaos". In: *arXiv preprint arXiv:1612.06212* (2016)

7.6 Methods to Increase Accuracy & Speedup

This section presents training methods and optimization schemes for faster and more accurate delta networks.

7.6.1 Training Directly on Delta Networks

The most principled method of training to minimize accuracy loss when running as a delta network would be to train directly on the delta network model. This should yield the best results as the network will receive errors that arise directly from the truncations of the delta network computation, and through training, learn to become robust to the types of errors that delta networks make.

More accurately, instead of training on the original GRU equations Eq. 7.11–7.14, the state is updated using the delta network model described in Eq. 7.23–7.34. Importantly, this change should incur no accuracy loss between train accuracy and test accuracy, though gradient descent may yet have more difficulty optimizing the model during training.

7.6.2 Rounding Network Activations

As the truncation of network activation due to the delta network is inherently non-differentiable, this training method should be compared against more widely used methods to verify its effectiveness. The delta network's computation can be viewed as analogous to the reduced-precision rounding training methods; small changes are rounded to zero while larger changes are propagated. Since many previous investigations have demonstrated methods to train networks to be robust against small rounding errors by rounding during training [18,19], these methods can be leveraged here to train a network that does not rely on small fluctuations in inputs. Low-precision computation and parameters can further reduce power consumption and improve the efficiency of the network for dedicated hardware implementations.

As explored in previous studies, a low-resolution activation θ_L in signed fixed-point format $Qm.f$ with m integer bits and f fractional bits can be produced from a high-resolution activation θ by using a deterministic and gradient-preserving rounding: $\theta_L = \text{round}(2^f \cdot \theta) \cdot 2^{-f}$ with $2^f \cdot \theta$ clipped to a bounding range $[-2^{m+f-1}, 2^{m+f-1}]$ to produce a quantized fixed-point activation. Thus, the output error cost forces the network to avoid quantization errors during training.

[18] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. "Low precision arithmetic for deep learning". In: *arXiv preprint arXiv:1412.7024* (2014)

[19] Evangelos Stromatias, Daniel Neil, Michael Pfeiffer, Francesco Galluppi, Steve B Furber, and Shih-Chii Liu. "Robustness of spiking Deep Belief Networks to noise and reduced bit precision of neuro-inspired hardware platforms". In: *Frontiers in Neuroscience* 9 (2015)

7.6.3 Adding Gaussian Noise to Network Activations

Random noise injection provides another useful comparison point. By injecting noise, the network will be unable to rely on small changes, and occasionally even larger changes will be incorrect (as may be the case of threshold rounding). This robustness can

be provided by adding Gaussian noise η to terms that will have a thresholded delta activation:

$$r_t = \sigma_r((x_t + \eta_x)W_{xr} + (h_{t-1} + \eta_h)W_{hr} + b_r) \tag{7.35}$$
$$u_t = \sigma_u((x_t + \eta_x)W_{xu} + (h_{t-1} + \eta_h)W_{hu} + b_u) \tag{7.36}$$
$$c_t = \sigma_c((x_t + \eta_x)W_{xc} + r_t \odot ((h_{t-1} + \eta_h)W_{hc}) + b_c) \tag{7.37}$$
$$h_t = (1 - u_t) \odot h_{t-1} + u_t \odot c_t \tag{7.38}$$

where $\eta \sim \mathcal{N}(\mu, \sigma)$. That is, η is a vector of samples drawn from the Gaussian distribution with mean μ and variance σ, and $\eta \in \{\eta_x, \eta_h\}$. Each element of these vectors is drawn independently. Typically, the value μ is set to 0 so that the expectation is unbiased, e.g., $\mathbf{E}[x_t + \eta_x] = \mathbf{E}[x_t]$.

As a result, the Gaussian noise should prevent the network from being sensitive to minor fluctuations, and increase its robustness to truncation errors.

7.6.4 *Considering Weight Sparsity*

In all training methods, considering the additional speedup from weight sparsity, in addition to skipping activation computation, should improve the theoretical speedup. Studies such as in [20] show that in trained low-precision networks, the weight matrices can be quite sparse. For example, in a ternary or 3-bit weight network the weight matrix sparsity can exceed 80% for small RNNs. Since every nonzero input vector element is multiplied by a column of the weight matrix, this computation can be skipped if the weight value is zero. That is, the zeros in the weight matrix act multiplicatively with the delta vector to produce even fewer necessary multiply-accumulates, as illustrated above in Fig. 7.2.

[20] Joachim Ott, Zhouhan Lin, Ying Zhang, Shih-Chii Liu, and Yoshua Bengio. "Recurrent Neural Networks With Limited Numerical Precision". In: *arXiv preprint arXiv:1608.06902* (2016)

The compute cost of the matrix-vector product will be $C_{\text{comp,sparse}} = o_m \cdot o_c \cdot n^2 + 2n$ and the memory cost will be $C_{\text{mem,sparse}} = o_m \cdot o_c \cdot n^2 + 4n$ for a weight matrix with occupancy o_m. By comparison to Eq. 7.10, the system can achieve a theoretical speedup of $1/(o_m \cdot o_c)$. That is, by compressing the weight matrix and only fetching nonzero weight elements that combine with the nonzero state vector, a higher speedup can be obtained without degrading the accuracy.

7.6.5 *Incurring Sparsity Cost on Changes in Activation*

Finally, a computation-specific cost can be associated with the delta terms and added to the overall cost. In an input batch, the L_1 norm for Δ_h can be calculated as the mean absolute delta changes, and this norm can be scaled by a weighting factor β. This L_{sparse} cost ($\mathcal{L}_{\text{sparse}} = \beta||\Delta h||_1$) can then be additively incorporated into the standard loss function. Here the L_1 norm is used to encourage sparse values in Δh, so that fewer delta updates are required.

7.7 Results

This section presents results demonstrating the trade-off between compute savings and accuracy loss, using Delta Network RNNs trained on the TIDIGITS digit recognition benchmark. Furthermore, it also demonstrates that the results found on small datasets also translate to the much larger Wall Street Journal speech recognition benchmark. The final example is for a CNN-RNN stack trained on end-to-end steering prediction using a recent driving dataset. The fixed-point $Q3.4$ (i.e. $m = 3$ and $f = 4$) format was used for network activation values in all speech experiments except the "Original" RNN line for TIDIGITS in Fig. 7.4, which was trained in floating-point representation. The driving dataset in Sec. 7.7.4 used $Q2.5$ activation. The networks were trained with Lasagne [21] powered by Theano [22]. Reported training time is for a single Nvidia GTX 980 Ti GPU.

[21] Sander Dieleman et al. *Lasagne: First release.* Aug. 2015. DOI: 10.5281/zenodo.27878. URL: http://dx.doi.org/10.5281/zenodo.27878

[22] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. "Theano: a CPU and GPU math expression compiler". In: *Proceedings of the Python for scientific computing conference (SciPy)*. Vol. 4. 2010, p. 3

7.7.1 TIDIGITS Dataset Trajectory Evolution

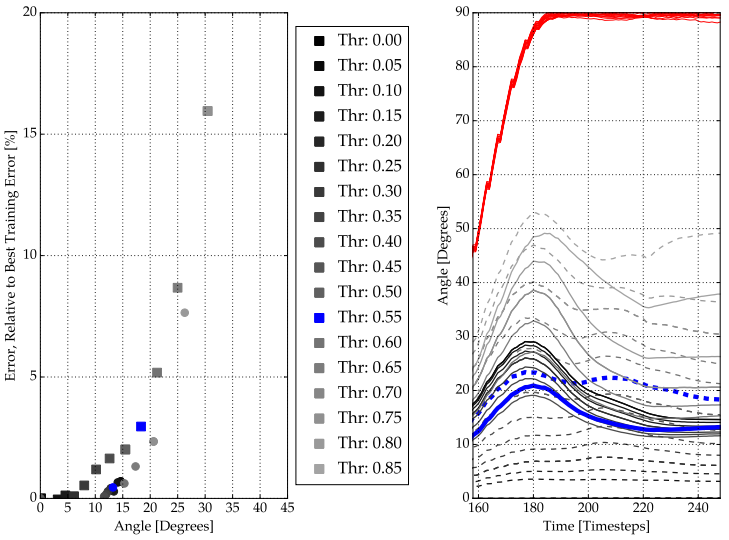


Figure 7.3: Comparison of trajectories over time by increasing Θ from 0 to 0.85 in steps of 0.05. At left, an increase of error angle between the final training state and the final thresholded state manifests as a decrease in accuracy, with the Gaussian-trained net as squares and Delta Network (DN)-trained net as circles. At right, the mean angle between the unapproximated state and the thresholded state over time. In red, the angle over time of an untrained network that has the same weight statistics as a trained network; in solid lines, a network that was trained as a delta network; in dashed lines, a network that was only trained with Gaussian noise. Curves for $\Theta = 0.55$ are highlighted in blue. Note that a DN-trained network has lower angle error, especially at higher thresholds, and an untrained net always quickly converges to an orthogonal state.

The TIDIGITS dataset was used as an initial evaluation task to study the trajectory evolution of delta networks. Single digits ("oh" and zero through nine), totalling 2464 digits in the training set and 2486 digits in the test set, were transformed in the standard way [23] to produce a 39-dimensional MFCC feature vector using a 25 ms window, 10 ms frame shift, and 20 filter bank channels. The labels for "oh" and "zero" were collapsed to a single label. Training time is approximately 8 minutes for a 150 epoch experiment.

The network architecture consists of a layer of 200 GRU units connected to a layer of 200 fully-connected units and finally to a classification layer for the 10 digit classes. First, a network was trained with Gaussian noise injection (Sec. 7.6.3) and subsequently tested using the delta network GRU formulation given in Sec. 7.4. A second network was trained directly on the delta network GRU

[23] Daniel Neil and Shih-Chii Liu. "Effective Sensor Fusion with Event-Based Sensors and Deep Network Architectures". In: *IEEE Int. Symposium on Circuits and Systems (ISCAS)*. 2016, pp. 2282–2285

formulation in accordance with Sec. 7.6.1, with the same architecture and $\Theta = 0.5$. Finally, a third network was constructed from the DN-trained network by permuting its weights to produce an "untrained" network with identical weight statistics.

To determine the robustness of the network to thresholded input, the trajectory evolution of these three networks were examined in comparison to their training conditions. Since the hidden states are bounded by (-1, 1) from the tanh nonlinearity, each 200-dimensional hidden state vector is normalized to construct a unit vector. Then, the error angle between the hidden state at training time and the hidden state with a threshold is measured. This error angle is correlated with the final accuracy, as seen in Fig. 7.3 left. The threshold is swept from 0 to 0.85, producing the results found in Fig. 7.3 right, in which each line represents the mean difference angle over all states across time. The figure begins at the median start point of a digit presentation (t=158), as the digits are pre-padded with zeros to match lengths.

A Gaussian-trained network's trajectory initially matches its training trajectory to produce a low error angle at low thresholds, which gradually increases as the threshold is raised. However, across a wide range of Θ, a DN-trained net's trajectory matches its training trajectory much more closely to produce a tighter-spaced arrangement and substantially lower angle error at higher threshold. Finally, an untrained network is indeed very sensitive to input approximations and quickly reaches an orthogonal representation, thus emphasizing the role of training to provide robustness.

7.7.2 *TIDIGITS Dataset Speedup and Accuracy*

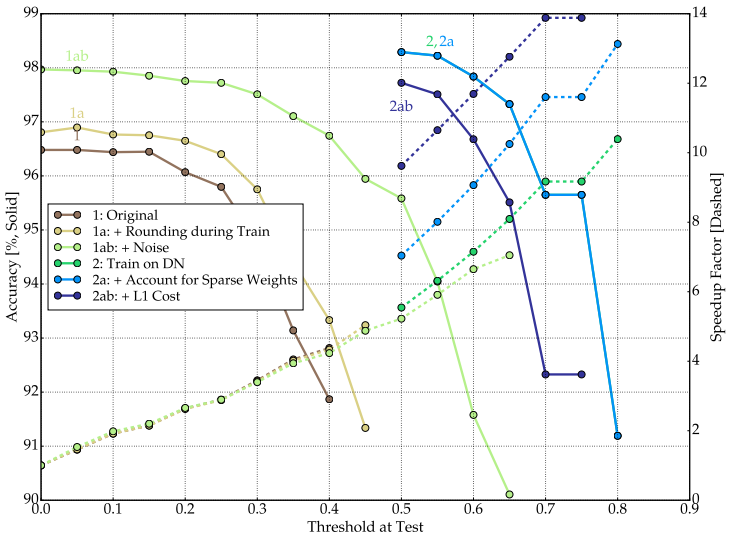


Figure 7.4: Test accuracy results from standard GRUs run as delta networks after training (curves **1**, **1a**, and **1ab**) and those trained as delta networks (curves **2**, **2a**, and **2ab**) under different constraints on the TIDIGITS dataset. The delta networks are trained for $\Theta = 0.5$, and the average of five runs is shown. Note that the methods are combined, hence the naming scheme. Additionally, the accuracy curve for **2** is hidden by the curve **2a**, since both achieve the same accuracy and only differ in speedup metric.

The results of applying the methods introduced in Sec. 7.6 can be found in Fig. 7.4. There are two quantities measured: the change in the number of memory fetches, and the accuracy as a function of the threshold Θ. Fig. 7.5 shows the same results, but removes the

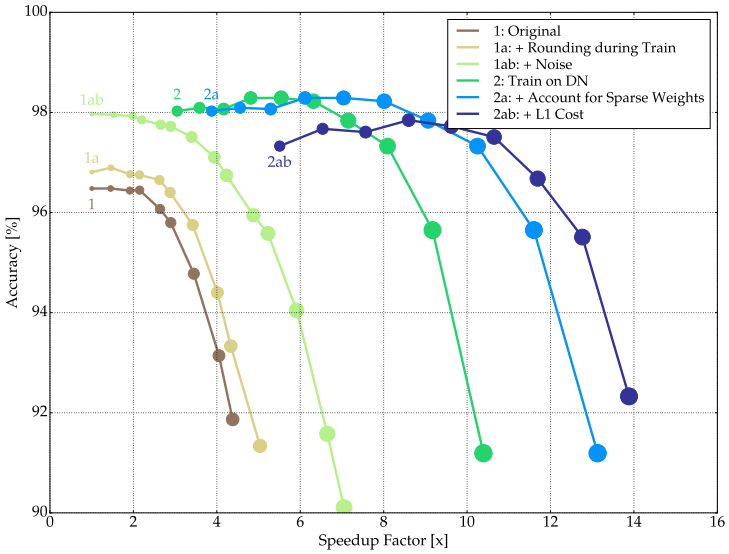


Figure 7.5: Accuracy-speedup tradeoff by adjusting Θ for TIDIGITS. By increasing Θ (indicated by sample point size), larger speedups can be obtained at greater losses of accuracy. For networks trained as delta networks, the training threshold is the first (leftmost) point in the line point sequence.

threshold axis to directly compare the accuracy-speedup tradeoff among the different training methods.

First, a standard GRU RNN achieving 96.59% accuracy on TIDIGITS was trained without data augmentation and regularization. This network has the architecture described in Sec. 7.7.1. It was then subsequently tested using the delta network GRU formulation given in Sec. 7.4.

The standard RNN run as a delta network ("Original") achieves 95% accuracy (a drop from zero delta threshold accuracy of 96%) with a speedup factor of about 2.2X. That is, only approximately 45% of the computes or fetches are needed in achieving this accuracy. By adding the rounding constraint during training ("+ Rounding during Training"), the accuracy is nearly 97% with an increase to a 3X speedup. By incorporating Gaussian noise ("+ Noise"), 97% accuracy can be maintained with a 5X speedup. Essentially, these methods added generalization robustness to the original GRU, while preventing small changes from influencing the network output. These techniques allow a higher threshold to be used while maintaining the same accuracy, therefore resulting in a decrease of memory fetches and a corresponding speedup.

The best model for training is the delta network itself ("Train on DN"). This network achieved 97.5% accuracy with a 8X speedup. Accounting for the pre-existing sparsity in the weight matrix ("+ Account for Sparse Weights"), the speedup increases to 10.5X, without affecting the accuracy (since it is the same network). Finally, incorporating an L1 cost on network changes in addition to training on the delta network model ("+ L1 cost") achieves 97% accuracy while boosting speedup to 11.9X. Adding in the final sparseness cost on network changes decreases the accuracy slightly since the loss minimization must find a tradeoff between both error and delta activation instead of considering error alone. However, using the L1 loss can offer a significant additional speedup while

retaining an accuracy increase over the original GRU network.

Finally, Fig. 7.5 also demonstrates the primary advantage given by each algorithm; an increase in generalization robustness manifests as an overall upward shift in accuracy, while an increase in sparsity manifests as a rightward shift in speedup. As proposed, methods 1a and 1b increase generalization robustness while only modestly influencing the sparsity. Method 2 greatly increases both, while method 2a only increases sparsity, and finally method 2ab slightly decreases accuracy but offers the highest speedup.

7.7.3 *Wall Street Journal Dataset*

The delta network methodology was applied to an RNN trained on the larger WSJ dataset to determine whether it could produce the same gains as seen with the TIDIGITS dataset. This dataset comprised 81 hours of transcribed speech, as described in [24]. Similar to that study, the first 4 layers of the network consisted of bidirectional GRU units with 320 units in each direction. Training time for each experiment was about 120h.

[24] Stefan Braun, Daniel Neil, and Shih-Chii Liu. "A Curriculum Learning Method for Improved Noise Robustness in Automatic Speech Recognition". In: *arXiv preprint arXiv:1606.06864* (2016)

Fig. 7.6 presents results on the achieved Word Error Rate (WER) and speedup on this dataset for two cases: First, running an existing speech transcription RNN as a delta network (results shown as solid curves labeled "RNN used as a DN"), and second, a network trained as a delta network with results shown as the dashed curves "Trained Delta Network". The speedup here accounts for weight matrix sparsity as described in Sec. 7.6.4 .

Surprisingly, the existing highly trained network already shows significant speedup without loss of accuracy as the threshold, Θ, is increased: At $\Theta = 0.2$, the speedup is about 5.5X with a WER of 10.8% compared with the WER of 10.2% at $\Theta = 0$. However, training theRNN to run as a delta network yields a network that achieves a slightly higher 5.7X speedup with the same WER. Thus, even the conventionally-trained RNN run as a delta network can provide greater than 5X speedup with only a 1.05X increase in the WER.

7.7.4 *Comma.ai Driving DataSet*

Driving scenarios are rapidly emerging as another area of RNN focused research. Here, the delta network model was applied to determine the gains of exploiting the redundancy of real-time video input. The open driving dataset from comma.ai [25] with 7.25 hours of driving data was used, with video data recorded at 20 Frames Per Second (FPS) from a camera mounted on the windshield. The network is trained to predict the steering angle from the visual scene similar to [26]. The approach in [27] is followed by using an RNN on top of the CNN feature detector. The CNN feature detector has three convolution layers without pooling layers and a fully-connected layer with 512 units. During training, the CNN feature detector was pre-trained with an analog output unit to learn the

[25] Eder Santana and George Hotz. "Learning a Driving Simulator". In: *arXiv preprint arXiv:1608.01230* (2016)

[26] Martin Hempel. "Deep Learning for Piloted Driving". In: *NVIDIA GPU Tech Conference*. 2016; Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Ziebam. "End to End Learning for Self-Driving Cars". In: *arXiv preprint arXiv:1604.07316* (2016)

[27] Martin Hempel. "Deep Learning for Piloted Driving". In: *NVIDIA GPU Tech Conference*. 2016

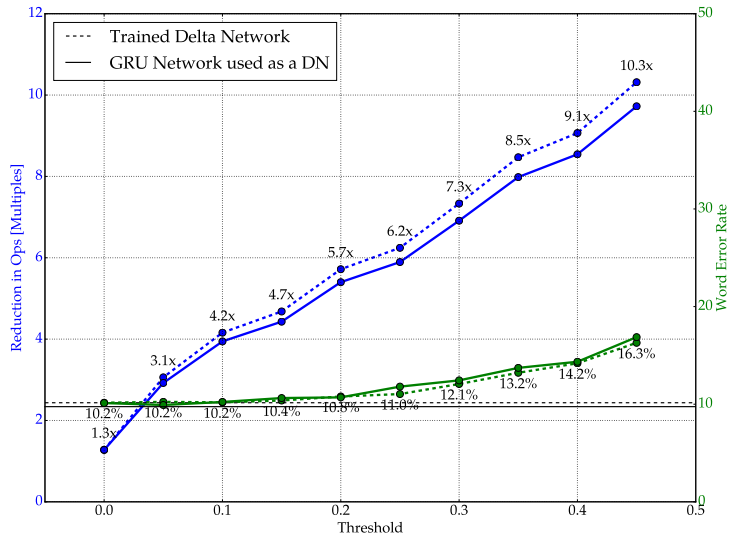


Figure 7.6: Accuracy and speedup tradeoffs on the WSJ dataset. The solid lines show results from an existing deep RNN run as a delta network. The dashed lines show results from a network trained as a delta network with $\Theta = 0.2$. The horizontal lines indicate the non-delta network accuracy level; similarly, the solid and dashed horizontal lines indicate the accuracy of the normal network and the DN network prior to rounding, respectively.

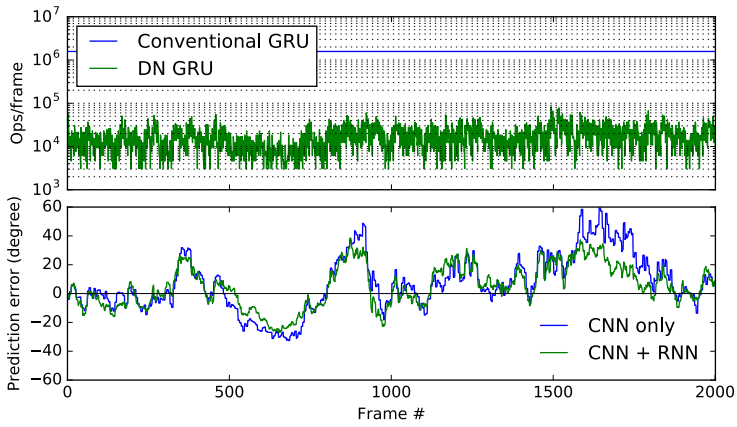


Figure 7.7: Reduction of RNN compute cost in the steering angle prediction task. Top figure shows the required # of ops per frame for the delta network GRU layer (trained with $\Theta = 0.1$) in comparison with the conventional GRU case. Bottom figure compares the prediction errors of CNN predictor and CNN+RNN predictor. The RNN slightly improves the steering angle prediction.

recorded steering angle from randomly selected single frame images. Afterwards, the delta network RNN was added, and trained by feeding sequences of the visual features from the CNN feature detector to learn sequences of the steering angle. Since the Q2.5 format was used for the GRU layer activations, the GRU input vectors were scaled to match the CNN output and the target output was scaled to match the RNN output.

However, this raw dataset results in a few practical difficulties and requires data preprocessing. By excluding the frames recorded during periods of low speed driving, the segments where the steering angle is not correlated to the direction of the car movement can be removed. Training time of the CNN feature detector was about 8h for 10k updates with the batch size of 200. Training of the RNN part took about 3h for 5k updates with the batch size of 32 samples consisting of 48 frames/sample.

A very large speedup exceeding 100X in the delta network GRU can be seen in Fig. 7.7, computed for the steering angle prediction task on 2000 consecutive frames (100s) from the validation set. While the number of operations per frame remains constant for the conventional GRU layer, those for the delta network GRU layer

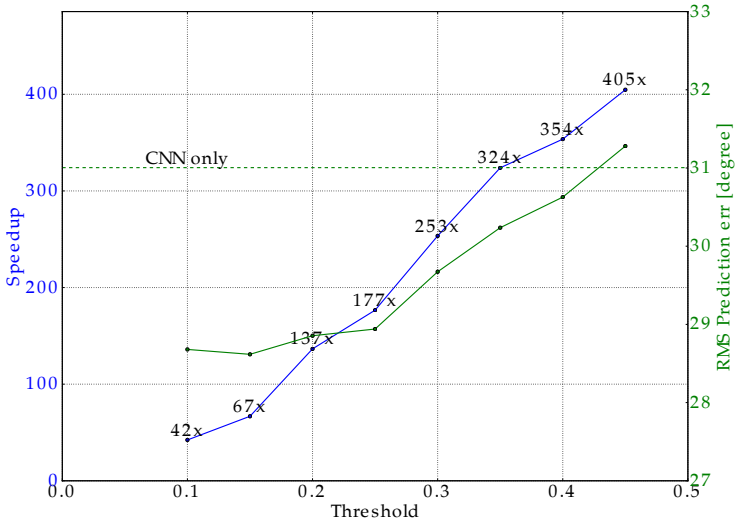


Figure 7.8: Tradeoffs between prediction error and speedup of the GRU layer on the steering angle prediction. The result was obtained from 1000 samples with 48 consecutive frames sampled from the validation set. Speedup here does not include weight matrix sparsity. The network was trained with $\Theta = 0.1$. A speedup of approximately 100X can be obtained without increasing the prediction error, using Θ between 0.1 and 0.25.

varies dynamically depending on the change of visual features.

In this steering network, the computational cost of the CNN (about 37 MOp/frame) dominates theRNN cost (about 1.58 MOp/frame), therefore the overall system-level computational savings is only about 4.2%. However, future applications will likely have efficient dedicated vision hardware or require a greater role for RNNs in processing numerous and complex data streams, which result in RNN models that consume a greater percentage of the overall energy/compute cost. Even now, the steering angle prediction network already benefits from a delta network approach.

7.8 *Discussion and Conclusion*

Although the delta network methodology can be applied to other network architectures, as was shown in similar concurrent work for CNNs [28], in practice a larger benefit is seen in RNNs because all the intermediate activation values for the delta networks are already stored between subsequent inputs. For example, the widely-used VGG19 CNN has 16M neuron states [29]. Employing the delta network approach for CNNs requires doubled memory access and significant additional memory space to store the entirety of the network state. Because the cost of external memory access is hundreds of times larger than that of arithmetic operations, delta network CNNs seem impractical without novel memory technologies to address this issue.

In contrast, RNNs have a much larger number of weight parameters than activations. The sparsity of the delta activations can therefore enable large savings in power consumption by reducing the number of memory accesses required to fetch weight parameters. CNNs, however, do not have this advantage since the weight parameters are few and shared by many units. Finally, the delta network approach is extremely flexible as pre-existing networks can be used without retraining, or trained specifically for increased

[28] Peter O'Connor and Max Welling. "Sigma Delta Quantized Networks". In: *arXiv preprint arXiv:1611.02024* (2016)

[29] Ken Chatfield, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. "Return of the devil in the details: Delving deep into convolutional nets". In: *arXiv preprint arXiv:1405.3531* (2014)

optimization.

7.9 Conclusion: Extending Event-based Principles to Computation

These recurrent networks were inspired by the way event-based sensors save on data transmission by not re-transmitting unchanging pixels. Here, in this chapter, the inherent temporal redundancy of neural activations over time was exploited to reduce computation, resulting in real-world speedups of 6x to 9x on speech applications and over 100x on steering angle prediction RNNs.

8
Conclusion, and Towards a Future of Event-based Machine Learning

The previous chapters have introduced many new algorithms and implementations to increase the state-of-the-art, both in accuracy and total-system cost of latency, computation, and power. These designs resulted from a systematic investigation of bridging the domains of deep learning with the advantages of event-based sensors such as the silicon retina [1] and the silicon cochlea [2]. Thankfully, this work has been fruitful in establishing directions that other authors have pushed forward as well, and, in some cases, the state-of-the-art lies far beyond what began in this thesis.

Chapter 2 introduced the Minitaur architecture for real-time classification of event-driven inputs. Other work [3,4] significantly sped up the Minitaur architecture and demonstrated it in a real-time, entirely-hardware, multi-modal audio-visual system. Similarly, the event-driven Deep Belief Network on SpiNNaker was also demonstrated in a real-time application [5]. However, it was clearer by these later developments that deep belief networks were not an ideal model for image classification; convolutional neural networks had begun to dominate image classification benchmarks [6], and extremely efficient platforms [7] provide exceedingly low-power, high-compute ASICs to efficiently acquire the result of the computation. While convolutional neural networks lacked the other advantages of event-based processing, their high efficiency and compatibility with traditional sensors make them extremely attractive targets for research. At least for now, it appears that static image classification is more closely aligned to be a problem for static sensors rather than dynamic, active sensors.

Chapter 3 demonstrated how to convert a standard frame-based Convolutional Neural Network into an event-driven spiking neural network, and showcased the work on the simple but well-studied dataset MNIST [8]. Work on this path has continued, overcoming challenges found when scaling up to more complex and interesting datasets. Now, networks as complex as VGG [9], trained on the large-scale image recognition challenge ImageNet [10] and even Residual Networks [11] can be converted to spiking networks with high ac-

[1] Patrick Lichtsteiner, Christoph Posch, and Tobi Delbruck. "A 128× 128 120 dB 15 μs latency asynchronous temporal contrast vision sensor". In: *IEEE Journal of Solid-State Circuits* 43.2 (2008), pp. 566–576

[2] S-C. Liu, A. van Schaik, B. Minch, and T. Delbrück. "Asynchronous Binaural Spatial Audition Sensor with $2 \times 64 \times 4$ Channel Output". In: *IEEE Trans. Biomed. Circuits Syst.* 8.4 (2014), pp. 453–464. DOI: 10.1109/TBCAS.2013.2281834

[3] Ilya Kiselev, Daniel Neil, and Shih-Chii Liu. "Event-Driven Deep Neural Network Hardware System for Sensor Fusion". In: *IEEE Int. Symposium on Circuits and Systems (ISCAS)*. 2016

[4] Ilya Kiselev, Daniel Neil, and Shih-Chii Liu. "Live demonstration: Event-Driven Deep Neural Network Hardware System for Sensor Fusion". In: *IEEE Int. Symposium on Circuits and Systems (ISCAS)*. 2016

[5] Evangelos Stromatias, Daniel Neil, Francesco Galluppi, Michael Pfeiffer, Shih-Chii Liu, and Steve Furber. "Event-Driven Deep Neural Network Hardware System for Sensor Fusion". In: *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*. 2015, pp. 1901–1901

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 770–778

[7] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks". In: *IEEE Journal of Solid-State Circuits* 52.1 (2017), pp. 127–138

curacy in the most recent advance [12]. However, in practice, given an event-based sensor, the methodology of converting a static convolutional network to a dynamic spiking network has been less appealing than the reverse: converting the dynamic input into a static frame compatible with a static convolutional network. That is, rather than taking a convolutional network and turning into a spiking one, practical experiments [13,14,15] - even ones that use dynamic event-based sensors - tend to bin the events into static frames. Unfortunately, the process of removing the dynamics by binning events into frames has been more thoroughly explored in the past, is easier to understand, and allows using state-of-the-art and industry standard tools [16] for the more challenging and computationally costly deep network training and execution. Further work continues to explore new methods to convert events into static frames [17] to attempt to maintain the advantages of both event-driven sensing and the ease of standard toolsets.

Chapter 4 introduced novel methods of optimizing neural network computation. These methods have been somewhat neglected because image classification, as a task, does not appear to be one which currently plays to the advantages of event-based sensors. These techniques, which aim to classify effectively with lower data integration time, could even be useful for convolutional networks that operate on frames of events to achieve lower latency. Another technique that has proved quite useful even for static convolutional networks is the low-precision work introduced in Chapter 2. Extensions of this work have produced a low-precision version of Caffe which incorporates hardware implementation constraints (such as bit precision of weights and activations) [18]. This software is now the dominant in-house method for training neural networks which work on the highly-optimized hardware platforms produced by our research group. In addition to low-precision constraints, perhaps future work can also introduce some of these methods to allow improved classification at reduced latencies.

Chapter 5 and Chapter 6 introduce the Phased LSTM model that extends a state-of-the-art recurrent network model with a time gate to make it compatible with continuous-time signals. To encourage PLSTM's use among the mainstream machine learning community, many of the introduced tasks are aimed at very conventional machine learning tasks, such as natural language processing and audio classification. However, while the experiments did indeed show advances over the state-of-the-art for these tasks, much of the improvement appeared to come from the sparse representation's long memory of the past.

Yet the Phased LSTM model offers so much more. The purpose of Phased LSTM is to work with continuous-time signals and expand the types of sensors to which state-of-the-art machine learning can be applied. It was demonstrated in Chapter 5 to work with the DVS surprisingly well, outperforming all other dynamic and static models. Importantly for many applications, it also allows merg-

[8] Yann LeCun, Corinna Cortes, and Christopher JC Burges. *The MNIST database of handwritten digits*. 1998

[9] Ken Chatfield, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. "Return of the devil in the details: Delving deep into convolutional nets". In: *arXiv preprint arXiv:1405.3531* (2014)

[10] Jia Deng, Wei Dong, R. Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. "ImageNet: A large-scale hierarchical image database". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 770–778

[12] Bodo Rueckauer, Iulia-Alexandra Lungu, Yuhuang Hu, and Michael Pfeiffer. "Theory and Tools for the Conversion of Analog to Spiking Convolutional Neural Networks". In: *arXiv preprint arXiv:1612.04052* (2016)

[13] Hongjie Liu, Diederik Paul Moeys, Daniel Neil, Shih-Chii Liu, and Tobias Delbruck. "Combined frame- and event-based detection and tracking". In: *IEEE Int. Symposium on Circuits and Systems (ISCAS)*. 2016

[14] D. P. Moeys, F. Corradi, E. Kerr, P. Vance, G. Das, D. Neil, D. Kerr, and T. Delbrück. "Steering a predator robot using a mixed frame/event-driven convolutional neural network". In: *2016 Second International Conference on Event-based Control, Communication, and Signal Processing (EBCCSP)*. June 2016, pp. 1–8. DOI: 10.1109/EBCCSP.2016.7605233

[15] Diederik Paul Moeys, Federico Corradi, Emmett Kerr, Philip Vance, Gautham Das, Daniel Neil, Dermot Kerr, and Tobi Delbrück. "Steering a predator robot using a mixed frame/event-driven convolutional neural network". In: *Event-based Control, Communication, and Signal Processing (EBCCSP), 2016 Second International Conference on*. IEEE. 2016, pp. 1–8

[16] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. "Caffe: Convolutional Architecture for Fast Feature Embedding". In: *arXiv preprint arXiv:1408.5093* (2014)

[17] J. Anumula, D. Neil, X-Y. Li, T. Delbruck, and S-C. Liu. "Live Demonstration: Event-Driven Real-Time Spoken Digit Recognition System". In: *IEEE International Symposium on Circuits and Systems*. May 2017

ing multiply-timed sensors together, performing computation on an input only when that particular sensor has input or when the underlying periodicity encourages a neuron to wake up and pay attention. In unpublished work, it was also used with wind farm data, and offered an unexpected advantage: data which was missing, normally presenting a challenge for machine learning but a frequent occurrence when using real-world datasets, was natively handled. When data was missing due to collection difficulties or sensor failure, the network that relies on this data interprets it like all other points in continuous time that do not receive data and ignores it - which can be difficult to do with standard RNN models that expect a periodic timestep.

However, Phased LSTM was also introduced to tackle a larger question. Time is an incredibly underutilized aspect of information processing, as very few applications make use of continuous time. One initial goal of Phased LSTM was to introduce a model that allowed simpler and easier explorations of continuous-time signals, as Phased LSTM effectively constructs a high-dimensional learned discretization of continuous-time signals. The aim was that the Phased LSTM models would latch onto the principal frequencies that compose a signal, and neurons would emerge that correspond to the rhythms of - for example - phonemes, syllables, words, sentences, and dialogues. Instead, all attempts to understand why Phased LSTM chooses the rhythms it learns have been foiled. The rhythms do appear to be effective at solving the tasks, but they follow no apparent logic. For the frequency discrimination task, they do not choose to learn the frequency or its complement, but rather remain close to their initial values perturbed in small but apparently informative ways. Similarly surprising, nearly identical accuracy results can be achieved with massively different distributions of timing parameters - whether favoring many shorter times, many longer times, or biased mixes of the two. Yet, they do play a critical role: disabling learning or perturbing the timing parameters after training can lead to dramatic losses in accuracy. I would encourage further work to enhance this aspect: a model that learns to decompose the data across time and shows an attentional focus that reflects the expected periodicities in data. Ample data can be found which exhibits patterns at multiple resolutions of time, e.g. solar power output, and an interpretable model should show neuron attentional cycles across hours, days, seasons, and years.

Finally, Chapter 7 outlined an algorithmic optimization for recurrent neural networks which offers 6x-100x speedups on real-world problems by exploiting redundancy in the input representation. While these theoretical speedups are significant, many challenges remain to implement them in practice. Current tensor computation methods suffer significant slowdowns when using sparse tensor operations, and often only begin to be faster than their dense equivalents when approximately 80% sparse. Faster sparse libraries and Just-in-time (JIT) tensor compilation will continue to increase the

[18] Now developed in collaboration with Moritz Milde, publication forthcoming.

speed of sparse software implementations to realize these theoretical gains, but to achieve the maximum speedup a specialized hardware approach should be used. Indeed, this hardware platform is under development and, excitingly, currently demonstrates state-of-the-art performance and efficiency for implementing recurrent neural networks, reinforcing the advantages proposed in this initial work.

8.1 Towards the Future

MUCH REMAINS to be yet accomplished, even in the restricted domain of deep learning for event-based sensors. While the advantages of event-based sensors have been demonstrated on benchmark tasks, the lack of real-world results using event-based sensors remains troubling. However, I am hopeful the future will make better use of this low-power and low-latency event-based sensors, as the next wave of applications seems to require the advantages of these sensors.

Autonomous vehicles will demand substantially *lower-latency sensors* to react quickly to unexpected stimuli. This sort of application, in which a rapid-changing environment needs efficient and accurate analysis, plays directly into the strengths of the sensors and the algorithms that satisfy the goals of this thesis. Moreover, whole-system analysis will become more commonplace in closed-loop systems like autonomous driving; factors like sensor latency, power consumption, and even sensor cost all play a role in determining the efficacy of the platform in ways that do not appear in standard machine learning benchmarks that focus solely on accuracy. Defining the holistic requirements for a sensor platform provides a way for the event-based sensors to better compete with state-of-the-art standard sensors, and the algorithms described in this thesis can help reinforce the usefulness of such systems.

Another area of rapid exploration is an increased focus on online learning. Rising concern about privacy issues is decreasing the availability of the large datasets that power the effectiveness of deep learning. Users have begun to desire increased personalization without submitting their data to companies beyond their control, which is exactly the scenario that *online learning* studies. A significant history of work in online learning in the neuromorphic community [19,20,21,22] should permit event-based methods relevance in the initial foray into online learning using deep neural networks.

Finally, one major purpose of event-based learning is to feed knowledge back into the neuroscience community that inspired these sensors. Inherently, the algorithms that operate on event-based inputs are algorithms that can use spikes; this property should be advantageous when studying biological neural systems which communicate using spikes. It would excite me greatly to see methods like PLSTM applied to neuroscientific targets and any

[19] Daniel Neil. "Online Learning in Event-based Restricted Boltzmann Machines". PhD thesis. Institute of Neuroinformatics, 2013

[20] Emre Neftci, Srinjoy Das, Bruno Pedroni, Kenneth Kreutz-Delgado, and Gert Cauwenberghs. "Event-driven contrastive divergence for spiking neuromorphic systems". In: *Frontiers in Neuroscience* 7.272 (2013)

[21] Enea Ceolini, Daniel Neil, Tobi Delbruck, and Shih-Chii Liu. "Temporal sequence recognition in a self-organizing recurrent network". In: *Event-based Control, Communication, and Signal Processing (EBCCSP), 2016 Second International Conference on*. IEEE. 2016, pp. 1–4

[22] Giacomo Indiveri, Elisabetta Chicca, and Rodney J Douglas. "Artificial cognitive systems: From VLSI networks of spiking neurons to neuromorphic cognition". In: *Cognitive Computation* 1.2 (2009), pp. 119–127

insights that such an approach could yield.

Beyond event-based sensors, other approaches also attempt to break free from the standard input and computation paradigms of frame-based computation to offer significant advances for the state-of-the-art. Recently, work has shown that backpropagation can be used in conjunction with spiking networks to achieve some of the highest benchmarked accuracy levels [23,24]. Other research groups have begun investigating how both the forward and backward (backpropagated) training pass could be implemented in a spiking way [25], arising from earlier work showing that mathematically precise backpropagation is not needed to train effectively [26]. Perhaps this hints at how biological implementations can train so effectively. Moreover, incorporating the non-idealities of implementations into training [27] shows that the extreme noise of low-power implementations can indeed be countered, and that deep network computation can be implemented on an even wider range of platforms than previously imagined.

Overall, returning to the question that motivated this thesis, it does satisfyingly appear that substantial improvements can be made over the state-of-the-art by freeing implementations from the constraints implicitly chosen by frame-based data paradigms. When using event-driven sensors and forced to consider aspects of the real-world that are typically abstracted away, aspects such as continuous time and gradual sensing, exciting new opportunities emerge to improve existing algorithms. This thesis presents many new algorithms that adapt ideas from the state-of-the-art in deep learning that yet allow the advantages of event-driven sensors to be maintained, yielding models that benefit both scientific fields.

[23] Peter O'Connor and Max Welling. "Deep Spiking Networks". In: *arXiv preprint arXiv:1602.08323* (2016)

[24] Jun Haeng Lee, Tobi Delbruck, and Michael Pfeiffer. "Training deep spiking neural networks using backpropagation". In: *Frontiers in Neuroscience* 10 (2016)

[25] Arash Samadi, Timothy P Lillicrap, and Douglas B Tweed. "Deep Learning with Dynamic Spiking Neurons and Fixed Feedback Weights". In: *Neural Computation* (2017)

[26] Timothy P Lillicrap, Daniel Cownden, Douglas B Tweed, and Colin J Akerman. "Random feedback weights support learning in deep neural networks". In: *arXiv preprint arXiv:1411.0247* (2014)

[27] Jonathan Binas, Daniel Neil, Giacomo Indiveri, Shih-Chii Liu, and Michael Pfeiffer. "Precise deep neural network computation on imprecise low-power analog hardware". In: *arXiv preprint arXiv:1606.07786* (2016)

Author's Cited Works

Anumula, J., D. Neil, X-Y. Li, T. Delbruck, and S-C. Liu. "Live Demonstration: Event-Driven Real-Time Spoken Digit Recognition System". In: *IEEE International Symposium on Circuits and Systems*. May 2017.

Binas, Jonathan, Daniel Neil, Giacomo Indiveri, Shih-Chii Liu, and Michael Pfeiffer. "Precise deep neural network computation on imprecise low-power analog hardware". In: *arXiv preprint arXiv:1606.07786* (2016).

Braun, Stefan, Daniel Neil, and Shih-Chii Liu. "A Curriculum Learning Method for Improved Noise Robustness in Automatic Speech Recognition". In: *arXiv preprint arXiv:1606.06864* (2016).

Ceolini, Enea, Daniel Neil, Tobi Delbruck, and Shih-Chii Liu. "Temporal sequence recognition in a self-organizing recurrent network". In: *Event-based Control, Communication, and Signal Processing (EBCCSP), 2016 Second International Conference on*. IEEE. 2016, pp. 1–4.

Diehl, Peter U, Daniel Neil, Jonathan Binas, Matthew Cook, Shih-Chii Liu, and Michael Pfeiffer. "Fast-Classifying, High-Accuracy Spiking Deep Networks Through Weight and Threshold Balancing". In: *International Joint Conference on Neural Networks (IJCNN)*. 2015.

Kiselev, Ilya, Daniel Neil, and Shih-Chii Liu. "Event-Driven Deep Neural Network Hardware System for Sensor Fusion". In: *IEEE Int. Symposium on Circuits and Systems (ISCAS)*. 2016.

– "Live demonstration: Event-Driven Deep Neural Network Hardware System for Sensor Fusion". In: *IEEE Int. Symposium on Circuits and Systems (ISCAS)*. 2016.

Liu, Hongjie, Diederik Paul Moeys, Daniel Neil, Shih-Chii Liu, and Tobias Delbruck. "Combined frame- and event-based detection and tracking". In: *IEEE Int. Symposium on Circuits and Systems (ISCAS)*. 2016.

Moeys, D. P., F. Corradi, E. Kerr, P. Vance, G. Das, D. Neil, D. Kerr, and T. Delbrück. "Steering a predator robot using a mixed frame/event-driven convolutional neural network". In: *2016 Second International Conference on Event-based Control, Communication, and Signal Processing (EBCCSP)*. June 2016, pp. 1–8. DOI: 10.1109/EBCCSP.2016.7605233.

Moeys, Diederik Paul, Federico Corradi, Emmett Kerr, Philip Vance, Gautham Das, Daniel Neil, Dermot Kerr, and Tobi Delbrück. "Steering a predator robot using a mixed frame/event-driven convolutional neural network". In: *Event-based Control, Communication, and Signal Processing (EBCCSP), 2016 Second International Conference on*. IEEE. 2016, pp. 1–8.

Neil, Daniel. "Online Learning in Event-based Restricted Boltzmann Machines". PhD thesis. Institute of Neuroinformatics, 2013.

Neil, Daniel, Tobi Delbruck, and Shih-Chii Liu. "Event-Driven Deep Multi-Layered Network Architectures". In: IEEE, 2017.

Neil, Daniel, Jun Haeng Lee, Tobi Delbruck, and Shih-Chii Liu. "Delta Networks for Optimized Recurrent Network Computation". In: *arXiv preprint arXiv:1612.05571* (2016).

Neil, Daniel and S-C Liu. "Minitaur, an event-driven FPGA-based spiking network accelerator". In: *IEEE Trans on Very Large Scale Integration (VLSI) Systems* 22.12 (2014), pp. 2621–2628.

Neil, Daniel and Shih-Chii Liu. "Effective Sensor Fusion with Event-Based Sensors and Deep Network Architectures". In: *IEEE Int. Symposium on Circuits and Systems (ISCAS)*. 2016, pp. 2282–2285.

– "Expanded Working Memory Enhances Phased LSTM". 2017.

Neil, Daniel, Michael Pfeiffer, and Shih-Chii Liu. "Learning to be Efficient: Algorithms for Training Low-Latency, Low-Compute Deep Spiking Neural Networks". In: *ACM Symposium on Applied Computing*. Vol. 31. 2016.

– "Phased LSTM: Accelerating Recurrent Network Training for Long or Event-based Sequences". In: *Advances in Neural Information Processing Systems*. 2016, pp. 3882–3890.

O'Connor, Peter, Daniel Neil, Shih-Chii Liu, Tobi Delbruck, and Michael Pfeiffer. "Real-time classification and sensor fusion with a spiking Deep Belief Network". In: *Frontiers in Neuroscience* 7 (2013).

Stromatias, Evangelos, Daniel Neil, Francesco Galluppi, Michael Pfeiffer, Shih-Chii Liu, and Steve Furber. "Event-Driven Deep Neural Network Hardware System for Sensor Fusion". In: *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*. 2015, pp. 1901–1901.

– "Scalable Energy-Efficient, Low-Latency Implementations of Spiking Deep Belief Networks on SpiNNaker". In: *Proceedings of the 2015 IEEE International Joint Conference on Neural Networks (IJCNN)*. 2015, pp. 1–8. DOI: 10.1109/IJCNN.2015.7280625.

Stromatias, Evangelos, Daniel Neil, Michael Pfeiffer, Francesco Galluppi, Steve B Furber, and Shih-Chii Liu. "Robustness of spiking Deep Belief Networks to noise and reduced bit precision of neuro-inspired hardware platforms". In: *Frontiers in Neuroscience* 9 (2015).

Full Bibliography

Agis, R., E. Ros, J. Diaz, R. Carrillo, and E. M. Ortigosa. "Hardware event-driven simulation engine for spiking neural networks". In: *International Journal of Electronics* 94.5 (2007), pp. 469–480.

Anumula, J., D. Neil, X-Y. Li, T. Delbruck, and S-C. Liu. "Live Demonstration: Event-Driven Real-Time Spoken Digit Recognition System". In: *IEEE International Symposium on Circuits and Systems*. May 2017.

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate". In: *arXiv preprint arXiv:1409.0473* (2014).

Benjamin, Ben Varkey, Peiran Gao, Emmett McQuinn, Swadesh Choudhary, Anand R Chandrasekaran, J Bussat, Rodrigo Alvarez-Icaza, John V Arthur, PA Merolla, and Kwabena Boahen. "Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations". In: *Proceedings of the IEEE* 102.5 (2014), pp. 699–716.

Bergstra, James, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. "Theano: a CPU and GPU math expression compiler". In: *Proceedings of the Python for scientific computing conference (SciPy)*. Vol. 4. 2010, p. 3.

Berner, R., C. Brandli, M. Yang, S. C. Liu, and T. Delbruck. "A 240 $\times$ 180 10mW 12μs latency sparse-output vision sensor for mobile applications". In: *2013 Symposium on VLSI Circuits*. June 2013, pp. C186–C187.

Binas, Jonathan, Daniel Neil, Giacomo Indiveri, Shih-Chii Liu, and Michael Pfeiffer. "Precise deep neural network computation on imprecise low-power analog hardware". In: *arXiv preprint arXiv:1606.07786* (2016).

Bojarski, Mariusz, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Ziebam. "End to End Learning for Self-Driving Cars". In: *arXiv preprint arXiv:1604.07316* (2016).

Braun, Stefan, Daniel Neil, and Shih-Chii Liu. "A Curriculum Learning Method for Improved Noise Robustness in Automatic Speech Recognition". In: *arXiv preprint arXiv:1606.06864* (2016).

Brette, Romain, Michelle Rudolph, Ted Carnevale, Michael Hines, David Beeman, James M Bower, Markus Diesmann, Abigail Morrison, Philip H Goodman, Frederick C Harris Jr, et al. "Simulation of networks of spiking neurons: a review of tools and strategies". In: *Journal of Computational Neuroscience* 23.3 (2007), pp. 349–398.

Brunel, N. and M. C. W. van Rossum. "Lapicqueś 1907 paper: from frogs to integrate-and-fire". In: *Biological Cybernetics* 97.5 (2007), pp. 337–339.

Buzsaki, György. *Rhythms of the Brain*. Oxford University Press, 2006.

Camunas-Mesa, Luis, Carlos Zamarreno-Ramos, Alejandro Linares-Barranco, Antonio J Acosta-Jimenez, Teresa Serrano-Gotarredona, and Bernabé Linares-Barranco. "An event-driven multi-kernel convolution processor module for event-driven vision sensors". In: *IEEE Journal of Solid-State Circuits* 47.2 (2012), pp. 504–517.

Cao, Yongqiang, Yang Chen, and Deepak Khosla. "Spiking Deep Convolutional Neural Networks for Energy-Efficient Object Recognition". In: *International Journal of Computer Vision* (2014), pp. 1–13.

Cassidy, A., A.G. Andreou, and J. Georgiou. "Design of a one million neuron single FPGA neuromorphic system for real-time multimodal scene analysis". In: *45th Annual Conference on Information Sciences and Systems (CISS)*. 2011, pp. 1–6.

Cauwenberghs, Gert. "An analog VLSI recurrent neural network learning a continuous-time trajectory". In: *IEEE Transactions on Neural Networks* 7.2 (1996), pp. 346–361.

Ceolini, Enea, Daniel Neil, Tobi Delbruck, and Shih-Chii Liu. "Temporal sequence recognition in a self-organizing recurrent network". In: *Event-based Control, Communication, and Signal Processing (EBCCSP), 2016 Second International Conference on*. IEEE. 2016, pp. 1–4.

Chan, V., S. C. Liu, and A. van Schaik. "AER EAR: A Matched Silicon Cochlea Pair With Address Event Representation Interface". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 54.1 (Jan. 2007), pp. 48–59. ISSN: 1549-8328. DOI: 10.1109/TCSI.2006.887979.

Chatfield, Ken, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. "Return of the devil in the details: Delving deep into convolutional nets". In: *arXiv preprint arXiv:1405.3531* (2014).

Chen, Yu-Hsin, Tushar Krishna, Joel S Emer, and Vivienne Sze. "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks". In: *IEEE Journal of Solid-State Circuits* 52.1 (2017), pp. 127–138.

Cheung, K., S. R. Schultz, and W. Luk. "A large-scale spiking neural network accelerator for FPGA systems". In: *International Conference Artificial Neural Networks and Machine Learning (ICANN 2012)*. Vol. 7552. Springer, 2012, pp. 113–120.

Cheung, K., S.R. Schultz, and P.H.W. Leong. "A parallel spiking neural network simulator". In: *International Conference on Field-Programmable Technology (FPT 2009)*. 2009, pp. 247–254.

Cho, Kyunghyun, Aaron Courville, and Yoshua Bengio. "Describing multimedia content using attention-based encoder-decoder networks". In: *IEEE Transactions on Multimedia* 17.11 (2015), pp. 1875–1886.

Cho, Kyunghyun, Bart van Merrienboer, Caglar Gulcehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. "Learning phrase representations using RNN encoder-decoder for statistical machine translation". In: *arXiv preprint arXiv:1406.1078* (2014).

Chung, Junyoung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. "Empirical evaluation of gated recurrent neural networks on sequence modeling". In: *arXiv preprint arXiv:1412.3555* (2014).

Ciresan, Dan Claudiu, Ueli Meier, Luca Maria Gambardella, and Jürgen Schmidhuber. "Deep, big, simple neural nets for handwritten digit recognition". In: *Neural Computation* 22.12 (2010), pp. 3207–3220.

Cohen, Gregory Kevin, Garrick Orchard, Sio Hoi Ieng, Jonathan Tapson, Ryad Benjamin Benosman, and André van Schaik. "Skimming Digits: Neuromorphic Classification of Spike-Encoded Images". In: *Frontiers in Neuroscience* 10.184 (2016). DOI: 10.3389/fnins.2016.00184.

Cooke, Martin, Jon Barker, Stuart Cunningham, and Xu Shao. "An audio-visual corpus for speech perception and automatic speech recognition". In: *The Journal of the Acoustical Society of America* 120.5 (2006), pp. 2421–2424.

Courbariaux, Matthieu and Yoshua Bengio. "Binarynet: Training deep neural networks with weights and activations constrained to+ 1 or-1". In: *arXiv preprint arXiv:1602.02830* (2016).

Courbariaux, Matthieu, Yoshua Bengio, and Jean-Pierre David. "Binaryconnect: Training deep neural networks with binary weights during propagations". In: *Advances in Neural Information Processing Systems*. 2015, pp. 3123–3131.

– "Low precision arithmetic for deep learning". In: *arXiv preprint arXiv:1412.7024* (2014).

Delorme, A. and S.J. Thorpe. "SpikeNET: an event-driven simulation package for modelling large networks of spiking neurons". In: *Network: Computation in Neural Systems* 14.4 (2003), pp. 613–627.

Deng, Jia, Wei Dong, R. Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. "ImageNet: A large-scale hierarchical image database". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

Diehl, Peter U and Matthew Cook. "Efficient implementation of STDP rules on SpiNNaker neuromorphic hardware". In: *2014 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2014, pp. 4288–4295.

Diehl, Peter U, Daniel Neil, Jonathan Binas, Matthew Cook, Shih-Chii Liu, and Michael Pfeiffer. "Fast-Classifying, High-Accuracy Spiking Deep Networks Through Weight and Threshold Balancing". In: *International Joint Conference on Neural Networks (IJCNN)*. 2015.

Dieleman, Sander et al. *Lasagne: First release.* Aug. 2015. DOI: 10.5281/zenodo.27878. URL: http://dx.doi.org/10.5281/zenodo.27878.

Esser, Steven K, Paul A Merolla, John V Arthur, Andrew S Cassidy, Rathinakumar Appuswamy, Alexander Andreopoulos, David J Berg, Jeffrey L McKinstry, Timothy Melano, Davis R Barch, et al. "Convolutional Networks for Fast, Energy-Efficient Neuromorphic Computing". In: *Proceedings of the National Academy of Sciences* 113.41 (2016), pp. 11441–11446.

Farabet, Clément et al. "Comparison between frame-constrained fix-pixel-value and frame-free spiking-dynamic-pixel convNets for visual processing". In: *Frontiers in Neuroscience* 6 (2012).

Farabet, Clement, Camille Couprie, Laurent Najman, and Yann LeCun. "Learning hierarchical features for scene labeling". In: *IEEE Trans on Pattern Analysis and Machine Intelligence* 35.8 (2013), pp. 1915–1929.

Fukushima, Kunihiko. "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position". In: *Biological Cybernetics* 36.4 (1980), pp. 193–202.

Funahashi, Ken-Ichi and Yuichi Nakamura. "Approximation of dynamical systems by continuous time recurrent neural networks". In: *Neural Networks* 6.6 (1993), pp. 801–806.

Furber, S.B., F. Galluppi, S. Temple, and L.A Plana. "The SpiNNaker Project". In: *Proceedings of the IEEE* 102.5 (May 2014), pp. 652–665. ISSN: 0018-9219. DOI: 10.1109/JPROC.2014.2304638.

Garbin, D., O. Bichler, E. Vianello, Q. Rafhay, C. Gamrat, L. Perniola, G. Ghibaudo, and B. DeSalvo. "Variability-tolerant Convolutional Neural Network for Pattern Recognition Applications based on OxRAM Synapses". In: *IEEE International Electron Devices Meeting (IEDM)* (2014), pp. 1–13.

Gers, Felix A and Jürgen Schmidhuber. "Recurrent nets that time and count". In: *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks (IJCNN)*. Vol. 3. IEEE. 2000, pp. 189–194.

Goodfellow, Ian J., David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. "Maxout Networks". In: *ICML*. 2013.

Graves, Alex. "Generating sequences with recurrent neural networks". In: *arXiv preprint arXiv:1308.0850* (2013).

Graves, Alex, Abdel-Rahman Mohamed, and Geoffrey Hinton. "Speech recognition with deep recurrent neural networks". In: *2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2013, pp. 6645–6649.

Graves, Alex, Greg Wayne, and Ivo Danihelka. "Neural Turing Machines". In: *arXiv preprint arXiv:1410.5401* (2014).

Graves, Alex, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. "Hybrid computing using a neural network with dynamic external memory". In: *Nature* 538.7626 (2016), pp. 471–476.

Han, Song, Huizi Mao, and William J Dally. "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding". In: *CoRR, abs/1510.00149* 2 (2015).

Han, Song, Junlong Kang, Huizi Mao, Yiming Hu, Xin Li, Yubin Li, Dongliang Xie, Hong Luo, Song Yao, Yu Wang, et al. "ESE: Efficient Speech Recognition Engine with Compressed LSTM on FPGA". In: *FPGA 2017; NIPS 2016 EMDNN workshop*. 2016.

Hannun, Awni, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, et al. "Deep speech: Scaling up end-to-end speech recognition". In: *arXiv preprint arXiv:1412.5567* (2014).

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 770–778.

– "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". In: *The IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 1026–1034.

Hempel, Martin. "Deep Learning for Piloted Driving". In: *NVIDIA GPU Tech Conference*. 2016.

Hinton, G.E. and R.R. Salakhutdinov. "Reducing the dimensionality of data with neural networks". In: *Science* 313.5786 (2006), pp. 504–507.

Hinton, Geoffrey E, Simon Osindero, and Yee-Whye Teh. "A fast learning algorithm for deep belief nets". In: *Neural computation* 18.7 (2006), pp. 1527–1554.

Hinton, Geoffrey E, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. "Improving neural networks by preventing co-adaptation of feature detectors". In: *arXiv preprint arXiv:1207.0580* (2012).

Hochreiter, Sepp and Jürgen Schmidhuber. "Long short-term memory". In: *Neural Computation* 9.8 (1997), pp. 1735–1780.

Horowitz, M. "1.1 Computing's energy problem (and what we can do about it)". In: *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*. Feb. 2014, pp. 10–14. DOI: 10.1109/ISSCC.2014.6757323.

Huang, Gao, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. "Deep networks with stochastic depth". In: *European Conference on Computer Vision*. Springer. 2016, pp. 646–661.

Hutter, Marcus. "The human knowledge compression contest". In: *http://prize.hutter1.net* (2012).

Iandola, Forrest N, Matthew W Moskewicz, Khalid Ashraf, Song Han, William J Dally, and Kurt Keutzer. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 1MB model size". In: *arXiv preprint arXiv:1602.07360* (2016).

Indiveri, G., E. Chicca, and R. Douglas. "A VLSI figurable network of integrate-and-fire neurons with spike-based learning synapses". In: (2004).

Indiveri, Giacomo, Elisabetta Chicca, and Rodney J Douglas. "Artificial cognitive systems: From VLSI networks of spiking neurons to neuromorphic cognition". In: *Cognitive Computation* 1.2 (2009), pp. 119–127.

Indiveri, Giacomo, Federico Corradi, and Ning Qiao. "Neuromorphic architectures for spiking deep neural networks". In: *2015 IEEE International Electron Devices Meeting (IEDM)*. 2015, pp. 2–4.

Itseez. *Open Source Computer Vision Library*. https://github.com/itseez/opencv. 2015.

Izhikevich, E.M. "Simple Model of Spiking Neurons". In: *IEEE Transactions on Neural Networks* 14 (2003), pp. 1569–1572.

Jia, Yangqing, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. "Caffe: Convolutional Architecture for Fast Feature Embedding". In: *arXiv preprint arXiv:1408.5093* (2014).

Jin, X., A. Rast, F. Galluppi, S. Davies, and S.B. Furber. "Implementing spike-timing-dependent plasticity on SpiNNaker neuromorphic hardware". In: *Neural Networks (IJCNN), The 2010 International Joint Conference on*. IEEE. 2010, pp. 1–8.

Johnson, Justin, Andrej Karpathy, and Li Fei-Fei. "Densecap: Fully convolutional localization networks for dense captioning". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 4565–4574.

Jug, F., M. Cook, and A. Steger. "Recurrent competitive networks can learn locally excitatory topologies". In: *Proceedings of 2012 International Joint Conference on Neural Networks (IJCNN)*. June 2012, pp. 1–8. DOI: 10.1109/IJCNN.2012.6252786.

Kingma, Diederik and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

Kiselev, Ilya, Daniel Neil, and Shih-Chii Liu. "Event-Driven Deep Neural Network Hardware System for Sensor Fusion". In: *IEEE Int. Symposium on Circuits and Systems (ISCAS)*. 2016.

– "Live demonstration: Event-Driven Deep Neural Network Hardware System for Sensor Fusion". In: *IEEE Int. Symposium on Circuits and Systems (ISCAS)*. 2016.

Koutnik, Jan, Klaus Greff, Faustino Gomez, and Juergen Schmidhuber. "A clockwork RNN". In: *arXiv preprint arXiv:1402.3511* (2014).

Krizhevsky, Alex and Geoffrey Hinton. "Learning multiple layers of features from tiny images". In: (2009).

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Proc. of NIPS*. 2012, pp. 1097–1105.

Lang, K. "Newsweeder: Learning to filter netnews". In: *Proceedings of the Twelfth International Conference on Machine Learning*. 1995, pp. 331–339.

Laurent, Thomas and James von Brecht. "A recurrent neural network without chaos". In: *arXiv preprint arXiv:1612.06212* (2016).

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *Nature* 521.7553 (2015), pp. 436–444.

LeCun, Yann, Léon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

LeCun, Yann, Corinna Cortes, and Christopher JC Burges. *The MNIST database of handwritten digits*. 1998.

Lee, Daniel D and H Sebastian Seung. "Learning the parts of objects by non-negative matrix factorization". In: *Nature* 401.6755 (1999), pp. 788–791.

Lee, Jun Haeng, Tobi Delbruck, and Michael Pfeiffer. "Training deep spiking neural networks using backpropagation". In: *Frontiers in Neuroscience* 10 (2016).

Leung, B., Y. Pan, C. Schroeder, S. O. Memik, G.n Memik, and M. Hartmann. "Towards an 'early neural circuit simulator': A FPGA implementation of processing in the rat whisker system". In: *International Conference on Field Programmable Logic and Applications (FPL 2008)*. 2008, pp. 191–196.

Lichtsteiner, Patrick, Christoph Posch, and Tobi Delbruck. "A 128× 128 120 dB 15 μs latency asynchronous temporal contrast vision sensor". In: *IEEE Journal of Solid-State Circuits* 43.2 (2008), pp. 566–576.

Lillicrap, Timothy P, Daniel Cownden, Douglas B Tweed, and Colin J Akerman. "Random feedback weights support learning in deep neural networks". In: *arXiv preprint arXiv:1411.0247* (2014).

Liu, Hongjie, Diederik Paul Moeys, Daniel Neil, Shih-Chii Liu, and Tobias Delbruck. "Combined frame- and event-based detection and tracking". In: *IEEE Int. Symposium on Circuits and Systems (ISCAS)*. 2016.

Liu, S-C., A. van Schaik, B. Minch, and T. Delbrück. "Asynchronous Binaural Spatial Audition Sensor with 2 × 64 × 4 Channel Output". In: *IEEE Trans. Biomed. Circuits Syst.* 8.4 (2014), pp. 453–464. DOI: 10.1109/TBCAS.2013.2281834.

Liu, Shih-Chii and Tobi Delbruck. "Neuromorphic sensory systems". In: *Current Opinion in Neurobiology* 20.3 (2010), pp. 288–295.

Lobb, C.J., Z. Chao, R.M. Fujimoto, and S.M. Potter. "Parallel event-driven neural network simulations using the Hodgkin-Huxley neuron model". In: *Workshop on Principles of Advanced and Distributed Simulation (PADS) 2005*. 2005, pp. 16–25.

Maass, Wolfgang and Henry Markram. "On the computational power of circuits of spiking neurons". In: *Journal of Computer and System Sciences* 69.4 (2004), pp. 593–616.

Maguire, L.P., T.M. McGinnity, B. Glackin, A. Ghani, A. Belatreche, and J. Harkin. "Challenges for large-scale implementations of spiking neural networks on FPGAs". In: *Neurocomputing* 71.1 (2007), pp. 13–29.

Marian, I., R. Reilly, and D. Mackey. "Efficient event-driven simulation of spiking neural networks". In: *Proceedings of 3rd WSES International Conference on: Neural Networks and Applications*. 2002.

Merolla, Paul A, John V Arthur, Rodrigo Alvarez-Icaza, Andrew S Cassidy, Jun Sawada, Filipp Akopyan, Bryan L Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, et al. "A million spiking-neuron integrated circuit with a scalable communication network and interface". In: *Science* 345.6197 (2014), pp. 668–673.

Merolla, Paul, John Arthur, Filipp Akopyan, Nabil Imam, Rajit Manohar, and Dharmendra S Modha. "A digital neurosynaptic core using embedded crossbar memory with 45pJ per spike in 45nm". In: *2011 IEEE Custom Integrated Circuits Conference (CICC)*. 2011, pp. 1–4.

Mikolov, Tomas, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. "Recurrent neural network based language model." In: *Interspeech* 2 (2010), p. 3.

Misra, J. and I. Saha. "Artificial neural networks in hardware: A survey of two decades of progress". In: *Neurocomputing* 74.1 (2010), pp. 239–255.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (2015), pp. 529–533.

Moeys, D. P., F. Corradi, E. Kerr, P. Vance, G. Das, D. Neil, D. Kerr, and T. Delbrück. "Steering a predator robot using a mixed frame/event-driven convolutional neural network". In: *2016 Second International Conference on Event-based Control, Communication, and Signal Processing (EBCCSP)*. June 2016, pp. 1–8. DOI: 10.1109/EBCCSP.2016.7605233.

Moeys, Diederik Paul, Federico Corradi, Emmett Kerr, Philip Vance, Gautham Das, Daniel Neil, Dermot Kerr, and Tobi Delbrück. "Steering a predator robot using a mixed frame/event-driven convolutional neural network". In: *Event-based Control, Communication, and Signal Processing (EBCCSP), 2016 Second International Conference on*. IEEE. 2016, pp. 1–8.

Mohamed, A., G. E. Dahl, and G. Hinton. "Acoustic modeling using deep belief networks". In: *IEEE Transactions on Audio, Speech, and Language Processing* 20.1 (2012), pp. 14–22.

Neftci, Emre, Srinjoy Das, Bruno Pedroni, Kenneth Kreutz-Delgado, and Gert Cauwenberghs. "Event-driven contrastive divergence for spiking neuromorphic systems". In: *Frontiers in Neuroscience* 7.272 (2013).

Neil, Daniel. "Online Learning in Event-based Restricted Boltzmann Machines". PhD thesis. Institute of Neuroinformatics, 2013.

Neil, Daniel, Tobi Delbruck, and Shih-Chii Liu. "Event-Driven Deep Multi-Layered Network Architectures". In: IEEE, 2017.

Neil, Daniel, Jun Haeng Lee, Tobi Delbruck, and Shih-Chii Liu. "Delta Networks for Optimized Recurrent Network Computation". In: *arXiv preprint arXiv:1612.05571* (2016).

Neil, Daniel and S-C Liu. "Minitaur, an event-driven FPGA-based spiking network accelerator". In: *IEEE Trans on Very Large Scale Integration (VLSI) Systems* 22.12 (2014), pp. 2621–2628.

Neil, Daniel and Shih-Chii Liu. "Effective Sensor Fusion with Event-Based Sensors and Deep Network Architectures". In: *IEEE Int. Symposium on Circuits and Systems (ISCAS)*. 2016, pp. 2282–2285.

– "Expanded Working Memory Enhances Phased LSTM". 2017.

Neil, Daniel, Michael Pfeiffer, and Shih-Chii Liu. "Learning to be Efficient: Algorithms for Training Low-Latency, Low-Compute Deep Spiking Neural Networks". In: *ACM Symposium on Applied Computing*. Vol. 31. 2016.

– "Phased LSTM: Accelerating Recurrent Network Training for Long or Event-based Sequences". In: *Advances in Neural Information Processing Systems*. 2016, pp. 3882–3890.

Nessler, Bernhard, Michael Pfeiffer, Lars Buesing, and Wolfgang Maass. "Bayesian computation emerges in generic cortical microcircuits through spike-timing-dependent plasticity". In: *PLoS Comput Biol* 9.4 (2013), e1003037.

O'Connor, Peter, Daniel Neil, Shih-Chii Liu, Tobi Delbruck, and Michael Pfeiffer. "Real-time classification and sensor fusion with a spiking Deep Belief Network". In: *Frontiers in Neuroscience* 7 (2013).

O'Connor, Peter and Max Welling. "Deep Spiking Networks". In: *arXiv preprint arXiv:1602.08323* (2016).

– "Sigma Delta Quantized Networks". In: *arXiv preprint arXiv:1611.02024* (2016).

Oord, Aaron van den, Nal Kalchbrenner, and Koray Kavukcuoglu. "Pixel recurrent neural networks". In: *arXiv preprint arXiv:1601.06759* (2016).

Oord, Aäron van den, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. "Wavenet: A generative model for raw audio". In: *CoRR abs/1609.03499* (2016).

Orchard, G., A. Jayawant, G. Cohen, and N. Thakor. "Converting static image datasets to spiking neuromorphic datasets using saccades". In: *Frontiers in Neuroscience* 9 (2015), p. 437. DOI: 10.3389/fnins.2015.00437. URL: http://journal.frontiersin.org/article/10.3389/fnins.2015.00437.

Ott, Joachim, Zhouhan Lin, Ying Zhang, Shih-Chii Liu, and Yoshua Bengio. "Recurrent Neural Networks With Limited Numerical Precision". In: *arXiv preprint arXiv:1608.06902* (2016).

Palm, R. B. "Prediction as a candidate for learning deep hierarchical models of data". MA thesis. 2012.

Pearlmutter, Barak A. "Learning state space trajectories in recurrent neural networks". In: *Neural Computation* 1.2 (1989), pp. 263–269.

Pérez-Carrasco, Jose and others. "Mapping from Frame-Driven to Frame-Free Event-Driven Vision Systems by Low-Rate Rate Coding and Coincidence Processing–Application to Feedforward ConvNets". In: *IEEE Trans on Pattern Analysis and Machine Intelligence* 35.11 (2013), pp. 2706–2719.

Posch, C., D. Matolin, and R. Wohlgenannt. "A QVGA 143 dB Dynamic Range Frame-Free PWM Image Sensor With Lossless Pixel-Level Video Compression and Time-Domain CDS". In: *IEEE Journal of Solid-State Circuits* 46.1 (Jan. 2011), pp. 259–275. ISSN: 0018-9200.

Posch, C., T. Serrano-Gotarredona, B. Linares-Barranco, and T. Delbruck. "Retinomorphic Event-Based Vision Sensors: Bioinspired Cameras With Spiking Output". In: *Proceedings of the IEEE* 102.10 (Oct. 2014), pp. 1470–1484. ISSN: 0018-9219. DOI: 10.1109/JPROC.2014.2346153.

Posch, Christoph, Teresa Serrano-Gotarredona, Bernabe Linares-Barranco, and Tobi Delbruck. "Retinomorphic event-based vision sensors: bioinspired cameras with spiking outputs". In: *Proc. of the IEEE* 102.10 (2014), pp. 1470–1484.

Poultney, Christopher, Sumit Chopra, Yann L Cun, et al. "Efficient learning of sparse representations with an energy-based model". In: *Advances in neural information processing systems*. 2006, pp. 1137–1144.

Povey, Daniel, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian, Petr Schwarz, et al. "The Kaldi speech recognition toolkit". In: *IEEE 2011 workshop on automatic speech recognition and understanding*. EPFL-CONF-192584. IEEE Signal Processing Society. 2011.

Rastegari, Mohammad, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. "Xnor-net: Imagenet classification using binary convolutional neural networks". In: *European Conference on Computer Vision*. Springer, 2016, pp. 525–542.

Rueckauer, Bodo, Iulia-Alexandra Lungu, Yuhuang Hu, and Michael Pfeiffer. "Theory and Tools for the Conversion of Analog to Spiking Convolutional Neural Networks". In: *arXiv preprint arXiv:1612.04052* (2016).

Samadi, Arash, Timothy P Lillicrap, and Douglas B Tweed. "Deep Learning with Dynamic Spiking Neurons and Fixed Feedback Weights". In: *Neural Computation* (2017).

Santana, Eder and George Hotz. "Learning a Driving Simulator". In: *arXiv preprint arXiv:1608.01230* (2016).

Schmidhuber, Jürgen. "Deep learning in neural networks: An overview". In: *Neural Networks* 61 (2015), pp. 85–117.

Schoenauer, T., N. Mehrtash, Andreas Jahnke, and H. Klar. "MASPINN: novel concepts for a neuroaccelerator for spiking neural networks". In: (1999), pp. 87–96. DOI: 10.1117/12.343072. URL: +%20http://dx.doi.org/10.1117/12.343072.

Seide, F., G. Li, and D. Yu. "Conversational Speech Transcription Using Context-Dependent Deep Neural Networks". In: *Interspeech*. 2011, pp. 437–440.

Semeniuta, Stanislau, Aliaksei Severyn, and Erhardt Barth. "Recurrent Dropout without Memory Loss". In: *arXiv* arXiv:1603.05118 (2016).

Sermanet, Pierre, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. "OverFeat: Integrated recognition, localization and detection using convolutional networks". In: *arXiv preprint* 312.6229 (2013).

Serrano-Gotarredona, R. and others. "CAVIAR: A 45k Neuron, 5M Synapse, 12G Connects/s AER Hardware Sensory–Processing– Learning–Actuating System for High-Speed Visual Object Recognition and Tracking". In: *IEEE Trans on Neural Networks* 20.9 (2009), pp. 1417–1438.

Serrano-Gotarredona, T. and B. Linares-Barranco. "A 128 × 128 1.5% Contrast Sensitivity 0.9% FPN 3 μs Latency 4 mW Asynchronous Frame-Free Dynamic Vision Sensor Using Transimpedance Preamplifiers". In: *IEEE Journal of Solid-State Circuits* 48.3 (Mar. 2013), pp. 827–838. ISSN: 0018-9200. DOI: 10.1109/JSSC.2012.2230553.

Seung, H Sebastian and Daniel D Lee. "The manifold ways of perception". In: *science* 290.5500 (2000), pp. 2268–2269.

Siegert, A. J. F. "On the first passage time probability problem". In: *Physical Review* 81.4 (1951), p. 617.

Silver, David, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529.7587 (2016), pp. 484–489.

Srivastava, Nitish, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: A simple way to prevent neural networks from overfitting". In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

Stromatias, E., F. Galluppi, C. Patterson, and S. Furber. "Power analysis of large-scale, real-time neural networks on SpiNNaker". In: *Proceedings of 2013 International Joint Conference on Neural Networks (IJCNN)*. Aug. 2013, pp. 1–8. DOI: 10.1109/IJCNN.2013.6706927.

Stromatias, Evangelos, Daniel Neil, Francesco Galluppi, Michael Pfeiffer, Shih-Chii Liu, and Steve Furber. "Event-Driven Deep Neural Network Hardware System for Sensor Fusion". In: *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*. 2015, pp. 1901–1901.

– "Scalable Energy-Efficient, Low-Latency Implementations of Spiking Deep Belief Networks on SpiNNaker". In: *Proceedings of the 2015 IEEE International Joint Conference on Neural Networks (IJCNN)*. 2015, pp. 1–8. DOI: 10.1109/IJCNN.2015.7280625.

Stromatias, Evangelos, Daniel Neil, Michael Pfeiffer, Francesco Galluppi, Steve B Furber, and Shih-Chii Liu. "Robustness of spiking Deep Belief Networks to noise and reduced bit precision of neuro-inspired hardware platforms". In: *Frontiers in Neuroscience* 9 (2015).

Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going deeper with convolutions". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 1–9.

Thomas, D.B. and W. Luk. "FPGA accelerated simulation of biologically plausible spiking neural networks". In: *17th IEEE Symposium on Field Programmable Custom Computing Machines (FCCM '09)*. 2009, pp. 45–52.

Wan, Li, Matthew Zeiler, Sixin Zhang, Yann L Cun, and Rob Fergus. "Regularization of neural networks using dropconnect". In: *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*. 2013, pp. 1058–1066.

Wand, Michael, Jan Koutník, and Jürgen Schmidhuber. "Lipreading with Long Short-Term Memory". In: *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2016, pp. 6115–6119.

Wen, B. and K. Boahen. "A silicon cochlea With active coupling". In: *IEEE Trans. Biomed. Circuits Syst.* 3.6 (2009), pp. 444–455.

Weston, Jason, Sumit Chopra, and Antoine Bordes. "Memory networks". In: *arXiv preprint arXiv:1410.3916* (2014).

Xu, Kelvin, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard Zemel, and Yoshua Bengio. "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention". In: *International Conference on Machine Learning*. 2015.

Yang, M., C. H. Chien, T. Delbruck, and S. C. Liu. "A 0.5 V 55 μW 64 x 2 Channel Binaural Silicon Cochlea for Event-Driven Stereo-Audio Sensing". In: *IEEE Journal of Solid-State Circuits* 51.11 (Nov. 2016), pp. 2554–2569. ISSN: 0018-9200. DOI: 10.1109/JSSC.2016.2604285.

Yang, M., S. C. Liu, and T. Delbruck. "A Dynamic Vision Sensor With 1% Temporal Contrast Sensitivity and In-Pixel Asynchronous Delta Modulator for Event Encoding". In: *IEEE Journal of Solid-State Circuits* 50.9 (Sept. 2015), pp. 2149–2160. ISSN: 0018-9200.

Zilly, Julian Georg, Rupesh Kumar Srivastava, Jan Koutník, and Jürgen Schmidhuber. "Recurrent highway networks". In: *arXiv preprint arXiv:1607.03474* (2016).

Appendix: Source code and implementation details

The original FPGA source code for the Minitaur hardware system outlined in Chapter 2 has been handed over to Ilya Kiselev [28] who has dramatically improved the system from its original form. Further details about the project, including descriptions of the simulator, test suite, and real-world challenges (like the 14-hour place and route compilation process) can be obtained in an unpublished short project report from me or my advisor Shih-Chii Liu. The source code for the SpiNNaker implementation of the event-based Deep Belief Networks can be obtained from Evangelos Stromatias. The source code used to train an event-based Deep Belief Network can be obtained online [29].

[28] kiselev@ini.uzh.ch

[29] https://github.com/dannyneil/edbn

The code that converts a standard convolutional or fully-connected neural network to a spiking neural network was cowritten with Peter U. Diehl and Michael Pfeiffer, and is available online [30]. It has since been superseded by excellent advances from Bodo Rueckauer [31], who maintains an up-to-date toolbox with all the convesion tricks that increase performance. The original Matlab implementation does indeed work and remains available on Github, but the newest version of the toolbox includes code that runs dramatically faster, converts much more powerful networks, and helps elucidate why conversion does not go well if there are challenges. Please contact him if you would like to pursue this research direction to be caught up on the state-of-the-art.

[30] https://github.com/dannyneil/spiking_relu_conversion

[31] rbodo@ini.uzh.ch

The source code for the algorithms in Chapter 4 have not been published, but can be made available on request. They are an extension of the Matlab DeepLearnToolbox, and the last work in this thesis that was done using Matlab and trained without the use of a GPU. Training time takes a few seconds per epoch; the completed parameter sweeps took a few weeks of training time to complete.

The PLSTM implementation in Chapter 5 has been made available online, with a demonstration implementation of the first (frequency discrimination) task [32]. Alternative implementations have been performed by the community, with available implementations in Keras [33], TensorFlow with the frequency task[34] and on a recurrent form of MNIST [35], and now an official Google implementation is available in the TensorFlow main branch [36]. That work discusses the training time in more detail, but in general is around 50% longer wall-clock time for the same number of epochs. Excitingly, for many applications, accelerated convergence allows

[32] https://github.com/dannyneil/public_plstm

[33] https://github.com/fferroni/PhasedLSTM-Keras

[34] https://github.com/Enny1991/PLSTM

[35] https://github.com/philipperemy/tensorflow-phased-lstm

[36] https://github.com/tensorflow/tensorflow/blob/master/tensorflow/contrib/rnn/python/ops/rnn_cell.py

earlier wall-clock completion, despite this per-computation slowdown. Hopefully, future work can accelerate the implementation per-timestep as well.

The alternative models presented in Chapter 6 are written in Theano, and can be made available upon request. In general, they take approximately the same time to compute as Phased LSTM, and are largely unoptimized as that work was more concerned with correctness than optimization. Training times for each experiment can be found in the description for that task.

The Delta RNN framework was originally developed under a grant from an industrial partner, and releasing source code can be challenging in this collaboration. The current maintainer of this code is Enea Ceolini [37], to whom I would direct your inquiries. The Delta RNN model was a prototype to lead to a hardware implementation, currently in development, so the current version is heavily instrumented to give statistics about sparsity and operations. It takes approximately twice as long to train per epoch as the standard GRU implementation, but decreasing the instrumentation will likely improve that. Using a sparse matrix implementation for a real-world speedup is possible, and may be pursued in future work.

[37] eceoli@ini.uzh.ch