

DISS. ETH NO. 27890

DEEP LEARNING BEYOND
THE TRAINING DISTRIBUTION

A thesis submitted to attain the degree of
DOCTOR OF SCIENCES OF ETH ZURICH
(Dr. sc. ETH Zurich)

presented by

GIAMBATTISTA PARASCANDOLO

M.Sc. in Information Technology
Tampere University of Technology

born on 16.02.1992
citizen of Italy

accepted on the recommendation of

Prof. Dr. Thomas Hofmann (ETH Zurich)
Prof. Dr. Bernhard Schölkopf (MPI Tübingen & ETH)
Prof. Dr. Yoshua Bengio (MILA)

2021

ABSTRACT

One of the goals of artificial intelligence is to create machines that can think like humans. *Deep learning* has been at the core of the remarkable progress made towards this goal. Large artificial neural networks trained on massive datasets can master tasks across vastly different domains. Despite the progress on *i.i.d. generalization* — i.e., when the training and test data are independently and identically distributed — these models struggle when tested outside of the support of the training data. But can we even expect to generalize out of the training distribution? In certain contexts, yes, and one of the hallmarks of human intelligence is to use our causal understanding of the data generating process to correctly make inference out of distribution (o.o.d.). This thesis investigates *four* different assumptions and techniques to support o.o.d. generalization with deep learning. (i) o.o.d. generalization via *composition*. By relying on the assumption of *independence of mechanisms* from the literature on causality, we learn a set of *modular, reusable* neural networks via competition of experts. These modules specialize and can be applied sequentially to account for novel combinations of transformations at test time; (ii) o.o.d. generalization via *invariances*, where the training data has a mixture of invariant and spurious features, and only the invariances support generalization at test time. We show that training neural networks with the arithmetic mean of gradients may lead to memorization and spurious features to emerge, while the geometric mean of the gradients suppresses them in favor of invariances; (iii) o.o.d. generalization via *symbolic expressions*: where identifying the correct underlying symbolic equation, as commonly done in the sciences, allows making accurate predictions far from the training distribution. We leverage large-scale pre-training to make

a neural network *learn* to predict symbolic equations from a set of input-output observations, vastly outperforming state-of-the-art hand-designed approaches; (iv) o.o.d. generalization via *planning*, a classic technique to reduce uncertainty by investing additional time and compute at test time to solve more complex instances of problems seen during training. We present a divide-and-conquer algorithm that builds on top of Monte Carlo Tree Search with neural policy and value functions. By recursively splitting the problem in half, horizons and their uncertainties get exponentially shorter as a function of planning depth, allowing the model to plan over much longer periods.

SOMMARIO

Uno degli obiettivi dell'intelligenza artificiale è quello di creare macchine che possano pensare come gli esseri umani. L'apprendimento profondo (*deep learning*) è stato al centro degli incredibili progressi fatti finora verso questo obiettivo. Grandi reti neurali artificiali addestrate su vasti insiemi di dati, possono padroneggiare compiti in domini molto diversi tra loro. Nonostante i progressi sulla generalizzazione i.i.d. — cioè quando i dati di allenamento e di test sono distribuiti in modo indipendente e identico — questi modelli faticano quando vengono testati al di fuori del supporto dei dati di allenamento. Ma è plausibile aspettarsi di generalizzare al di fuori della distribuzione dei dati di allenamento (o.o.d., per 'out of distribution')? In certi contesti sì, e uno dei tratti distintivi dell'intelligenza umana è quello di usare la nostra comprensione causale dei processi di generazione dei dati per fare inferenza correttamente anche al di fuori della distribuzione. In questa tesi, indaghiamo quattro diverse ipotesi e tecniche per supportare la generalizzazione o.o.d. con l'apprendimento profondo: (i) generalizzazione o.o.d. tramite *composizione*, dove basandosi sull'ipotesi di indipendenza dei meccanismi dalla letteratura sulla causalità, alleniamo un insieme di reti neurali *modulari e riutilizzabili* tramite la competizione di esperti. Questi moduli si specializzano durante l'addestramento e possono essere applicati in modo sequenziale per tenere conto di nuove combinazioni di trasformazioni al tempo di test; (ii) generalizzazione o.o.d. tramite *invarianze*, dove i dati di allenamento hanno un misto di caratteristiche invariante e spurie, e solo le invarianze supportano la generalizzazione al momento del test. Mostriamo che l'addestramento delle reti neurali con la media aritmetica dei gradienti può portare alla memorizzazione e all'emergere di caratteristiche spu-

rie, mentre la media geometrica dei gradienti le riduce a favore delle invarianze; (iii) la generalizzazione o.o.d. tramite *espressioni simboliche*, dove l'identificazione della corretta equazione simbolica sottostante, come comunemente fatto nelle scienze, permette di fare previsioni accurate lontano dalla distribuzione di allenamento. Sfruttiamo il pre-addestramento su larga scala per far sì che una rete neurale impari a predire equazioni simboliche da un insieme di osservazioni input-output, superando di gran lunga lo stato dell'arte degli approcci progettati a mano; (iv) la generalizzazione o.o.d. tramite la *pianificazione*, una tecnica classica per ridurre l'incertezza investendo ulteriore tempo e calcolo al momento del test per risolvere istanze più complesse di problemi visti durante l'allenamento. Presentiamo un algoritmo divide-et-impera che si basa sulla *Ricerca ad albero Monte Carlo* con policy e funzioni di valore neurali. Suddividendo ricorsivamente il problema a metà, gli orizzonti diventano esponenzialmente più brevi in funzione della profondità di pianificazione, permettendo al modello di pianificare su tempi molto più lunghi.

ACKNOWLEDGEMENTS

These have been the most exciting and fulfilling years of my life. I want to thank my advisors Bernhard and Thomas, and the Center for Learning System, for giving me the opportunity to start this wonderful adventure. To Bernhard, for encouraging me to ask the big questions of generalization from the very beginning, and our many great collaborations. To Thomas, for making me a part of his lab at ETH while the world had to go into lockdown. To Yoshua, for inspiring me with his revolutionary research on neural networks over the years, without which this thesis would not exist, and for being on my graduation committee.

My time in Tübingen would not have been the same without *the pups*: the Axel (Alex), Niki and Mateo woof, Paul, Alessandro, John, Matej. Thank you for the memories, the laugh, the trips, the internships, the dinners, the postcards, the inspiration, the encouragement, the friendship. A special thanks to Axel for our countless collaborations and projects, for our daily chats, for teaching me RL and proper coding, for showing me new perspectives on research, for being a role model of humbleness and open-mindedness.

To Gigi for our *pastiere*. To Timmy for his L^AT_EX-wizardry and his disturbingly wide trivia knowledge, that single-handedly won pub quizzes for our team '*the Empirical Inferencers*' — Matthias, Francesco, Axel, Gigi. To the *volley-pong* team — Dieter, Justus, Axel, Mara — for helping shape the rules of the sport of the future. A big thanks to everyone else in the department of Empirical Inference, and in particular to Sabrina for all her help over the years. And to everyone I was lucky to meet on other floors of the MPI: to Cristina for the genuineness and the pizza parties,

to Michal for bringing quality to our MPI football games, to Yana for our delightful and reflective chats, and many more.

For my stay in Zurich I want to thank the DaLab for the warm welcome, and in particular Antonio for our film photography and the long nights discussing invariances surrounded by his mystical incenses (mixed with the fumes of the film developers).

I want to thank my team at Google X for my time there, in particular Jesse and Patricia for patiently teaching me all about automated design for electromagnetic devices.

Thanks to everyone at DeepMind who made my stay so memorable: my collaborators, Dave (for the precious feedback), Demis (for teaching me and Niki how to lose a table-football semi-final), Geoff (for the inspiring chat during his visit), the pups (for the music, the adventures, and mutual support), and in particular Lars for teaching me all about MCTS and the fun time together.

To the open source projects that make so much research possible: Python, Numpy, Pytorch, \LaTeX , and many more.

To my great collaborators, for sharing the excitement of discovery over the years.

To those I met over these years across the globe and enriched my life: to Linda $\times_{\leq 3}$, Simone, Cusuh, Nina, Bianca, Erica, and everyone else I am forgetting to mention.

Ai miei genitori, per avermi trasmesso l'amore per la conoscenza. Questo lavoro è dedicato a loro.

PUBLICATIONS

The material presented in this thesis is mostly based on the following articles:

1. **"Learning Independent Causal Mechanisms"**
Giambattista Parascandolo, Niki Kilbertus, Mateo Rojas-Carulla, Bernhard Schölkopf
ICML 2018: *International Conference on Machine Learning*
Parascandolo et al., 2018
2. **"Learning explanations that are hard to vary"**
Giambattista Parascandolo*, Alexander Neitz*, Antonio Orvieto, Luigi Gresele, Bernhard Schölkopf
ICLR 2021: *International Conference on Learning Representations*
Parascandolo et al., 2021a
3. **"Neural Symbolic Regression that Scales"**
Luca Biggio*, Tommaso Bendinelli*, Alexander Neitz, Aurelien Lucchi, Giambattista Parascandolo
ICML 2021: *International Conference on Machine Learning*
Biggio et al., 2021
4. **"Divide-and-Conquer Monte Carlo Tree Search"**
Giambattista Parascandolo*, Lars Buesing*, Josh Merel, Leonard Hasenclever, John Aslanides, Jessica Hamrick, Nicolas Heess, Alexander Neitz, Theophane Weber
under submission
Parascandolo et al., 2021b

* indicates equal contribution.

The following publications were part of my PhD research, but are not covered in this dissertation:

5. **"Avoiding Discrimination through Causal Reasoning"**
Niki Kilbertus, Mateo Rojas-Carulla, Giambattista Parascandolo, Moritz Hardt, Dominik Janzing, Bernhard Schölkopf
NeurIPS 2017: *Advances in Neural Information Processing Systems*
Kilbertus et al., 2017
6. **"ConvWave: Searching for Gravitational Waves with Fully Convolutional Neural Nets"**
Timothy Gebhard., Niki Kilbertus., Giambattista Parascandolo, Ian Harry, Bernhard Schölkopf
NeurIPS 2017 Workshop: *Deep Learning for Physical Sciences*
Gebhard et al., 2017
7. **"Tempered Adversarial Networks"**
Mehdi Sajjadi, Giambattista Parascandolo, Arash Mehrjou, Bernhard Schölkopf
ICML 2018: *International Conference on Machine Learning*
Sajjadi et al., 2018
8. **"Adaptive Skip Intervals: Temporal Abstraction for Recurrent Dynamical Models"**
Alexander Neitz, Giambattista Parascandolo, Stefan Bauer, Bernhard Schölkopf
NeurIPS 2018: *Advances in Neural Information Processing Systems*
Neitz et al., 2018
9. **"Generalization in anti-causal learning"**
Niki Kilbertus*, Giambattista Parascandolo*, Bernhard Schölkopf*
NeurIPS 2018 Workshop: *Critiquing and correcting trends in*

machine learning

Kilbertus* et al., 2018

10. **"A teacher-student framework to distill future trajectories"**

Alexander Neitz*, Giambattista Parascandolo*,
Bernhard Schölkopf

ICLR 2021: International Conference on Learning Representations

Neitz et al., 2021

The following articles are still unpublished and are not covered in this dissertation:

11. **"Topographic Neural Networks"**

Federico Pirovano, Giambattista Parascandolo
under submission

PROLOGUE

As Dante descends into the 8th Circle of the *Inferno*, he meets Ulysses. After the war, on his way back home, Ulysses had to fight against a six-headed monster, the Sirens, a Cyclops, and a goddess-witch, before reaching the shores of his kingdom ten long years later. Dante asks him what had happened next: did Ulysses stay in Ithaca, rejoined with his family, after reconquering his kingdom? Not for long, says Ulysses:

*né dolcezza di figlio, né la pièta
del vecchio padre, né 'l debito amore
lo qual dovea Penelopè far lieta,*

*vincer potero dentro a me l'ardore
ch' i' ebbi a divenir del mondo esperto
e de li vizi umani e del valore;*

*ma misi me per l'alto mare aperto
sol con un legno e con quella compagna
picciola da la qual non fui disertò.*

*L'un lito e l'altro vidi infin la Spagna,
fin nel Morrocco, e l'isola d' i Sardi,
e l'altre che quel mare intorno bagna.*

*Io e' compagni eravam vecchi e tardi
quando venimmo a quella foce stretta
doo' Ercole segnò li suoi riguardi*

*acciò che l'uom più oltre non si metta;
da la man destra mi lasciai Sibilia,
da l'altra già m'avea lasciata Setta.*

*"O frati," dissi, "che per cento milia
perigli siete giunti a l'occidente,*

*not tenderness for a son, nor filial duty
toward my aged father, nor the love I owed
Penelope that would have made her glad,*

*could overcome the fervor that was mine
to gain experience of the world
and learn about man's vices, and his worth.*

*And so I set forth upon the open deep
with but a single ship and that small band
of shipmates who had not deserted me.*

*One shore and the other I saw as far as Spain,
Morocco, the island of Sardegna,
and other islands set into that sea.*

*I and my shipmates had grown old and slow
before we reached the narrow strait
where Hercules marked off the limits,*

*warning all men to go no farther.
On the right-hand side I left Seville behind,
on the other I had left Ceüta.*

*"O brothers," I said, "who, in the course
of a hundred thousand perils, at last*

a questa tanto picciola vigilia

have reached the west, to such brief wakefulness

d'i nostri sensi ch'è del rimanente
non vogliate negar l'esperienza,
di retro al sol, del mondo sanza gente.

of our senses as remains to us,
do not deny yourselves the chance to know—
following the sun—the world where no one lives.

Considerate la vostra semenza:
fatti non foste a viver come bruti,
ma per seguir virtute e canoscenza”

Consider how your souls were sown:
you were not made to live like brutes or beasts,
but to pursue virtue and knowledge.”

Dante Alighieri, *La Divina Commedia – Inferno, Canto XXVI vv 94-120*¹

Restless, with an unquenchable thirst for knowledge, Ulysses had taken a little ship, a few companions, and set off to the *Pillars of Hercules*, the edge of the known world on the Strait of Gibraltar. The Pillars of Hercules in the classical and pre-modern era were not only a geographical concept, but also metaphorically expressed the limits of knowledge and of what *could* be known.²

Our minds allow us to go vastly beyond the distribution of our experiences: we discovered the laws governing stars and atoms, inventing and mastering tools to expand our reach and explain the world around us. Ulysses embodies the desire and capacity to go *beyond the limits of our direct experience*, which (in his words) is ultimately what makes us human. This is the type of *generalization* that is still out of reach for artificial intelligence.

In the next pages we will embark on our own short voyage accompanied by a loyal crew of artificial neural networks. We will set sails beyond the known lands of what can be observed in training, and crossing the support of the training distribution — the *Pillars of Hercules* of learning — we will explore the mysterious ocean out of the distribution.

1 The English translation is by R. Hollander and J. Hollander.

2 The Pillars of Hercules appear on the title page of Francis Bacon's *Novum Organum*, where he advocated for empirical investigation as the foundation for science, to open up new frontiers of knowledge.

CONTENTS

1	INTRODUCTION AND OVERVIEW	1
1.1	Generalization and deep learning	1
1.2	Outline	3
2	LEARNING INDEPENDENT CAUSAL MECHANISMS	9
2.1	Introduction	10
2.2	Related work	13
2.3	Learning causal mechanisms as independent modules	15
2.3.1	Formal setting	15
2.3.2	Competitive learning of independent mechanisms	16
2.4	Experiments	20
2.5	Results	23
2.6	Conclusions	30
3	LEARNING EXPLANATIONS THAT ARE HARD TO VARY	33
3.1	Introduction	34
3.2	Explanations that are hard to vary	37
3.2.1	Formal definition of ILC	40
3.2.2	ILC as a logical AND between landscapes	42
3.2.3	Masking gradients with a logical AND	44
3.3	Experiments	47
3.3.1	The synthetic memorization dataset	47
3.3.2	Experiments on CIFAR-10	51
3.3.3	Behavioral Cloning on CoinRun	53
3.4	Related Work	56
3.5	Conclusions	57
4	NEURAL SYMBOLIC REGRESSION THAT SCALES	59
4.1	Introduction	60
4.2	Related Work	62

4.3	Neural Symbolic Regression that Scales	65
4.3.1	Pre-training	65
4.3.2	Test time	68
4.4	Experimental Set-up	68
4.4.1	The Model S_θ	69
4.4.2	Pre-training Data Generator	69
4.4.3	Symbolic Regression at Test Time	71
4.4.4	Evaluation	71
4.4.5	Baselines	72
4.4.6	Metrics	74
4.5	Results	75
4.6	Discussion	80
5	DIVIDE-AND-CONQUER MONTE CARLO TREE SEARCH	83
5.1	Introduction	84
5.2	Improving Goal-Directed Policies with Planning	86
5.2.1	Planning over Sub-Goal Sequences	87
5.2.2	AND/OR Search Tree Representation	89
5.3	Best-First AND/OR Planning	90
5.3.1	Divide-and-Conquer Monte Carlo Tree Search	92
5.3.2	Designing and Training Search Heuristics	94
5.3.3	Algorithmic Complexity of DC-MCTS	95
5.4	Related Work	96
5.5	Experiments	98
5.5.1	Grid-World Mazes	98
5.5.2	Continuous Control Mazes	103
5.5.3	Visualizing MCTS and DC-MCTS	105
5.6	Discussion	106
6	CONCLUSION	109
6.1	Recap	109
6.2	Other axes of o.o.d. generalization	110
6.3	i.i.d. or o.o.d.? A distinction blurred by scale.	111

6.4	On sample-(in)efficiency of human and machine intelligence	112
A	APPENDIX LICM	115
A.1	Additional results.	115
A.1.1	Too many or too few experts.	115
A.2	Details of neural networks	116
A.3	Transformations	116
A.4	Notes on the Formalization of Independence of Mechanisms	117
B	APPENDIX ILC	119
B.1	Appendix to Section 3.2	119
B.1.1	A classic example of a patchwork solution	119
B.1.2	Section 3.2.2: Consistency as arithmetic/-geometric mean of landscapes	121
B.1.3	Proof of Proposition 1	125
B.1.4	Proof of Proposition 2	126
B.2	Appendix to Section 3.3	130
B.2.1	Section 3.3.1	130
B.2.2	Dataset	130
B.2.3	Experiment	132
B.2.4	Further visualizations and experiments	135
B.2.5	Section 3.3.2: CIFAR-10 memorization and label noise experiments	136
B.2.6	Section 3.3.3: Behavioral Cloning on Coin-Run	137
B.3	Appendix to Section 3.4	140
B.3.1	Related work in causal inference	140
B.3.2	Learning invariances in the data	141
C	APPENDIX EQL	143
C.1	Model details	143
C.1.1	NeSymReS Transformer Details	143
C.1.2	Baselines	145
C.2	Experimental details	147

c.2.1	Training	147
c.2.2	Evaluation details	150
c.3	Additional Results	154
c.3.1	Additional Metrics on all Benchmarks	154
D	APPENDIX DC-MCTS	157
D.1	Additional Details for DC-MCTS	157
D.1.1	Proof of Proposition 1	157
D.1.2	Additional algorithmic details	158
D.1.3	Descending into one node at the time during search	158
D.1.4	Parsers for Hindsight Experience Replay	161
D.1.5	Details on Algorithmic Complexity	162
D.2	Training details	163
D.2.1	Details for training the value function	163
D.2.2	Details for training the policy prior	164
D.2.3	Neural networks architectures for grid- world experiments	165
D.2.4	Low-level controller training details	166
D.2.5	Pseudocode	167
D.3	More solved mazes	168
D.3.1	Supplementary material and videos	168
	BIBLIOGRAPHY	171

INTRODUCTION AND OVERVIEW

1.1 GENERALIZATION AND DEEP LEARNING

One of the goals in the field of artificial intelligence is to create machines that can think like humans. In the past ten years we have witnessed an incredible progress towards this goal, as machine learning research has focused heavily on *artificial neural networks*, also called *deep learning* (LeCun et al., 2015). Deep learning models are computer programs that implement networks of simple interconnected units inspired by biological neurons. The connections between these units (also called neurons) determine how signals propagate and interact, and what outputs and behaviors are eventually expressed by the network. Similarly to what happens in a biological brain, in an artificial neural network *learning* amounts to adjusting the connections between the neurons in response to experience, with the objective of improving its behavior on a given task.¹

One of the most impressive aspects about deep learning is that, akin to biological brains, these artificial neural networks can master tasks across a very diverse set of domains, in some cases reaching human level or super-human level performance. The list of domains goes from computer vision (Krizhevsky et al., 2012; Ramesh et al., 2021; Radford et al., 2021) to natural language processing (NLP) (Brown et al., 2020), audio processing (Oord et al., 2018b), game playing (Silver et al., 2016; Silver et al., 2017), to robotics and continuous control (Akkaya et al., 2019), all the way to protein folding (Jumper et al., 2021) and more.

¹ For a thorough technical background on deep learning see Goodfellow et al., 2016.

Artificial neural networks have a large expressive power that has also been quantified theoretically. Multi Layer Perceptrons (MLPs) are universal approximators, i.e., given enough hidden units and non-linear activations, they can approximate any well-behaved function to desired precision on a bounded domain (Cybenko, 1989).² Recurrent Neural Networks (RNNs) are even more expressive (Siegelmann and Sontag, 1995), as they can compute any computable function (i.e. implement any algorithm) by simulating a pushdown automaton with two stacks, which in turn can implement any Turing machine.

Trained with very large amounts of data, these powerful models can learn to solve complex tasks, as long as we test them on inputs that are not ‘too far’ from what they observed in training. Technically, we talk about a *training distribution* P_{train} , from which the training data D_{train} is sampled, and a *test distribution* P_{test} , that may or may not coincide with P_{train} . If they do coincide we call the problem of generalizing to novel test examples *i.i.d. generalization* (for independently and identically distributed), or often in the literature simply *generalization*. This is the most well known, studied and formalized type of generalization, and for which *statistical learning theory* provides a rigorous foundation under a small set of assumptions (Bishop, 1995; Vapnik, 1999). However, the test distribution P_{test} does not always coincide with P_{train} in practice. The term *covariate shift* (Shimodaira, 2000) is often used to describe the setting where $P_{train}(X) \neq P_{test}(X)$, but $P(Y|X)$ stays the same. In this context, the ratio $P_{test}(x)/P_{train}(x)$ (sometimes referred to as *importance*) is assumed to be finite (Shimodaira, 2000) for all x , i.e., the support of the test distribution is assumed to be fully contained within the support of the training distribution; in this setting, the importance can be used as a re-weighting factor (Shimodaira, 2000).

² In Appendix B.1.1 we review in detail a classic technique to construct such an approximation.

In this work we will look at the more extreme case, where the supports of $P_{train}(X)$ and $P_{test}(X)$ are disjoint, and therefore importance weighting is not an option. This is a challenging setting, where even the the notion of ‘ $P(Y|X)$ stays the same’ might sound vacuous under these conditions. Moreover, how can we expect to make meaningful predictions outside of the support of the datapoints we observed during training? If anything can happen beyond the support of the training distribution, *out-of-distribution generalization* (o.o.d.) should more fittingly be called wishful thinking. The literature on causal inference (Peters et al., 2017b) provides a foundation with the necessary assumptions to ask these questions, by invoking the underlying causal generating process of the data (for example expressed by a structural equation model (Pearl, 2000)).

The use of causal reasoning to support o.o.d. generalization in novel situations is regarded as one of the hallmarks of human intelligence. While the progress of deep learning on *i.i.d. generalization* has been outstanding — even for domains where the training distribution is extremely wide and complex — *o.o.d. generalization* is still a very open research problem. This is the general setting investigated in this thesis. In particular, we examine *four* different kinds of out-of-distribution generalization in deep learning under different assumptions: for each one of them, we analyze where the traditional approaches fall short and contribute novel learning algorithms that can improve them.

1.2 OUTLINE

This thesis consists of four main chapters, with each chapter investigating a different aspect of out-of-distribution generalization with neural networks. A final chapter presents conclusions and perspectives on the future. Each chapter is based on a paper that I worked on during my PhD with my co-authors, and will cover

different learning modalities including unsupervised learning, supervised learning, and reinforcement learning. Each chapter discusses the technical background required to understand the algorithms, theory, and experiments presented within.

Chapter 2 – o.o.d. generalization via composition

In Chapter 2 the main assumption is that the data observed is generated by a set of *independent causal mechanisms* (Schölkopf et al., 2012; Peters et al., 2017b) and that these mechanisms can be *composed* in novel combinations at test time. We present an algorithm centered around a *competition of experts* to leverage the independence of the causal mechanisms that generated the data. The algorithm allows to train, without label supervision, a set of *modules* that specialize in inverting one mechanism at the time. By applying these trained modules one after the other, we show that they can generalize out of distribution to novel combinations of transformations. One of the objectives of this work is to not only disentangle the factors of variations, but to go one step further and *disentangle the mechanisms* such that they can be re-arranged at test time as needed. Chapter 2 is based on Parascandolo et al., 2018

“Learning Independent Causal Mechanisms”

Giambattista Parascandolo, Niki Kilbertus, Mateo Rojas-Carulla, Bernhard Schölkopf

ICML 2018: *International Conference on Machine Learning*

Chapter 3 – o.o.d. generalization via invariance

The main assumption for Chapter 3 is that the o.o.d. test data can be correctly classified if the model learns to rely on the *in-*

variant features. At training time the model observes datapoints that come from a set of different environments that share a common mechanism. We show that standard training using gradient descent on the loss encourages fast convergence at the cost of potentially relying on spurious patterns, as a pattern will be reinforced even if it produces gradients only in a single environment. This can result in a ‘patchwork’ solution, a logical OR between patterns that appear in some environments, but not necessarily in all of them. Inspired by David Deutsch’s principle that ‘*good explanations should be hard to vary*’, we investigate how to learn a representation that focuses on the invariant patterns. We show that the geometric mean of the gradients can act as a logical AND, preserving patterns that occur in a substantial number of environments. We derive a simple algorithm — that we call the AND-mask — that has the same linear time complexity as standard training with empirical risk minimization using the arithmetic mean of the gradients. We test the AND-mask against state-of-the-art baselines for learning invariant representations, and evaluate it on a set of supervised learning tasks and an imitation learning task via behavioral cloning. Chapter 3 is based on [Parascandolo et al., 2021a](#)

“Learning explanations that are hard to vary”

Giambattista Parascandolo*, Alexander Neitz*,

Antonio Orvieto, Luigi Gresele, Bernhard Schölkopf

ICLR 2021: *International Conference on Learning Representations*

Chapter 4 – o.o.d. generalization via symbolic expressions

Symbolic representations of physical phenomena are ubiquitous in the sciences, and usually take the form of *equations*. The most appealing property of symbolic equations is that they describe complex phenomena in a very compressed and interpretable rep-

resentation. From the point of view of o.o.d. generalization, symbolic equations are incredible tools. Take for example *general relativity*: it was discovered by making a small number of observations, and yet it can successfully predict physical phenomena light-years apart across the universe. Despite the appealing properties of symbolic representations, the process of *discovering* an equation from observations is a challenging task. State-of-the-art approaches are typically complex hand-designed symbolic methods like Genetic Programming. Deep learning models are not typically used in this setting given (i) their poor o.o.d. generalization if the network is used to fit $y = f_{\theta}(x)$, (ii) the solution is not interpretable, and (iii) that they need large quantities of training data. To combine the strengths of symbolic representations and deep neural networks, we investigate an approach that *learns* end-to-end to discover equations from observations. We large-scale pre-train on billions of equations a neural network that takes as input a set of observations $\{(x_i, y_i)\}$, and outputs autoregressively a symbolic equation, symbol by symbol, just like a language model. The pre-training data can be generated quickly on the fly by any standard computer, from a pre-defined and controllable distribution over equations, which provides a simple and interpretable way to tune the inductive bias of the model. Across several datasets, our method that scales with experience and compute outperforms state-of-the-art baselines by a factor of 100x in time, even while running on CPU. Chapter 4 is based on [Biggio et al., 2021](#)

“Neural Symbolic Regression that Scales”

Luca Biggio* , Tommaso Bendinelli* , Alexander Neitz, Aurelien Lucchi, Giambattista Parascandolo

ICML 2021: *International Conference on Machine Learning*

Chapter 5 – o.o.d. generalization via planning

Achieving goals that are far in the future is a challenging task. *Planning* is a classic technique used to reduce uncertainty: by investing additional time and compute at test time, we can generalize o.o.d. to problem instances that are more complex than those seen at training time. Most planning algorithms are *sequential* in nature (like Monte Carlo Tree Search, Model Predictive Control, etc.); for example AlphaGo (Silver et al., 2016) explores a sequence of moves from the current position. However, for goals far in the future, planning sequentially cannot significantly reduce the long-term uncertainty, as the planning resources reduce it only near the current state and the goal is far off. Human intelligence, instead, can plan by mentally traveling back and forth in time, iteratively reducing uncertainty across the entire plan: e.g., to plan a trip from Rome to Tokyo, we would start by looking for flights, then go forward and book a hotel, then go backwards and schedule a visa appointment, etc., and not by thinking about the elevator to leave the building we are currently in, then calling a taxi, and so on. In Chapter 5, we study planning over long horizons with goal-directed reinforcement learning that follows a *divide-and-conquer* approach. The algorithm we present — Divide-and-Conquer Monte Carlo Tree Search (DC-MCTS) — trains a policy and a value network to solve long horizon problems by learning to recursively subdividing them where necessary to reduce uncertainty, jumping back and forth in time. Chapter 5 is based on Parascandolo et al., 2021b

“Divide-and-Conquer Monte Carlo Tree Search”

Giambattista Parascandolo*, Lars Buesing*, Josh Merel, Leonard Hasenclever, John Aslanides, Jessica Hamrick, Nicolas Heess, Alexander Neitz, Theophane Weber
under submission

LEARNING INDEPENDENT CAUSAL MECHANISMS

CHAPTER ABSTRACT Statistical learning relies upon data sampled from a distribution, and we usually do not care what actually generated it in the first place. From the point of view of causal modeling, the structure of each distribution is induced by physical mechanisms that give rise to dependences between observables. Mechanisms, however, can be meaningful autonomous modules of generative models that make sense beyond a particular entailed data distribution, lending themselves to transfer between problems. We develop an algorithm to recover a set of independent (inverse) mechanisms from a set of transformed data points. The approach is unsupervised and based on a set of experts that compete for data generated by the mechanisms, driving specialization. We analyze the proposed method in a series of experiments on image data. Each expert learns to map a subset of the transformed data back to a reference distribution. The learned mechanisms generalize to novel domains. We discuss implications for transfer learning and links to recent trends in generative modeling.

This chapter is based on the paper *“Learning Independent Causal Mechanisms”*, Giambattista Parascandolo, Niki Kilbertus, Mateo Rojas-Carulla, Bernhard Schölkopf (Parascandolo et al., 2018)

2.1 INTRODUCTION

Humans are able to recognize objects such as handwritten digits based on distorted inputs. They can correctly label translated, corrupted, or inverted digits, without having to re-learn them from scratch. The same applies for new objects, essentially after having seen them once. Arguably, human intelligence utilizes *mechanisms* (such as translation) that are independent from an input domain and thus generalize across object classes. These mechanisms are *modular, re-usable and broadly applicable*, and the problem of learning them from data is fundamental for the study of transfer and domain adaptation.

In the field of causality, the concept of independent mechanisms plays a central role both on the conceptual level and, more recently, in applications to inference. The *independent mechanisms (IM)* assumption states that the causal generative process of a system’s variables is composed of autonomous modules that do not inform or influence each other (Schölkopf et al., 2012; Peters et al., 2017a).

If a joint density is Markovian with respect to a directed graph \mathcal{G} , we can write it as

$$p(\mathbf{x}) = p(x_1, \dots, x_d) = \prod_{j=1}^d p(x_j | \text{pa}_{\mathcal{G}}^j), \quad (1)$$

where $\text{pa}_{\mathcal{G}}^j$ denotes the parents of variable x_j in the graph. For a given joint density, there are usually many decompositions of the form equation 1, with respect to different graphs. If \mathcal{G} is a *causal graph*, i.e., if its edges denote direct causation (Pearl, 2000), then the conditional $p(x_j | \text{pa}_{\mathcal{G}}^j)$ can be thought of as physical *mechanism* generating x_j from its parents, and we refer to it as a *causal conditional*. In this case, we consider the factorization equation 1 a *generative model* where the term “generative” truly refers to a

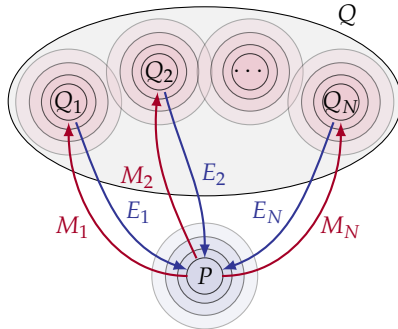


Figure 1: An overview of the problem setup. Given a sample from a canonical distribution P , and one from a mixture of transformed distributions Q_i obtained by mechanisms M_i on P , we want to learn inverse mechanisms E_i as independent modules. Modules (or *experts*) compete amongst each other for data points, encouraging specialization.

physical generative process. As an aside, we note that in the alternative view of causal models as structural equation models, each of the causal conditionals corresponds to a functional mapping and a noise variable (Pearl, 2000).

By the IM assumption, the causal conditionals are autonomous modules that do not influence or inform each other. This has multiple consequences. First, knowledge of one mechanism does not contain information about another one (Appendix A.4). Second, if one mechanism changes (e.g., due to distribution shift), there is no reason that other mechanisms should also change, i.e., they tend to remain *invariant*. As a special case, it is (in principle) possible to locally *intervene* on one mechanism (for instance, by setting it to a constant) without affecting any of the other modules. In all these cases, most of equation 1 will remain unchanged. However, since the overall density will change, most generic (non-causal) conditionals would change.

The IM assumption can be exploited when performing causal structure inference (Peters et al., 2017a). However, it also has implications for machine learning more broadly. A model which is expressed in terms of causal conditionals (rather than conditionals with respect to some other factorization) is likely to have components that better transfer or generalize to other settings (Schölkopf et al., 2012), and its modules are better suited for building complex models from simpler ones. Independent mechanisms as sub-components can be trained independently, from multiple domains, and are more likely to be re-usable. They may also be easier to *interpret* and provide more *insight* since they correspond to physical mechanisms.

Animate intelligence cannot afford to learn new models from scratch for every new task. Rather, it is likely to rely on robust local components that can flexibly be re-used and re-purposed. It also requires local mechanisms for adapting and training modules rather than re-training the whole brain every time a new task is learned. Currently, machine learning excels at optimizing well-defined tasks from large i.i.d. datasets. However, if we want to move towards life-long learning and generalization across tasks, then we need to understand how modules can be learnt from data and shared between tasks.

In this chapter, we focus on a class of such modules, and on algorithms to learn them from data. We describe an architecture using competing experts that automatically specialize on different image transformations. The resulting model is attractive for lifelong learning, with the possibility of easily adding, removing, retraining, and re-purposing its components independently. It is unsupervised in the sense that the images are not labelled by the transformations they have undergone. We only need a sample from a reference distribution and a set of transformed images. The transformed images are based on another sample, and no pairing or information about the transformations is available.

We test our approach on *MNIST* digits which have undergone various transformations such as contrast inversion, noise addition and translation. Information about the nature and number of such transformations is not known at the beginning of training. We identify the independent mechanisms linking the reference distribution to a distribution of modified digits, and learn to invert them without supervision.

The inverse mechanisms can be re-purposed as preprocessors, to transform modified digits which are subsequently classified using a standard *MNIST* classifier. The trained experts also generalize to *Omniglot* characters, none of which were seen during training. These are promising results pointing towards a form of robustness that animate intelligence excels at.

2.2 RELATED WORK

Our work mainly draws from mixtures of experts, domain adaptation, and causality.

Early works on mixture of experts date back to the early nineties (Jacobs et al., 1991; Jordan and Jacobs, 1994), and since then the topic has been subject of extensive research. Recent work includes that of Shazeer et al., 2017, successfully training a mixture of 1000 experts using a gating mechanism that selects only a fraction of experts for each example. Aljundi et al., 2017 train a network of experts on multiple tasks, with a focus on lifelong learning; autoencoders are trained for each task and used as gating mechanisms. Lee et al., 2016b propose Stochastic Multiple Choice Learning, an algorithm which resembles the one we describe in Section 2.3, aimed at training mixture of experts to propose a diverse set of outputs. The main differences are that our model is trained jointly with a *learned* selection system which is valid also at test time, that our trained experts learn independent mechanisms and

can be combined (cf. Figure 8), and in the way experts are initialized.

Another research direction that is relevant to our work is unsupervised domain adaptation (Bousmalis et al., 2017). These methods often use some supervision from labeled data and/or match the two distributions in a learned feature space (e.g. Tzeng et al., 2017).

The novelty of our work lies in the following aspects: (1) we automatically identify and invert a set of independent (inverse) causal mechanisms; (2) we do so using only data from an original distribution and from the mixture of transformed data, without labels; (3) the architecture is modular, can be easily expanded, and its trained modules can be reused; and (4) the method relies on competition of experts.

Ideas from the field of causal inference inspire this work. Understanding the data generating mechanisms plays a key role in causal inference, and goes beyond the statistical assumptions usually exploited in machine learning. Causality provides a framework for understanding how a system responds to *interventions*, and causal graphical models as well as structural equation models (SEM) are common ways of describing causal systems (Pearl, 2000; Peters et al., 2017a). The IM assumption discussed in the introduction can be used for identification of causal models (Daniusis et al., 2010; Zhang et al., 2015), but causality has also proven a useful tool for discussing and understanding machine learning in the non-i.i.d. regime. Recent applications include semi-supervised learning (Schölkopf et al., 2012) and transfer learning (Rojas-Carulla et al., 2015), in which the authors focus only on linear regression models. We seek to extend applications of causal inference to more complex settings and aim to learn causal mechanisms and ultimately causal SEMs without supervision.

On the conceptual level, our setting is related to recent work on deep learning for disentangling factors of variation (Chen et al.,

2016; Higgins et al., 2017) as well as non-linear ICA (Hyvärinen and Morioka, 2016). In our work, causal mechanisms play the role of factors of variation. The main difference is that we recover mechanisms as independent modules.

2.3 LEARNING CAUSAL MECHANISMS AS INDEPENDENT MODULES

The aim of this section is twofold. First, we describe the generative process of our data. We start with a distribution P that we will call “canonical” and an a priori *unknown* number of independent mechanisms which act on (examples drawn from) P . At training time, a sample from the canonical distribution is available, as well as a dataset obtained by applying the mechanisms to (unseen) examples drawn from P . Second, we propose an algorithm which recovers and learns to invert the mechanisms in an unsupervised fashion.

2.3.1 Formal setting

Consider a canonical distribution P on \mathbb{R}^d , e.g., the empirical distribution defined by MNIST digits on pixel space. We further consider N measurable functions $M_1, \dots, M_N : \mathbb{R}^d \rightarrow \mathbb{R}^d$, called *mechanisms*. We think of these as independent causal mechanisms in nature, and their number is a priori unknown. A more formal definition of independence between mechanisms is relegated to Appendix A.4. The mechanisms give rise to N distributions Q_1, \dots, Q_N where $Q_j = M_j(P)$.¹ This setup is illustrated in Figure 1. In the MNIST example, we consider translations or adding noise as mechanisms, i.e., the corresponding Q distributions are translated and noisy MNIST digits.

¹ Each distribution Q_j is defined as the pushforward measure of P induced by M_j .

At training time, we receive a dataset $\mathcal{D}_Q = (x_i)_{i=1}^n$ drawn i.i.d. from a mixture of Q_1, \dots, Q_N , and a dataset \mathcal{D}_P sampled independently from the canonical distribution P . Our goal is to identify the underlying mechanisms M_1, \dots, M_N and learn approximate inverse mappings which allow us to map the examples from \mathcal{D}_Q back to their counterpart in P .

If we were given distinct datasets \mathcal{D}_{Q_j} each drawn from Q_j , we could individually learn each mechanism, resulting in independent (approximations of the) mechanisms regardless of the properties of the training procedure. This is due to the fact that the datasets are drawn from independent mechanisms in the first place, and the separate training procedure cannot generate a dependence between them. This statement does not require that the procedure is successful, i.e., that the obtained mechanisms approximate the true M_j in some metric.

In contrast, we do not require access to the distinct datasets. Instead we construct a larger set \mathcal{D}_Q by first taking the union of the sets \mathcal{D}_{Q_j} , and then applying a random permutation. This corresponds to a dataset where each element has been generated by one of the (independent) mechanisms, but we do not know by which one. Clearly, it should be harder to identify and learn independent mechanisms from such a dataset. We next describe an approach to handle this setting.

2.3.2 *Competitive learning of independent mechanisms*

The training machine is composed of N' parametric functions $E_1, \dots, E_{N'}$ with distinct trainable parameters $\theta_1, \dots, \theta_{N'}$. We refer to these functions as the *experts*. Note that we do not require $N' = N$, since the real number of mechanisms is unknown a priori. The goal is to maximize an objective function $c : \mathbb{R}^d \rightarrow \mathbb{R}$ with the key property that c takes high values on the support of the canonical distribution P , and low values outside. Note

that c could be a parametric function, and its parameters could be jointly optimized with the experts during training. Below, we specify the details of this rather general definition.

During training, the experts compete for the data points. Each example x' from \mathcal{D}_Q is fed to all experts independently and in parallel. Comparing the outputs of all experts $c_j = c(E_j(x'))$, we select the winning expert E_{j^*} , where $j^* = \arg \max_j (c_j)$. Its parameters θ_{j^*} are updated such as to maximize $c(E_{j^*}(x'))$, while the other experts remain unchanged. The motivation behind competitively updating only the winning expert is to enforce specialization; the best performing expert becomes even better at mapping x' back to the corresponding example from the canonical distribution. We will describe below that alongside with the expert's parameters, we train parameters of c (which in our experiments will be carried in an adversarial fashion). Figure 2 depicts this procedure. Overall, our optimization problem reads:

$$\theta_1^*, \dots, \theta_{N'}^* = \arg \max_{\theta_1, \dots, \theta_{N'}} \mathbb{E}_{x' \sim Q} \left[\max_{j \in \{1, \dots, N'\}} c(E_{\theta_j}(x')) \right]. \quad (2)$$

The training described above raises a number of questions, which we address next.

1. SELECTING THE APPROPRIATE NUMBER OF EXPERTS. Generally, the number of mechanisms N which generated the dataset \mathcal{D}_Q is not available a priori. Therefore, we require an adaptive procedure to choose the number of experts N' . This is one of the challenges shared with most clustering techniques. Given the modular behavior of the procedure, experts may be added or removed during or after training, making the framework very flexible. Assuming however that the number of experts is fixed, the following behaviors could occur.

If $N' > N$ (too many experts): a) some of the experts do not specialize and do not win any example in the dataset; or b) some

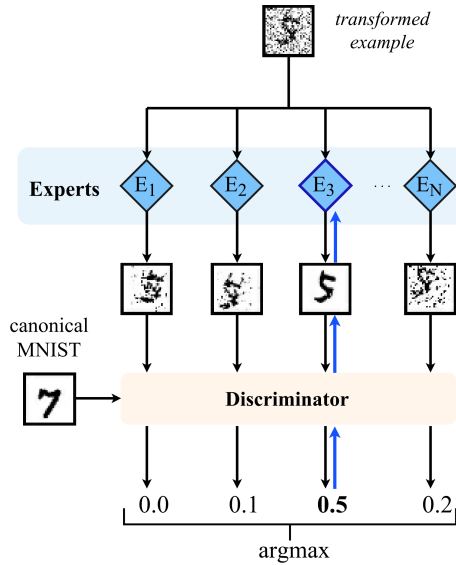


Figure 2: We show how a transformed example, here a noisy digit, is processed by a competition of experts. Only Expert 3 is specializing on denoising, it wins the example and gets trained on it, whereas the others perform translations and are not updated.

tasks are divided between experts (for instance, each expert can specialize in a mode of the distribution of the same task). In a), the inactive experts can be removed, and in b) experts sharing the same task can be merged into a wider expert.²

If $N' < N$ (too few experts): a) some of the experts specialize in multiple tasks or b) some of the tasks are not learned by the experts, so that data points from such tasks lead to a poor score

² However, note that in order to do this, it is necessary to first acknowledge that the two experts have learned part of the same task, which would require extra information or visual inspection.

across all experts. We provide experiments substantiating these claims in [A.1.1](#).

2. CONVERGENCE CRITERION. Since the problem is unsupervised, there is no straightforward way of measuring convergence, which raises the question of how to choose a stopping time for the competitive procedure. As an example, one may act according to one of the following: *a*) fix a maximum number of iterations or *b*) stop if each example is assigned to the same experts for a pre-defined number of iterations (i.e., each expert consistently wins the same data points).

3. TIME AND SPACE COMPLEXITY. Each example has to be evaluated by all experts in order to assign it to the winning expert. While this results in a computational cost that depends linearly on the number of experts, these evaluations can be done in parallel and therefore the time complexity of a single iteration can be bounded by the complexity to compute the output of a single expert. Moreover, as each expert will in principle have a smaller architecture than a single large network, the committee of experts will typically be faster to execute.

CONCRETE PROTOCOL FOR NEURAL NETWORKS. One possible model class for the experts are deep neural networks. Training using backpropagation is particularly well suited for the online nature of the training proposed: after an expert wins a data point x' , its parameters are updated by backpropagation, while the other experts remain untouched. Moreover, recent advances in generative modeling give rise to natural choices for the loss function c . For instance, through adversarial training ([Goodfellow et al., 2014](#)), one can use as objective function the output of a discriminator network trained on the canonical sample \mathcal{D}_P and against the outputs of the experts. In the next section we

introduce a formal description of a training procedure based on adversarial training in Algorithm 1, and empirically evaluate its performance.

While in this work we focus on adversarial training, preliminary experiments have shown that similar results can be achieved for example with variational autoencoders (VAE) (Kingma and Welling, 2014). Given a VAE trained on the canonical distribution P , one may define $c(x')$ as the opposite of the VAE loss.

2.4 EXPERIMENTS

In this set of experiments we test the method presented in Section 2.3 on the MNIST dataset transformed with the set of mechanisms described in detail in the Appendix A.3, i.e. eight directions of translations by 4 pixels (up, down, left, right, and the four diagonals), contrast inversion, addition of noise, for a total of 10 transformations. We split the training partition of MNIST in half, and transform all and only the examples in the first half; this ensures that there is no matching ground truth in the dataset, and that learning is unsupervised. As a preprocessing step, the digits are zero-padded so that they have size 32×32 pixels, and the pixel intensities are scaled between 0 and 1. This is done even before any mechanism is applied. We use neural networks for both the experts and the selection mechanism, and employ an adversarial training scheme.

Each expert E_i can be seen as a generator from a GAN conditioned on an input image rather than (as usually) a noise vector. A discriminator D provides gradients for training the experts and acts also as a selection mechanism c : only the expert whose output obtains the higher score from D wins the example, and is trained on it to maximize the output of D . We describe the exact algorithm used to train the networks in these experiments in Al-

Algorithm 1 Learning independent mechanisms using competition of experts and adversarial training

Precondition: X : data sampled from P ; X' : data sampled from \mathcal{D}_Q ; D discriminator; N' : number of experts; T : maximum number of iterations;
 every step can be run in parallel across experts

```

1  $\{E_i \leftarrow \text{TrainAsIdentityOn}(X')\}_{j=1}^{N'}$   $\triangleright$  Init experts as  $\sim$  identity
2 for  $t \leftarrow 1$  to  $T$  do
3    $x, x' \leftarrow \text{Sample}(X), \text{Sample}(X')$   $\triangleright$  Sample minibatches
4    $\{c_j \leftarrow D(E_j(x'))\}_{j=1}^{N'}$   $\triangleright$  Scores from  $D$  for each expert
5    $\theta_D^{t+1} \leftarrow \text{Adam}\left(\theta_D^t, \nabla \log D(x) \right.$   $\triangleright$  Update  $D$ 
      $\left. + \nabla(1/N' \sum_{j=1}^{N'} \log(1 - c_j))\right)$ 
6    $\{\theta_{E_j}^{t+1} \leftarrow \text{Adam}(\theta_{E_j}^t, \nabla \max_{j \in \{1, \dots, N'\}} \log(c_j))\}_{j=1}^{N'}$   $\triangleright$  Update experts
```

gorithm 1. The discriminator is trained to maximize the following cross-entropy loss:

$$\max_{\theta_D} \left(\mathbb{E}_{x \sim P} \log(D_{\theta_D}(x)) + \frac{1}{N'} \sum_{j=1}^{N'} \mathbb{E}_{x' \sim Q} (\log(1 - D_{\theta_D}(E_{\theta_j}(x')))) \right) \quad (3)$$

For simplicity, we assume for the rest of this section that the number of experts N' equals the number of true mechanisms N . Results where $N \neq N'$ are relegated to Appendix A.1.1.

NEURAL NETS DETAILS. Each expert is a CNN with five convolutional layers, 32 filters per layer of size 3×3 , ELU (Clever et al., 2016) as activation function, batch normalization (Ioffe and Szegedy, 2015), and zero padding. The discriminator is also a

CNN, with average pooling every two convolutional layers, growing number of filters, and a fully connected layer with 1024 neurons as last hidden layer. Both networks are trained using Adam as optimizer (Kingma and Ba, 2015), with the default hyperparameters.³

Unless specified otherwise, after a random weight initialization we first train the experts to approximate the identity mapping on our data, by pretraining them on predicting identical input-output pairs randomly selected from the transformed dataset. This makes the experts start from similar grounds, and we found that this improved the speed and robustness of convergence. We will refer to this as *approximate identity initialization* for the rest of the chapter.

A minibatch of 32 transformed MNIST digits, each transformed by a randomly chosen mechanism, is fed to all experts E_i . The outputs are fed to the discriminator D , which computes a score for each of them. For each example the cross entropy loss in Equation equation 3 and the resulting gradients are computed only for the output of the highest scoring expert, and they are used to update both the discriminator (when 0 is the target in the cross entropy) and the winning expert (when using 1 as the target). In order to further support the winning expert, we punish the losing experts by training the discriminator against their outputs as well. Then, a minibatch of canonical MNIST digit is used in order to update the discriminator with ‘real’ data. We refer to the above procedure as one *iteration*.

We ran the experiments 10 times with different random seeds for the initializations. Each experiment is run for 2000 iterations.

³ For the exact experimental parameters and architectures see the Appendix A.2 or the PyTorch implementation at <https://drive.google.com/drive/folders/1cEUUpQbCctoc7locRc81RJwjK3fKRPmIw?usp=sharing>.

3	1	9	1	4	9	3	3	0	9	4	9	1	9	6	4
3	1	9	1	4	9	3	3	0	9	4	9	1	9	6	4

Figure 3: The top row contains 16 random inputs to the networks, and the bottom row the corresponding outputs from the highest scoring experts against the discriminator after 1000 iterations.

2.5 RESULTS

The experts correctly specialized on inverting exactly one mechanism each in 7 out of the 10 runs; in the remaining 3 runs the results were only slightly suboptimal: one expert specialized on two tasks, one expert did not specialize on any, and the remaining experts still specialized on one task each, thus still covering all the existing tasks. In Figure 3 we show a randomly selected batch of inputs and corresponding outputs from the model. Each independent mechanism was inverted by a different expert.

We first discuss our three main findings, and then move on to additional experiments.

- The experts specialize w.r.t. c .** In Figure 4, we plot the scores assigned by the discriminator for each expert on each task in a typical successful run. Each expert is represented with the same color and linestyle across all tasks. The figure shows that after an initial phase of heavy competition, the experts exhibit the desired behavior and obtain a high score on D on one mechanism each. Note how the green expert tries to learn two similar tasks until iteration 750 (left and left-down translation), at which point the red expert takes over one of the tasks. Subsequently, both specialize rapidly. Figure 5 provides further evidence, by visualizing that the assignments of data points to experts induced by c are meaningful. We report the proportion of examples from each task assigned to each expert at the beginning and at the end of training: at first,

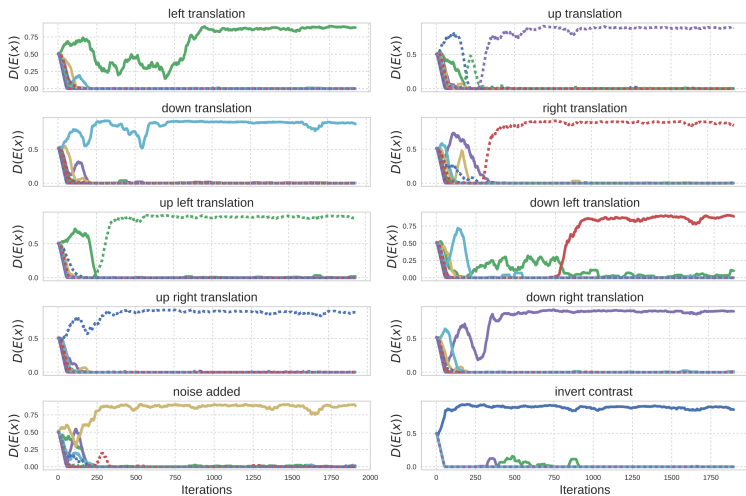
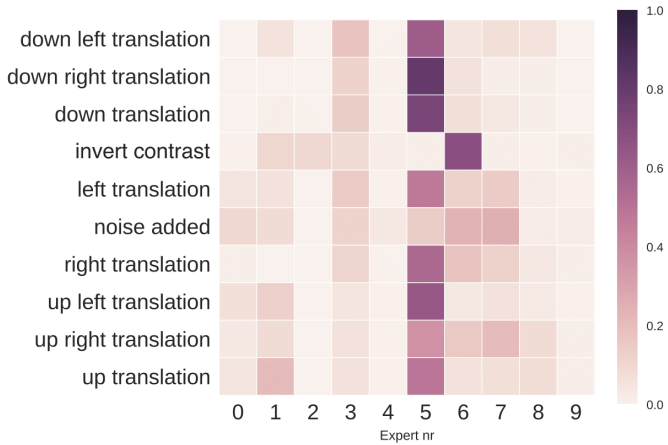


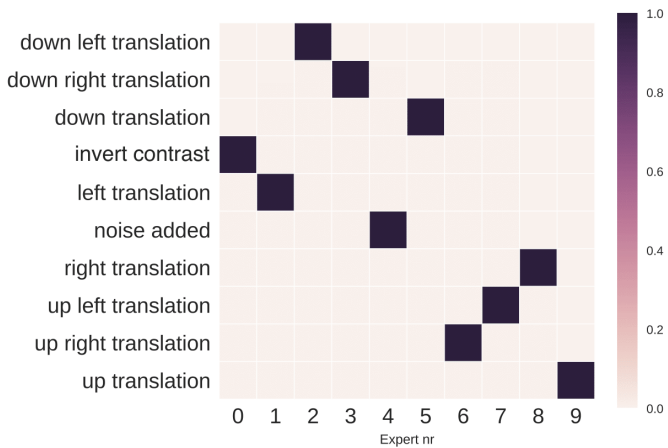
Figure 4: Experts’ performance, measured by discriminator scores. Each line color/style represents one expert. For each of ten different mechanisms (top left to bottom right), the experts are being fed transformed digits. Each expert learns to specialize on a different mechanism, as shown by the score approaching 1. Each curve is smoothed with a moving average of 50 iterations.

the assignment of experts to tasks by the discriminator is almost uniform; by the end of the training, each expert wins almost all examples coming from one transformation, and no others.

2. The transformed outputs improve a classifier. In order to test if the committee of experts can recover a good approximation of the original digits, we test the output of our experts against a pretrained standard MNIST classifier. For this, we use the test partition of the data. We compare the accuracy for three inputs: *a)* the test digits transformed by the mechanisms, *b)* the transformed digits after being processed by the highest scoring experts (which tries to invert the mechanisms), *c)* the original test digits.



(a) Before training



(b) After 1000 iterations

Figure 5: The proportion of data won by each expert for each transformation on the digits from the test set.

The latter can be seen as an upper bound to the accuracy that can be achieved.

As shown by the two dashed horizontal lines in Figure 6, the transformed test digits achieve a 40% accuracy when tested directly on the classifier, while the untransformed digits would achieve $\approx 99\%$ accuracy. The accuracy for the output digits starts at 40% — due to the identity initialization of the experts — but it subsequently quickly approaches the performance on the original digits as it is trained. Note that after about 600 iterations, i.e., once the networks have seen about one third of the whole dataset *once*, the accuracy has almost reached the upper bound.

3. The experts learn mechanisms that generalize. Given that towards the end of training, each expert E_i is updated only on data points from Q_i , one could imagine that they will not perform well on data points from other distributions. In fact this is not the case. Not only do all experts E_i generalize to all other transformed distributions Q_j , but also to different datasets all together. To show this, we use the Omniglot dataset of letters from different alphabets (Lake et al., 2015) and rescale them to 46×46 pixels (instead of 32×32 of MNIST, which is not an issue since the experts are fully convolutional). We transform a random sample with all mechanisms M_i and test each on all experts E_i , which have only been trained on MNIST. As shown in Figure 7, each network consistently applies the same transformation also on inputs outside of the domain they have specialized on. They indeed learn a *mechanism* that is independent of the input.

Having made our main points, we continue with a few more observations.

The learned inverse mechanisms can be combined. We test whether the trained experts could in principle be used to undo several transformations applied at once, even though the training set consisted only of images transformed by a single mechanism. For simplicity, we assume we know which transformations were

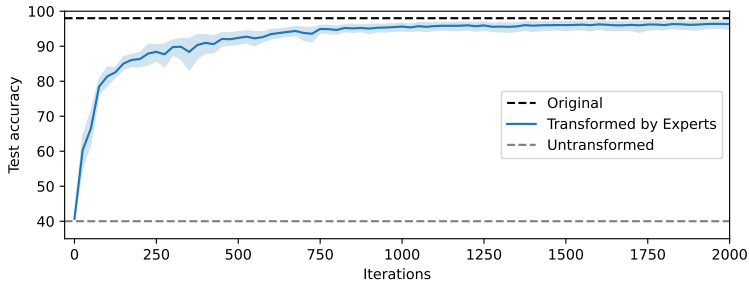


Figure 6: Accuracy of a pretrained CNN MNIST classifier on transformed test digits \mathcal{D}_Q , on the same digits after going through our model, and on the original digits. Our system manages to invert the transformations, with the classifier accuracy quickly approaching the optimum. Note that 600 iterations correspond to having seen about a third of the dataset.

used. In Figure 8, we test on Omniglot letters transformed with three consecutive transformations (noise, up left translation, contrast inversion) by applying the corresponding experts previously trained on MNIST, and correctly recover the original letters.

EFFECT OF THE APPROX. IDENTITY INITIALIZATION. For the same experiments but without the approximate identity initialization, several experts fail to specialize. Out of 10 new runs with random initialization, only one experiment had arguably good results, with eight experts specializing on one task each, one on two tasks, and the last one on none. The performance was worse in the remaining runs. The problem was not that the algorithm takes longer to converge following a random initialization, as with an additional experiment for 10 000 iterations the results did not improve. Instead, the random initialization can lead to one expert winning examples from many tasks at the beginning of training, in which case it is hard for the others to catch up.













Inputs		ण	स	उ	ऌ	ल	त	७	८	९
Exp0		ण	स	उ		ल	त	७	८	९
Exp1		ण	स	उ	ऌ	ल	त	७	८	९
Exp2		ण	स	उ	ऌ	ल	त	७	८	९
Exp3		ण	स	उ	ऌ	ल	त	७	८	९
Exp4		ण	स	उ	ऌ	ल	त	७	८	९
Exp5		ण	स	उ	ऌ	ल	त	७	८	९
Exp6		ण	स	उ	ऌ	ल	त	७	८	९
Exp7		ण	स	उ	ऌ	ल	त	७	८	९
Exp8		ण	स	उ	ऌ	ल	त	७	८	९
Exp9		ण	स	उ	ऌ	ल	त	७	८	९

Figure 7: Each column shows how each expert transforms the input presented on top. We arrange the tasks such that the diagonal contains the highest scoring expert for the input given at the top of the column. The experts have learned the inverse mechanisms, consistently applying them to previously unseen symbols.

A SIMPLE SINGLE-NET BASELINE. Training a single network instead of a committee of experts makes the problem more difficult to solve. Using identical training settings, we trained a single network once with 32, once with 64, and once with 128 filters per layer, and none of them managed to correctly learn more than one inverse mechanism.⁴ Note that a single network with 128 filters per layer has about twice as many parameters overall as the committee of 10 experts with 32 filters per layer each. We also tried *a*) random initialization instead of the approximate identity,

⁴ Specifically, the network performs well on the contrast inversion task, and poorly on all others.



Figure 8: First row: input Omniglot letters that were transformed with noise, contrast inversion and translation up left. Second to fourth row: application of denoising, contrast inverting and right down translating experts. Last row: ground truth. Although the experts were not trained on a combination of mechanisms nor on Omniglot letters, they can be used to recover the original digits.

b) reducing the learning rate of the discriminator by a factor of 10, and *c)* increasing the receptive field by adding two pooling and two upsampling layers, without any improvement. While we do not exclude that careful hyperparameter tuning may enable a single net to learn multiple mechanisms, it certainly was not straightforward in our experiments.

SPECIALIZATION OCCURS ALSO WITH HIGHER CAPACITY EXPERTS. While in principle with infinite capacity and data, a single expert could solve all tasks simultaneously, in practice limited resources and the proposed training procedure favor specialization in independent modules. Increasing the size of the experts

from 32 filters per layer to 64 or 128 filters,⁵ or enlarging the overall receptive field by using two pooling and two upsampling layers, still resulted in good specialization of the experts, with no more than two experts specializing on two tasks at once.

FEWER EXAMPLES FROM THE CANONICAL DISTRIBUTION. In some applications, we might only have a small sample from the original distribution. Interestingly, if we reduce the number of examples from the original distribution from 30 000 down to 64, we find that all experts still specialize and recover good approximations of the inverse mechanisms, using the exact same training protocol. Although the output digits turn out less clean and sharp, we still achieve 96% accuracy on the pretrained MNIST classifier.

2.6 CONCLUSIONS

We have developed a method to identify and learn a set of independent causal mechanisms. Here these are *inverse* mechanisms, but an extension to forward mechanisms appears feasible and worthwhile. We reported promising results in experiments using image transformations; future work could study more complex settings and diverse domains. The method does not explicitly minimize a measure of dependence of mechanisms, but works if the data generating process contains independent mechanisms in the first place: As the different tasks (mechanisms) do not contain information about each other, improving on one of them does not improve performance on another, which is exactly what encourages specialization.

A natural extension of our work is to consider independent mechanisms that *simultaneously* affect the data (e.g. lighting and position in a portrait), and to allow multiple passes through our

⁵ Equivalent to an increase of parameters from $\sim 27\text{K}$ to $\sim 110\text{K}$ or $\sim 440\text{K}$ parameters respectively.

committee of experts to identify local mechanisms (akin to Lie derivatives) from more complex datasets — for instance, using recurrent neural networks that allow the application of multiple mechanisms by iteration. With many experts, the computational cost (or parallel processing) might become unnecessarily high. This could be mitigated by hybrid approaches incorporating gated mixture of experts or a hierarchical selection of competing experts.

We believe our work constitutes a promising connection between causal modeling and deep learning. As discussed in the introduction, causality has a lot to offer for crucial machine learning problems such as transfer or compositional modeling. Our systems sheds light on these issues. Independent modules as sub-components could be learned using multiple domains or tasks, added subsequently, and transferred to other problems. This may constitute a step towards causally motivated life-long learning.

LEARNING EXPLANATIONS THAT ARE HARD TO VARY

The quest for good explanations does the job: inventing falsehoods is easy, and therefore they are easy to vary once found; discovering good explanations is hard, but the harder they are to find, the harder they are to vary once found.

David Deutsch

CHAPTER ABSTRACT In this chapter, we investigate the principle that good explanations are hard to vary in the context of deep learning. We show that averaging gradients across examples – akin to a logical OR of patterns – can favor memorization and ‘patchwork’ solutions that sew together different strategies, instead of identifying invariances. To inspect this, we first formalize a notion of consistency for minima of the loss surface, which measures to what extent a minimum appears only when examples are pooled. We then propose and experimentally validate a simple alternative algorithm based on a logical AND, that focuses on invariances and prevents memorization in a set of real-world tasks. Finally, using a synthetic dataset with a clear distinction between invariant and spurious mechanisms, we dissect learning signals and compare this approach to well-established regularizers.

This chapter is based on the paper “*Learning explanations that are hard to vary*”, Giambattista Parascandolo*, Alexander Neitz*, Antonio Orvieto, Luigi Gresele, Bernhard Schölkopf (Parascandolo et al., 2021a)

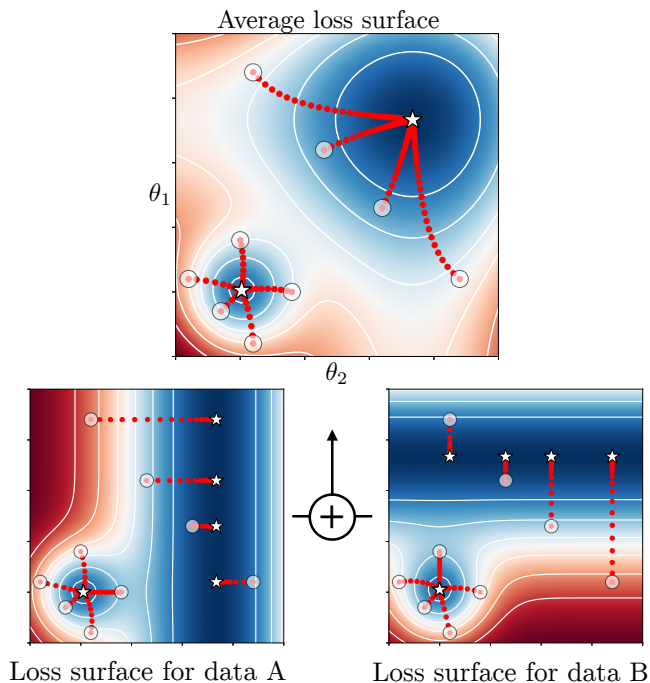


Figure 9: Loss landscapes of a two-parameter model. Averaging gradients forgoes information that can identify patterns shared across different environments.

3.1 INTRODUCTION

Consider the top of Figure 9, which shows a view from above of the loss surface obtained as we vary a two dimensional parameter vector $\theta = (\theta_1, \theta_2)$, for a fictional dataset containing two observations x_A and x_B . Note the two global minima on the top-right and bottom-left. Depending on the initial values of θ — marked as white circles — gradient descent converges to one of the two

minima. Judging solely by the value of the loss function, which is zero in both cases, the two minima look equally good.

However, looking at the loss surfaces for x_A and x_B separately, as shown below, a crucial difference between those two minima appears: Starting from the same initial parameter configurations and following the gradient of the loss, $\nabla_{\theta} \mathcal{L}(\theta, x_i)$, the probability of finding the same minimum on the top-right in either case is zero. In contrast, the minimum in the lower-left corner has a significant overlap across the two loss surfaces, so gradient descent can converge to it even if training on x_A (or x_B) only. Note that after averaging there is no way to tell what the two loss surfaces looked like: *Are we destroying information that is potentially important?*

In this chapter, we argue that the answer is yes. In particular, we hypothesize that if the goal is to find *invariant mechanisms* in the data, these can be identified by finding explanations (e.g. model parameters) that are hard to vary across examples. A notion of invariance implies something that stays the same, as something else changes. We assume that data comes from different *environments*: An invariant mechanism is shared across all, generalizes out of distribution (o.o.d.), but might be hard to model; each environment also has spurious explanations that are easy to spot ('shortcuts'), but do not generalize o.o.d. From the point of view of causal modeling, such invariant mechanisms can be interpreted as conditional distributions of the targets given causal features of the inputs; invariance of such conditionals is expected if they represent *causal mechanisms*, that is — stable properties of the physical world (see e.g. Hoover, 1990). Generalizing o.o.d. means therefore that the predictor should perform equally well on data coming from different settings, as long as they share the causal mechanisms.

We formalize a notion of *consistency*, which characterizes to what extent a minimum of the loss surface appears *only* when

data from different environments are pooled. Minima with low consistency are ‘patchwork’ solutions, which (we hypothesize) sew together different strategies and should not be expected to generalize to new environments. An intuitive description of this principle was proposed by physicist David Deutsch: “*good explanations are hard to vary*” (Deutsch, 2011).

Using the notion of consistency, we define *Invariant Learning Consistency* (ILC), a measure of the expected consistency of the solution found by a learning algorithm on a given hypothesis class. The ILC can be improved by changing the hypothesis class or the learning algorithm, and in the last part of the chapter we focus on the latter. We then analyse why current practices in deep learning provide little incentive for networks to learn invariances, and show that standard training is instead set up with the explicit objective of greedily maximizing speed of learning, i.e., progress on the training loss. When learning “as fast as possible” is not the main objective, we show we can trade-off some “learning speed” for prioritizing learning the invariances. A practical instantiation of ILC leads to o.o.d. generalization on a challenging synthetic task where several established regularizers fail to generalize; moreover, following the memorization task from Zhang et al., 2017, ILC prevents convergence on CIFAR-10 with random labels, as no shared mechanism is present, and similarly when a portion of training labels is incorrect. Lastly, we set up a behavioural cloning task based on the game CoinRun (Cobbe et al., 2019), and observe better generalization on new unseen levels.

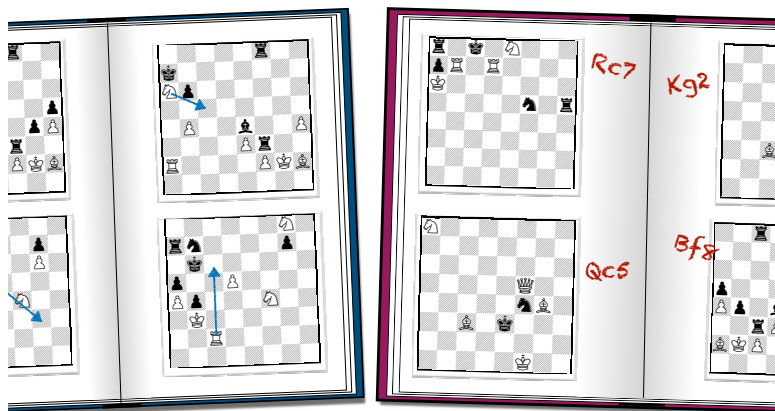


Figure 10: Two second-hand books of chess puzzles.

AN EXAMPLE. Take the two second-hand books of chess puzzles in 10. We can learn the two independent shortcuts (blue arrows for the left book OR hand-written solutions on the right), or actually learn to play chess (the invariant mechanism). While both strategies solve other problems from the same books (i.i.d.), only the latter generalises to new chess puzzle books (o.o.d.). How to distinguish the two? We would not have learned about the red arrows had we trained on the book on the right, and vice versa with the hand-written notes.

3.2 EXPLANATIONS THAT ARE HARD TO VARY

We consider datasets $\{\mathcal{D}^e\}_{e \in \mathcal{E}}$, with $|\mathcal{E}| = d$, and $\mathcal{D}^e = (x_i^e, y_i^e)$, $i_e = 1, \dots, n^e$. Here $x_i^e \in \mathcal{X} \subseteq \mathbb{R}^m$ is the vector containing the observed inputs, and $y_i^e \in \mathcal{Y} \subseteq \mathbb{R}^p$ the targets. The superscript $e \in \mathcal{E}$ indexes some aspect of the data collection process, and can

be interpreted as an environment label. Our objective is to infer a function $f : \mathcal{X} \rightarrow \mathcal{Y}$ — which we call *mechanism* — assigning a target y_i^e to each input x_i^e ; as explained in the introduction, we assume that such function is shared across all environments. For estimation purposes, f may be parametrized by a neural network with continuous activations; for weights $\theta \in \Theta \subseteq \mathbb{R}^n$, we denote the neural network output at $x \in \mathcal{X}$ as $f_\theta(x)$.

GRADIENT-BASED OPTIMIZATION. To find a good model f_θ , standard optimizers rely on gradients from a *pooled* loss function $\mathcal{L} : \mathbb{R}^n \rightarrow \mathbb{R}$. This function measures the *average* performance of the neural network when predicting data labels, across all environments: $\mathcal{L}(\theta) := \frac{1}{|\mathcal{E}|} \sum_{e \in \mathcal{E}} \mathcal{L}_e(\theta)$, with $\mathcal{L}_e(\theta) := \frac{1}{|\mathcal{D}^e|} \sum_{(x_i^e, y_i^e) \in \mathcal{D}^e} \ell(f(x_i^e; \theta), y_i^e)$; where $\ell : \mathbb{R}^p \times \mathbb{R}^p \rightarrow [0, +\infty)$ is usually chosen to be the L_2 loss or the cross-entropy loss. The parameter updates according to gradient descent (GD) are given by $\theta_{\text{GD}}^{k+1} = \theta_{\text{GD}}^k - \eta \nabla \mathcal{L}(\theta_{\text{GD}}^k)$, where $\eta > 0$ is the learning rate. Under some standard assumptions (Lee et al., 2016a), $(\theta_{\text{GD}}^k)_{k \geq 0}$ converges to a local minimizer of \mathcal{L} , with probability one.

WHEN DO WE *not* LEARN INVARIANCES? We start by describing what might prevent learning invariances in standard gradient-based optimization.

(i) *Training stops once the loss is low enough.* If optimization learned spurious patterns by the time it converged, invariances will not be learned anymore. This depends on the rate at which different patterns are learned. The rates at which invariant patterns emerge (and vice-versa, the spurious patterns do not) can be improved by e.g.: (a) careful architecture design, e.g. as done by hardcoding spatial equivariance in convolutional networks; (b) fine-tuning models pre-trained on large amounts of data, where strong features already emerged and can be readily selected.

(ii) *Learning signals: everything looks relevant for a dataset of size 1.* Due to the summation in the definition of the pooled loss \mathcal{L} , gradients for each example are computed independently. Informally, each signal is identical to the one for an equivalent dataset of size 1, where every pattern appears relevant to the task. To find invariant patterns across examples, if we compute our training signals on each of them independently, we have to rely on the way these are aggregated.¹

(iii) *Aggregating gradients: averaging maximizes learning speed.* The default method to pool gradients is the *arithmetic mean*. GD applied to \mathcal{L} is designed to minimize the pooled loss *by prioritizing descent speed*.² Indeed, a step of GD is equivalent to finding a tight³ quadratic upper bound $\hat{\mathcal{L}}$ to \mathcal{L} , and then jumping to the minimizer of this approximation (Nocedal and Wright, 2006). While speed is often desirable, by construction GD ignores one potentially crucial piece of information: The gradient $\nabla \mathcal{L}$ is the result of averaging signals $\nabla \mathcal{L}_e$, which correspond to the patterns visible from each environment at this stage of optimization. In other words, GD with average gradients greedily maximizes for learning speed, but in some situations we would like to trade some convergence speed for invariance. For instance, instead of performing an arithmetic mean between gradients (logical OR), we might want to look towards a logical AND, which can be characterized as a *geometric mean*. Fig. 9 shows how a sum can be seen as a logical OR: the two orthogonal gradients from data A

-
- 1 After computing the gradients for a dataset of $n - 1$ examples, if an n -th example appeared, we would just compute one more vector of gradients and add it to the sum. A Gaussian Process (Rasmussen, 2003) for example would require recomputing the entire solution from scratch, as all interactions are considered.
 - 2 The same reasoning holds for SGD in the finite-sum optimization case $\mathcal{L} = \frac{1}{m} \sum_{i=1}^m \mathcal{L}_i$, where gradients from a mini-batch are seen as unbiased estimators of gradients from the pooled loss. (Bottou et al., 2018).
 - 3 Assume that \mathcal{L} has L -Lipschitz gradients (i.e. curvature bounded from above by L). Then, at any point $\hat{\theta}$, we can construct the upper bound $\hat{\mathcal{L}}_{\hat{\theta}}(\theta) = \mathcal{L}(\hat{\theta}) + \nabla \mathcal{L}(\hat{\theta})^\top (\theta - \hat{\theta}) + L \|\theta - \hat{\theta}\|^2 / 2$.

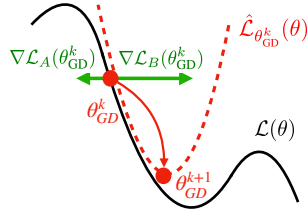


Figure 11: Inconsistency in gradient directions.

and data B at $(0.5, 0.5)$ point to different directions, yet both are kept in the combined gradient.⁴ In Sec. 3.2.3 we elaborate on this idea and on implementing a logical AND between gradients. Before presenting this discussion, we take some time to better motivate the need for invariant learning consistency and to construct a precise mathematical definition of consistency.

3.2.1 Formal definition of ILC

Let $\Theta_{\mathcal{A}}^*$ be the set of convergence points of algorithm \mathcal{A} when trained using all environments (pooled data): that is, $\Theta_{\mathcal{A}}^* = \{\theta^* \in \Theta \mid \exists \theta^0 \in \mathbb{R}^n \text{ s.t. } \mathcal{A}_{\infty}(\theta^0, \mathcal{E}) = \theta^*\}$. For instance, if \mathcal{A} is gradient descent, the result of Lee et al., 2016a implies that $\Theta_{\mathcal{A}}^*$ is the set of local minimizers of the pooled loss \mathcal{L} . To each $\theta^* \in \Theta_{\mathcal{A}}^*$, we want to associate a consistency score, quantifying the concept “good θ^* are hard to vary”. In other words, we would like the score to capture the consistency of the loss landscape around θ^* across the different environments. For example, in Fig. 9 the loss landscape near the bottom-left minimizer is consistent across environments, while the top-right minimizer is not.

⁴ Loosely speaking, a sum is large if any of the summands is large, a product is large if all factors are large.

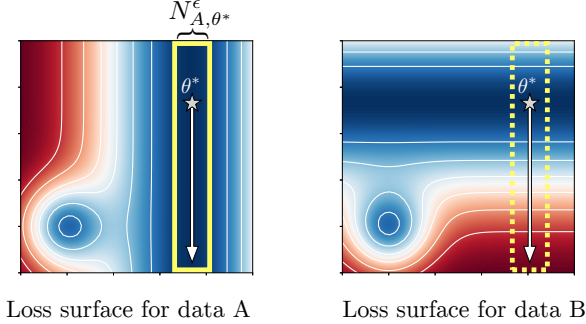


Figure 12: The minimum on θ^* is easy to vary without affecting the loss on environment A while increasing the loss significantly on environment B. As such, it is inconsistent under equation 4

Let us characterize the landscape around θ^* from the perspective of a fixed environment $e \in \mathcal{E}$. We define the set N_{e, θ^*}^ϵ to be the largest path-connected region of space containing both θ^* and the set $\{\theta \in \Theta \text{ s.t. } |\mathcal{L}_e(\theta) - \mathcal{L}_e(\theta^*)| \leq \epsilon\}$, with $\epsilon > 0$. In other words, if $\theta \in N_{e, \theta^*}^\epsilon$ then there exist a path-connected region in parameter space including θ^* and θ where each parameter also is in N_{e, θ^*}^ϵ and its loss on environment e is comparable. From the perspective of environment e , all these points are equivalent to θ^* . We would like to evaluate the elements of this set with respect to a different environment $e' \neq e$ (see Figure 12). We will say that e' is consistent with e in θ^* if $\max_{\theta \in N_{e, \theta^*}^\epsilon} |\mathcal{L}_{e'}(\theta) - \mathcal{L}_e(\theta)|$ is small. Repeating this reasoning for all environment pairs, we arrive at the following *inconsistency score*:

$$\mathcal{I}^\epsilon(\theta^*) := \max_{(e, e') \in \mathcal{E}^2} \max_{\theta \in N_{e, \theta^*}^\epsilon} |\mathcal{L}_{e'}(\theta) - \mathcal{L}_e(\theta)|. \tag{4}$$

This consistency is our formalization of the principle “*good explanations are hard to vary*”. Finally, we can write down an invariant learning consistency score for \mathcal{A} :

$$\text{ILC}(\mathcal{A}, p_{\theta^0}) := -\mathbb{E}_{\theta^0 \sim p(\theta^0)} \left[\mathcal{I}^\epsilon(\mathcal{A}_\infty(\theta^0), \mathcal{E}) \right]. \quad (5)$$

That is, the learning consistency of an algorithm measures the expected consistency across environments of the minimizer it converges to on the pooled data.

EXAMPLE: LOW CONSISTENCY OF A CLASSIC PATCHWORK SOLUTION. One-hidden-layer networks with sigmoid activations and enough neurons can approximate any function $f^* : [0, 1] \rightarrow \mathbb{R}$ (Cybenko, 1989). In appendix B.1.1 we show how the construction used to obtain the weights leads to a maximally inconsistent solution according to $\mathcal{I}^\epsilon(\theta^*)$, which would not be expected to generalize o.o.d.

3.2.2 ILC as a logical AND between landscapes

Here we draw a connection between our definition of inconsistency and the local geometric properties of the loss landscapes. For the sake of clarity, we consider two environments (A and B) and assume θ^* to be a local minimizer (with zero loss) for both environments. Using a Taylor approximation⁵, we get $\mathcal{L}(\theta) \approx \frac{1}{2}(\theta - \theta^*)^\top H_{A+B}(\theta - \theta^*)$ for $\|\theta - \theta^*\| \approx 0$, where $H_{A+B} = (H_A + H_B) / 2$ is the *arithmetic mean* of the Hessians $H_A := \nabla^2 \mathcal{L}_A(\theta^*)$ and $H_B := \nabla^2 \mathcal{L}_B(\theta^*)$. H_{A+B} does not capture the possibly conflicting geometries of landscape A or B : It performs a “logical OR” on the dominant eigendirections. In contrast, the *geometric mean*, or

⁵ This provides a useful simplified perspective. Indeed, this *quadratic model* is heavily used in the optimization community (see e.g. Jastrzbski et al., 2017; Zhang et al., 2019a; Mandt et al., 2017.)

Karcher mean, $H_{A \wedge B}$ (Ando et al., 2004) is affected by the inconsistencies between landscapes: It performs a “logical AND”. In appendix B.1.2, we give a formal definition of $H_{A \wedge B}$, and show that for diagonal Hessians, $\mathcal{I}^e(\theta^*) \leq 2\epsilon \left(\frac{\det(H_{A+B})}{\det(H_{A \wedge B})} \right)^2$. As for the geometric mean of positive numbers, $0 \leq \det(H_{A \wedge B}) \leq \det(H_{A+B})$; thus, inconsistency is lowest when *shapes* of A and B are similar – exactly as in the bottom-left minimizer of Fig. 9.

FROM HESSIANS TO GRADIENTS. We just saw that the consistency of θ^* is linked to the geometric mean of the Hessians $\{H_e(\theta^*)\}_{e \in \mathcal{E}}$. Under the simplifying assumption that each H_e is diagonal⁶ and all eigenvalues λ_i^e are positive, their geometric mean is $H^\wedge := \text{diag}((\prod_{e \in \mathcal{E}} \lambda_1^e)^{1/|\mathcal{E}|}, \dots, (\prod_{e \in \mathcal{E}} \lambda_n^e)^{1/|\mathcal{E}|})$. The curvature of the corresponding loss in the i -th eigendirection depends on how consistent the curvatures of each environment are in that direction. Consider now optimizing from a point θ^k ; gradient descent reads $\theta^{k+1} = \theta^k - \eta H^+(\theta^k - \theta^*)$, where $H^+ := \text{diag}(\frac{1}{|\mathcal{E}|} \sum_{e \in \mathcal{E}} \lambda_1^e, \dots, \frac{1}{|\mathcal{E}|} \sum_{e \in \mathcal{E}} \lambda_n^e)$. For η small enough⁷, we have $|\theta_i^{k+1} - \theta_i^*| = (1 - \eta \frac{1}{|\mathcal{E}|} \sum_{e \in \mathcal{E}} \lambda_i^e) |\theta_i^k - \theta_i^*|$. As noted, this choice maximises the speed of convergence to θ^* , but does not take into account whether this minimizer is consistent. We can reduce the speed of convergence on directions where landscapes have different curvatures – which would lead to a high inconsistency – by following the gradients from the geometric mean of the landscapes, as opposed to the arithmetic mean. I.e, we substitute the full gradient $\nabla \mathcal{L}(\theta) = H^+(\theta^k - \theta^*)$ with $\nabla \mathcal{L}^\wedge(\theta) = H^\wedge(\theta^k - \theta^*)$. Also, we have that⁸ $\nabla \mathcal{L}^\wedge(\theta) = (\prod_{e \in \mathcal{E}} \nabla \mathcal{L}_e(\theta))^{1/|\mathcal{E}|}$: to reduce the speed

6 It was shown in (Becker and Le Cun, 1988) and recently in (Adolphs et al., 2019; Singh and Alistarh, 2020) that neural networks have a strong diagonal dominance of the Hessian matrix at the end of training.

7 Smaller than $1/\lambda_{\max}$, λ_{\max} is the maximum eigenvalue of Hessians from different environments,

8 This holds if $\theta - \theta^*$ is positive, otherwise $\nabla \mathcal{L}^\wedge(\theta) = -(\prod_{e \in \mathcal{E}} |\nabla \mathcal{L}_e(\theta)|)^{1/|\mathcal{E}|}$.

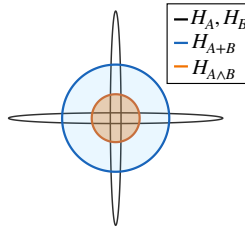


Figure 13: Contour lines $\theta^\top H^{-1} \theta = 1$ for $H_A = \text{diag}(0.05, 1)$ and $H_B = \text{diag}(1, 0.05)$. $H_{A \wedge B}$ retains the original volumes, while for H_{A+B} it is $5\times$ bigger. This magnification shows inconsistency of A and B .

of convergence in directions with inconsistency, we can take the element-wise geometric mean of gradients from different environments (see also Fig. 35 in the appendix).

3.2.3 Masking gradients with a logical AND

The element-wise geometric mean of gradients, instead of the arithmetic mean, increases consistency in the convex quadratic case. However, there are a few practical limitations:

- (i) The geometric mean is only defined when all the signs are consistent. It is still to be defined how sign inconsistencies, which can occur in non-convex settings, should be dealt with.
- (ii) It provides little flexibility for ‘partial’ agreement: Even a single zero gradient component in one environment stops optimization in that direction.
- (iii) For numerical stability, it needs to be computed in log domain (more computationally expensive).
- (iv) Adaptive step-size schemes (e.g. Adam (Kingma and Ba, 2015)) rescale the signal component-wise for local curvature adapta-

tion. The exact magnitude of the geometric mean would be ignored and most of the difference from arithmetic averaging will come from the zero-ed components.

(i) can be overcome by treating different signs as zeros, resulting in a geometric mean of 0 if there is any sign disagreement across environments for a gradient component. For (ii) we can allow for *some* disagreement (with a hyperparameter), by not masking out if there is a large percentage of environments with gradients in that direction. (iii) and (iv) can be addressed together: Since the final magnitude will be rescaled except for masked components, i.e. where the geometric mean is 0, we can use the average gradients (fast to compute) and mask out the components based on the sign agreement (computable avoiding the log domain).

THE AND-MASK. We translate the reasoning we just presented to a practical algorithm that we will refer to as the *AND-mask*. In its most simple implementation, we zero out those gradient components with respect to weights that have *inconsistent signs* across environments. Formally, the masked gradients at iteration k are $m_t(\theta^k) \odot \nabla \mathcal{L}(\theta^k)$, where $m_t(\theta^k)$ vanishes for any component where there are less than $t \in \{d/2, d/2 + 1, \dots, d\}$ agreeing gradient signs across environments (d is the number of environments in the batch), and is equal to one otherwise. For convenience, our implementation of the AND-mask uses a *threshold* $\tau \in [0, 1]$ as hyper-parameter instead of t , such that $t = \frac{d}{2}(\tau + 1)$. Mathematically, for every component $[m_\tau]_j$ of m_τ , $[m_\tau]_j = \mathbb{1}[\tau d \leq |\sum_e \text{sign}([\nabla \mathcal{L}_e]_j)|]$.

Computing the AND-mask has the same time and space complexity of standard gradient descent, i.e., linear in the number of examples that we average. Due to its simplicity and computational efficiency, this is the algorithm that we will use in the experiment section. As a first result, we show that following the AND-

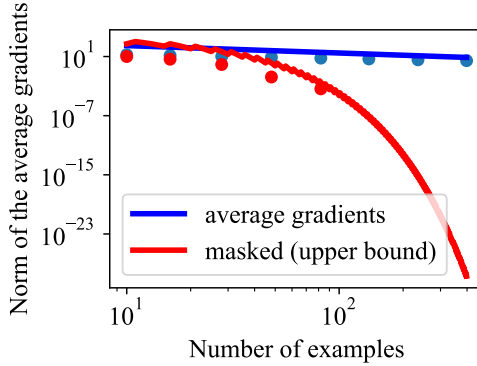


Figure 14: Magnitude of gradient (average or masked) on random data ($|\theta| = 3000$, $t = 0.8d$).

masked gradient leads to convergence in the directions made visible by the AND-mask. The proof is presented in appendix B.1.3.

Proposition 1. Let \mathcal{L} have L -Lipschitz gradients and consider a learning rate $\eta \leq 1/L$. After k iterations, AND-masked GD visits at least once a point θ where $\|m_t(\theta) \odot \nabla \mathcal{L}(\theta)\|^2 \leq \mathcal{O}(1/k)$.

BEHAVIOUR IN THE FACE OF RANDOMNESS. Here we put the AND mask through a theoretical test: For gradients coming from different environments that are inconsistent (or even random), how fast does the AND mask reduce the magnitude of the step taken in parameter space, compared to standard GD? *In case of inconsistency, the AND mask should quickly make the gradient steps more conservative.*

To assess this property, we consider a fixed set of n parameters θ and gradients $\nabla \mathcal{L}_e$ drawn independently from a multivariate Gaussian with zero mean and unit covariance.

Proposition 2. Consider the setting we just outlined, with $\mathcal{L} = (1/d) \sum_{e=1}^d \mathcal{L}_e$. While $\mathbb{E} \|\nabla \mathcal{L}(\theta)\|^2 = \mathcal{O}(n/d)$, we have that $\forall t \in \{d/2 + 1, \dots, d\}, \exists c \in (1, 2]$ such that $\mathbb{E} \|m_t(\theta) \odot \nabla \mathcal{L}(\theta)\|^2 \leq \mathcal{O}(n/c^d)$.

The proof is presented in Appendix B.1.4, and an illustration with numerical verification in Fig. 14 (the magnitudes of masked gradients (\bullet) for more than 100 examples were always zero in the numerical verification). Intuitively, in the presence of purely random patterns, the AND-mask has a desirable property: it decreases the strength of these signals exponentially fast, as opposed to linearly.

3.3 EXPERIMENTS

Real-world datasets are generated by (causal) generative processes which share mechanisms (Pearl, 2009). However, mechanisms and spurious signals are often entangled, making it hard to assess what part of the learning signal is due to either. As the goal of this chapter is to dissect these two components to understand how they ultimately contribute to the learning process, we create a simple synthetic dataset that allows us to control the complexity, intensity, and number of shortcuts in the data. After that, we evaluate whether spurious signals can be detected even in high-dimensional networks and datasets by testing the AND-mask on a memorization task similar to the one proposed in Zhang et al., 2017, and on a behavioral cloning task using the game CoinRun (Cobbe et al., 2019).

3.3.1 The synthetic memorization dataset

We introduce a binary classification task. The input dimensionality is $d = d_M + d_S$. While $p(y|x_{d_M})$ is the same across all environments (i.e. the *mechanism*), $p(y|x_{d_S}, e)$ is *not the same* across all environments (the *shortcuts*). While the mechanism is shared, it

needs a highly non-linear decision boundary to classify the data. The shortcuts are not shared across environments, but provide a simple way to classify the data, even when pooling all the environments together. See Figure 15 for a concrete example with d_M and d_S equal to 2, and two environments (A and B). The spirals (on d_M) are invariant but hard to model. The shortcuts (on d_S) are simple blobs but different in every environment: in A , linearly separable through a vertical decision boundary, in B with a horizontal one. If the two environments are pooled, a new diagonal decision boundary emerges on the shortcut dimensions as the most ‘natural’ one. While this perfectly classifies data in both environments A and B , critically *it would have not been found by training on either partition A or B alone*. The out-of-distribution (o.o.d.) test data has the same mechanism but random shortcuts. Therefore, any method relying exclusively on the shortcuts will have chance-level o.o.d. performance. Details about the dataset, baselines, and training curves are reported in appendix B.2.

Despite the apparent simplicity of this dataset, note that it is challenging to find the invariant mechanism. In high dimensions, even with tens of pooled environments, the shortcuts allow for a *simple* classification rule under almost every classical definition of ‘simple’: the boundary is *linear*, it has a *large margin*, it can be expressed with *small weights*, it is *fast to learn*, robust to input noise, and has *perfect accuracy and no i.i.d. generalization gap*. Finding the complex decision boundary of the spirals, instead, is a fiddly process and arguably a much slower path towards small loss.

BASELINES. We evaluate several domain-agnostic baselines (all multilayer perceptrons) with some of the most common regularizers used in deep learning — Dropout, L1, L2, Batch normalization. We also consider methods that explicitly make use of the environment labels, namely: (i) Domain Adversarial Neural Networks (DANN) (Ganin et al., 2016), a method specifically de-

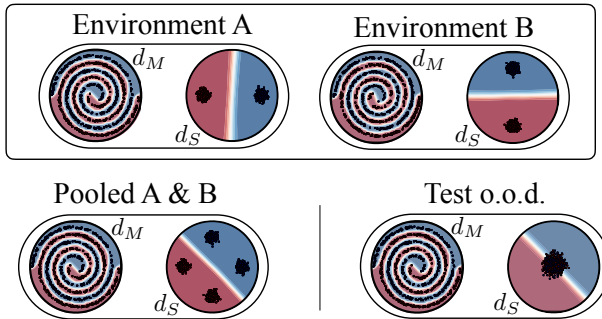


Figure 15: A 4-dimensional instantiation of the synthetic memorization dataset for visualization. Every example is a dot in both circles, and it can be classified by finding either of the “oracle” decision boundaries shown.

signed to address domain adaptation by obfuscating domain information with an adversarial classifier; (ii) Invariant Risk Minimization (IRM) (Arjovsky et al., 2019), discussed in detail in appendix B.2. The AND-mask is trained with the same configurations in Table 3.

RESULTS. Fig. 16 shows training and test accuracy. DANN fails because it can align the representation-layer distributions from different environments using only shortcuts, such that they become indistinguishable to the domain-discriminating classifier. The AND-mask was the only method to achieve perfect test accuracy, by fitting the spirals instead of the shortcuts. In particular, the combination of the AND-mask with L1 or L2 regularization gave the most robust results overall, as they help suppress neurons that at initialization are tuned towards the shortcuts.

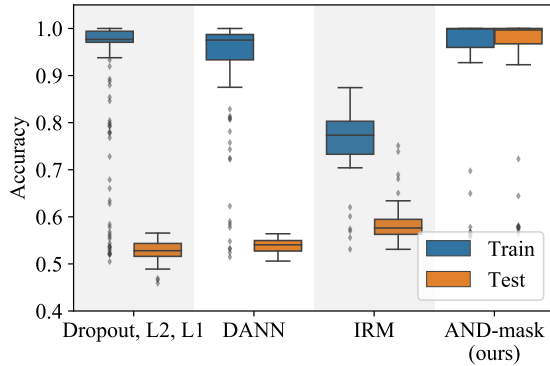


Figure 16: Results on the synthetic dataset.

CORRELATIONS BETWEEN AVERAGE, MEMORIZATION AND GENERALIZATION GRADIENTS. Due to the synthetic nature of the dataset, we can intervene on its data-generating process in order to examine the learning signals coming from the mechanisms and from the shortcuts. We isolate the two and measure their contribution to the average gradients, as we vary the agreement threshold of the mask. More precisely, we look at the gradients computed with respect to the weights of a randomly initialized network for different sets of data: (i) The original data, with mechanisms and shortcuts. (ii) Randomly permuting the dataset over the mechanisms dimensions, thus leaving the “memorization” signal of the shortcuts. (iii) Randomly permuting over the shortcuts dimensions, isolating the “generalization” signal of the mechanisms alone. Figure 17 shows the correlation between the components of the original average gradient (i) and the shortcut gradients ((ii), dashed line), and between the original average gradients and the mechanism gradients ((iii), solid line). While the signal from the mechanisms is present in the original average gradients (i.e.

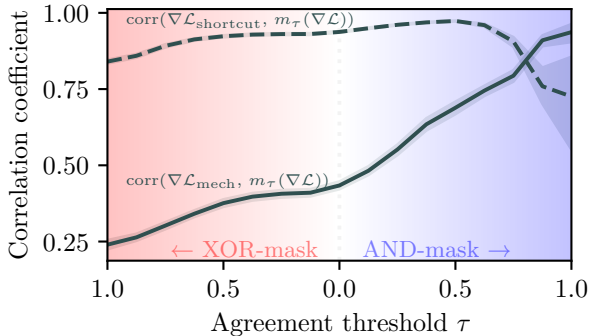


Figure 17: Gradient correlations.

$\rho \approx 0.4$ for $\tau = 0$), its magnitude is smaller and it is ‘drowned’ by the memorization signal. Instead, increasing the threshold of the AND-mask (right side) suppresses memorization gradients due to the shortcuts, and for $\tau \approx 1$ most of the gradient components remaining contain signal from the mechanism. On the left side, we test the other side of our hypothesis: An XOR-mask zeroes out consistent gradients, preserves those with different signs, and results in a sharper decrease of the correlation with the mechanism gradients.

3.3.2 Experiments on CIFAR-10

MEMORIZATION IN A VISION TASK. Zhang et al., 2017 showed that neural networks trained with standard regularizers — like L2 and Dropout — can still memorize large training datasets with *shuffled* labels, i.e. reaching $\approx 100\%$ training accuracy. Their experiments raised significant questions about the generalization properties of neural networks and the role of regularizers in constraining the hypothesis class. Our hypothesis is that ILC — for

example implemented as the AND-mask — should prevent memorization on a similar task with the shuffled labels, as gradients will tend to largely ‘disagree’ in the absence of a shared mechanism. However, when the labels are *not shuffled*, ILC should have a much weaker effect, as real shared mechanisms are still present in the data.

To test our hypothesis, we ran an experiment that closely resembles the one in (Zhang et al., 2017) on CIFAR-10. We trained a ResNet on CIFAR-10 with *random labels*, with and without the AND-mask. In all experiments we used batch size 80, and treated each example as its own “environment”. Recall that standard gradient averaging is equivalent to an AND-mask with threshold 0. As shown in Figure 18, the ResNet with standard average gradients memorized the data, while slightly increasing the threshold for the AND-mask quickly prevented memorization (dark blue line). In contrast, training the same networks on the dataset *with the original labels* resulted in both of them converging and generalizing to the test set, confirming that the mask did not significantly affect the generalization error with a general underlying mechanism in the data.

Note that there is no standard notion of environments in CIFAR-10, which is why we treated every example as coming from its own environment. This assumption is not unreasonable, as every image in the dataset was literally collected in a different physical environment. If anything, it is the standard i.i.d. assumption that hides this variety behind a notion of a single distribution encompassing all environments. The results of this experiment further support this interpretation, and can serve as evidence that — in some cases — we might be able to identify invariances even without an explicit partition into environments, as this can be already identified at the level of individual examples.

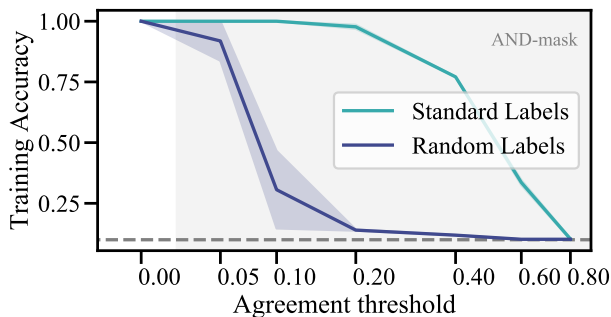


Figure 18: As the AND-mask threshold increases, memorization on CIFAR-10 with random labels is quickly hindered.

LABEL NOISE. Following up on this experiment, we test how the AND-mask performs in the presence of label noise, i.e. when a portion of the labels in the training set are randomly shuffled (25% here). According to our hypothesis, gradients computed on examples with random labels should disagree and get masked out by the AND-mask, while signal from correctly labeled data should contribute to update the model. As shown in Figure 19, the performance on the *incorrectly* labeled portion of the dataset is well *below* chance for the AND-mask (as it predicts correctly despite the wrong labels), while the baseline again memorizes the incorrect labels. On the test set (with untouched labels), the baseline peaks early then decreases as the model overfits, while the AND-mask slowly but steadily improves.

3.3.3 Behavioral Cloning on CoinRun

CoinRun (Cobbe et al., 2019) is a game introduced to test how RL agents generalize to novel situations. The agent needs to collect

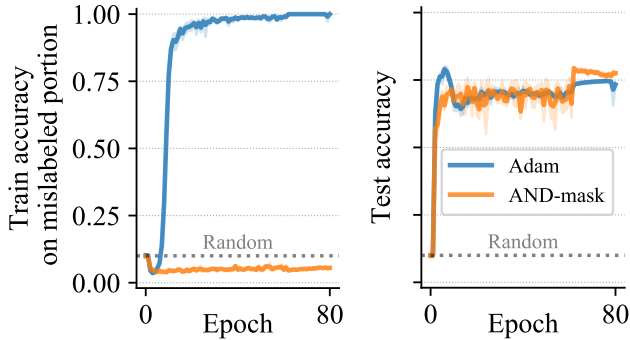


Figure 19: The AND-mask prevents overfitting to the incorrectly labeled portion of the training set (left) without hurting the test accuracy (right).

coins, jumping on top of walls and boxes and avoiding enemies.⁹ Each level is procedurally generated — i.e. it has a different combination of sprites, background, and layout — but the physics and goals are invariant. Cobbe et al., 2019 showed that state-of-the-art RL algorithms fail to model these invariant mechanisms, performing poorly on new levels unless trained on thousands of them. To test our hypothesis, we set up a behavioral cloning task using CoinRun.¹⁰ We start by pre-training a strong policy π^* using standard PPO (Schulman et al., 2017) for 400M steps on the full distribution of levels. We then generate a dataset of pairs $(s, \pi^*(a|s))$ from the on-policy distribution. The training data consists of 1000 states from each of 64 levels, while test data comes from 2000 levels. A ResNet-18 $\hat{\pi}_\theta$ is then trained to minimize the

⁹ See Figure 41 in appendix B.2.6 for a visualization of the game.

¹⁰ To obtain a robust evaluation, we preferred to approach behavioral cloning instead of the full RL problem, as it is a standard supervised learning task and has substantially fewer moving parts than most deep RL algorithms.

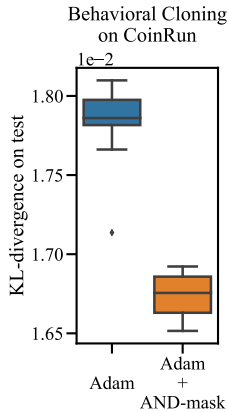


Figure 20: Results on CoinRun with and without the temporal variation of the AND-mask

loss $D_{\text{KL}}(\pi^* || \hat{\pi}_\theta)$ on the training set. We compare the generalization performance of regular Adam to a version that uses the AND-mask. For each method we ran an automatic hyperparameter optimization study using Tree-structured Parzen Estimation (Bergstra et al., 2013) of 1024 trials. Despite the theoretical computational efficiency of computing the AND-mask as presented in Section 3.2.3 (i.e., linear time and memory in the size of the mini-batch, just like classic SGD), current deep learning frameworks like PyTorch (Paszke et al., 2017) have optimized routines that sum gradients across examples in a mini-batch before it is possible to efficiently compute the AND-mask. We therefore test the AND-mask in a slightly different way. In training, in each iteration we sample a batch of data from a randomly chosen level out of the 64 available (and cycle through them all once per epoch). We then apply the AND-mask ‘temporally’, only allowing gradients that are consistent across time (and therefore across levels). See Algorithm 4 in appendix B.2.6 for a detailed description of

this alternative formulation of the AND-mask. The figure shows the minimum test loss for the 10 best runs, supporting the hypothesis that the AND-mask helps identify invariant mechanisms across different levels.

3.4 RELATED WORK

Generalization and covariate shift. The classic formulation of statistical learning theory (Vapnik, 1999) concerns learning from independent and identically distributed samples. The case where the distribution of the covariates at test time differs from the one observed during training is termed *covariate shift* (Sugiyama et al., 2007; Quionero-Candela et al., 2009; Sugiyama and Kawabata, 2012). Standard solutions involve re-weighting of the training examples, but require the additional assumption of overlapping supports for train and test distributions.

Causal models and invariances. As we mentioned in the introduction, causality provides a strong motivation for our work, based on the notion that statistical dependencies are epiphenomena of an underlying causal model (Pearl, 2009; Peters et al., 2017a). The causal description identifies stable elements – e.g. physical *mechanisms* – connecting causes and effects, which are expected to remain *invariant* under interventions or changing external conditions (Haavelmo, 1943; Schölkopf et al., 2012)). This motivates our notion of invariant mechanisms, and inspired related notions which have been proposed for robust regression (Rojas-Carulla et al., 2018; Heinze-Deml et al., 2018; Arjovsky et al., 2019; Hermann and Lampinen, 2020; Ahuja et al., 2020; Krueger et al., 2020). We discuss this in more detail in appendix B.3.1.

Domain generalization. ILC can be used in a setting of domain generalization (Muandet et al., 2013), but it is not limited to it: as demonstrated in the experiments in Section 3.3.2, the AND-mask can be applied even if domain labels are not available. In contrast,

by treating every example as a single domain, methods relying on domain classifiers (like DANN [Ganin et al., 2016](#) or [Balaji et al. \(2018\)](#)) would require as many output units as there are training examples (i.e. 50'000 for CIFAR-10).

Gradient agreement. Looking at gradient agreement to learn meaningful representations in neural networks has been explored in ([Du et al., 2018](#); [Eshratifar et al., 2018](#); [Fort et al., 2019](#); [Zhang et al., 2019b](#)). These approaches mainly rely on a measure of cosine similarity between gradients, which we did not consider here for two main reasons: (i) It is a ‘global’ property of the gradients, and it would not allow us to extract precise information about different patterns in the network; (ii) It is unclear how to extend it beyond pairs of vectors, and for pairwise interactions its computational cost scales quadratic in the number of examples used.

3.5 CONCLUSIONS

Generalizing out of distribution is one of the most significant open challenges in machine learning, and relying on invariances across environments or examples may be key in certain contexts. In this chapter we analyzed how neural networks trained by averaging gradients across examples might converge to solutions that ignore the invariances, especially if these are harder to learn than spurious patterns. We argued that if learning signals are collected *on one example at the time* — as it is the case for gradients, e.g., computed with backpropagation — the way these signals are aggregated can play a significant role in the patterns that will ultimately be expressed: Averaging gradients in particular can be too permissive, acting as a *logical OR* of a collection of distinct patterns, and lead to a ‘patchwork’ solution. We introduced and formalized the concept of Invariant Learning Consistency, and showed how to learn invariances even in the face of alternative explanations that — although spurious — fulfill most characteristics of a good solution. The AND-mask is but one of multiple

possible ways to improve consistency, and it is unlikely to be a practical algorithm for all applications. However, we believe this should not distract from the general idea which we are trying to put forward — namely, that it is worthwhile to study learning of explanations that are *hard to vary*, with the longer term goal of advancing our understanding of learning, memorization and generalization.

NEURAL SYMBOLIC REGRESSION THAT SCALES

A picture may be worth a thousand words, a formula is worth a thousand pictures.

Edsger Dijkstra

CHAPTER ABSTRACT Symbolic equations are at the core of scientific discovery. The task of discovering the underlying equation from a set of input-output pairs is called symbolic regression. Traditionally, symbolic regression methods use hand-designed strategies that do not improve with experience. In this chapter, we introduce the first symbolic regression method that leverages large scale pre-training. We procedurally generate an unbounded set of equations, and simultaneously pre-train a Transformer to predict the symbolic equation from a corresponding set of input-output-pairs. At test time, we query the model on a new set of points and use its output to guide the search for the equation. We show empirically that this approach can re-discover a set of well-known physical equations, and that it improves over time with more data and compute.

This chapter is based on the paper “*Neural Symbolic Regression that Scales*”, Luca Biggio* , Tommaso Bendinelli* , Alexander Neitz, Aurelien Lucchi, Giambattista Parascandolo (Biggio et al., 2021)

4.1 INTRODUCTION

Since the early ages of Natural Sciences in the sixteenth century, the process of scientific discovery has rooted in the formalization of novel insights and intuitions about the natural world into compact symbolic representations of such new acquired knowledge, namely, mathematical equations.

Mathematical equations encode both objective descriptions of experimental data and our inductive biases about the regularity we attribute to natural phenomena. When seen under the perspective of modern machine learning, they present a number of appealing properties: (i) They provide *compressed* and *explainable* representations of complex phenomena. (ii) They allow to easily incorporate prior knowledge. (iii) When relevant aspects about the data generating process are captured, they often generalize well beyond the distribution of the observations from which they were derived.

The process of discovering symbolic expressions from experimental data is hard and has traditionally been one of the hallmarks of human intelligence. *Symbolic regression* is a branch of regression analysis that tries to emulate such a process. More formally, given a set of n input-output pairs $\{(x_i, y_i)\}_{i=1}^n \sim \mathcal{X} \times \mathcal{Y}$, the goal is to find a symbolic equation e and corresponding function f_e such that $y \approx f_e(x)$ for all $(x, y) \in \mathcal{X} \times \mathcal{Y}$. In other words, the goal of symbolic regression is to infer both model structure and model parameters in a data-driven fashion. Even assuming that the vocabulary of primitives — e.g. $\{\sin, \exp, +, \dots\}$ — is sufficient to express the *correct* equation behind the observed data, symbolic regression is a hard problem to tackle. The number of functions associated with a string of symbols grows exponentially with the string length, and the presence of numeric constants further exacerbates its difficulty.

Due to its challenging combinatorial nature, existing approaches to symbolic regression are mainly based on search-techniques whose goal is typically to minimize a pre-specified fitness function measuring the distance between the predicted expression and the available data. The two main drawbacks of such methods are that: (i) *They do not improve with experience*. As every equation is regressed from scratch, the system does not improve if access to more data from different equations is given. (ii) *The inductive bias is opaque*. It is difficult for the user to steer the prior towards a specific class of equations (e.g. polynomials, etc.). In other words, even though most symbolic regression algorithms generate their prediction starting from a fixed set of primitives reflecting the user’s prior knowledge, such elementary building blocks can be combined in many arbitrary ways, providing little control over the equation distribution. To overcome both drawbacks, in this chapter we take a step back, and let the model *learn the task* of symbolic regression over time, on a user-defined prior over equations.

Building on the recent successes of large models trained on large datasets [Brown et al., 2020](#); [Devlin et al., 2019](#); [Chen et al., 2020b](#); [Chen et al., 2020c](#), we show that a strong symbolic regressor can be purely learned from data. The key factor behind our approach is that computers can generate unbounded amounts of data with perfect accuracy and at virtually no cost. The distribution over equations used during pre-training strongly influences the prior over equations of the final system. Such a prior thus becomes easy to understand and control.

The main contributions of this chapter are the following:

- We introduce a simple, flexible, and powerful framework for symbolic regression, the first approach (to the best of our knowledge) to improve over time with data and compute.

- We demonstrate that *learning the task* of symbolic regression from data is sufficient to significantly outperform state-of-the-art approaches relying on hand-designed strategies.
- We release our code and largest pre-trained model ¹

In Section 5.4, we detail related work in the literature. In Section 4.3, we present our algorithm for neural symbolic regression that scales. We evaluate the method in the experiments described in Section 5.5 and 4.5 and compare it to state-of-the-art baselines. In Section 4.6 we discuss results, limitations, and potential for future work.

4.2 RELATED WORK

GENETIC PROGRAMMING FOR SYMBOLIC REGRESSION Traditional approaches to symbolic regression are based on genetic algorithms [Forrest, 1993](#) and, in particular, genetic programming (GP) [Koza, 1994](#). GP methods used for symbolic regression iteratively «evolve» a population of candidate mathematical expressions via mutation and recombination. The most popular GP-based technique applied to symbolic regression is undoubtedly the commercial software Eureqa [Dubčáková, 2011](#) which is based on the approach proposed by Schmidt and Lipson (2009). Despite having shown for the first time the potential of data-driven approaches to the problem of function discovery, GP-based techniques do not scale well to high dimensional problems and are highly sensitive to hyperparameters [Petersen, 2021](#).

NEURAL NETWORKS FOR SYMBOLIC REGRESSION A more recent line of research explores the potential of deep neural networks to tackle the combinatorial challenge of symbolic regression. Martius and Lampert (2016) propose a simple fully-connected

¹ <https://github.com/SymposiumOrganization/NeuralSymbolicRegressionThatScales>

neural network where standard activation functions are replaced with symbolic building blocks (e.g. « $\sin(\cdot)$ », « $\cos(\cdot)$ », « $+$ », « $\text{Id}(\cdot)$ »). Once the model is trained, a symbolic formula can be automatically read off from the network architecture and weights. This method inherits the ability of neural networks to deal with high-dimensional data and scales well with the number of input-output pairs. However, it requires specific extensions [Sahoo et al., 2018](#) to deal with functions involving divisions between elementary building blocks (e.g. $\frac{\sin(x)}{x^2}$) and the inclusion of exponential and logarithmic activations result in exploding gradients and numerical issues.

A different approach to circumvent the discrete combinatorial search inherent in the symbolic regression framework is proposed in [Kusner et al., 2017](#). Here, a variational autoencoder [Kingma and Welling, 2014](#) is first trained to reconstruct symbolic expressions and the search for the best fitting function is then performed over the latent space in a subsequent step. While the idea of moving the search for the best expression from a discrete space to a continuous one is interesting and has been exploited by other approaches (e.g. [Alaa and Schaar, 2019](#)), the method does not prove to be effective in recovering relatively simple symbolic formulas. More recently, Petersen ([2021](#)) developed a new technique where a recurrent neural network (RNN) is used to model a probability distribution over the space of mathematical expressions. Output expressions contain symbolic placeholders to indicate the presence of numerical constants. Such constants are then fit in a second stage by an out-of-the-box nonlinear optimizer. The RNN is trained by minimizing a risk-seeking RL objective that assigns a larger reward to the top-epsilon samples from the output distribution. The method represents a significant step forward in the application of deep learning to symbolic regression. While showing promising results, the network has to be retrained from scratch

for each new equation and the RNN is never directly conditioned on the data it is trained to model.

Finally, neural networks can also be used in combination with existing techniques or hand-designed rules to perform symbolic regression. Notable examples are [Udrescu and Tegmark, 2020](#); [Udrescu et al., 2020](#), where neural networks are employed to identify simplifying properties in the data such as additive separability and compositionality. These properties are exploited to recursively simplify the original dataset into less challenging sub-problems that can be tackled by a symbolic regression technique of choice. A similar rationale is followed in [Cranmer et al., 2020](#), where different components of a trained Graph Neural Network (GNN) are independently fit by a symbolic regression algorithm. By joining the so-found expressions, a final algebraic formula describing the network can be obtained. The aforementioned approaches might provide very good performances when it is known a priori whether the data are characterized by specific structural properties, such as symmetries or invariances. However, when such information is not accessible, more domain-agnostic methods are required.

LARGE SCALE PRE-TRAINING Our approach builds upon a large body of work emphasizing the benefits of pre-training large models on large datasets [Kaplan et al., 2020](#); [Devlin et al., 2019](#); [Brown et al., 2020](#); [Chen et al., 2020b](#); [Chen et al., 2020c](#); [Belkin et al., 2019](#). Examples of such models can be found in Computer Vision [Radford et al., 2021](#); [Chen et al., 2020b](#); [Chen et al., 2020c](#); [Kolesnikov et al., 2020](#); [Oord et al., 2018a](#) and Natural Language Processing [Devlin et al., 2019](#); [Brown et al., 2020](#). There have also been recent applications of Transformers [Vaswani et al., 2017](#) to tasks involving symbolic mathematics manipulations [Lample and Charton, 2020](#); [Saxton et al., 2019](#) and automated theorem proving

[Polu and Sutskever, 2020](#). Our work builds on the results from [Lample and Charton \(2020\)](#), where Transformers are trained to successfully perform challenging mathematical tasks such as symbolic integration and solving differential equations. However, our setting presents the additional challenge of mapping *numerical values* to the corresponding symbolic formula, instead of working exclusively within the symbolic domain.

4.3 NEURAL SYMBOLIC REGRESSION THAT SCALES

A symbolic regressor S is an algorithm which takes a set of n input-output pairs $\{(x_i, y_i)\}_{i=1}^n \sim \mathcal{X} \times \mathcal{Y}$ as input and returns a symbolic equation e representing a function f_e such that: $y \approx f_e(x), \forall (x, y) \in \mathcal{X} \times \mathcal{Y}$. In this section, we describe our framework to *learn* a parametrized symbolic regressor S_θ from a large number of training data.

4.3.1 Pre-training

We pre-train a Transformer on hundreds of millions of equations which are procedurally generated for every minibatch. As equations and datapoints can be generated quickly and in any amount using a computer and standard math libraries, we can train the network end-to-end to predict the equations on a dataset that is potentially unbounded. We describe the exact process we use to generate the dataset in [Section 5.5](#).

An illustration of the main steps involved in the pre-training phase is shown in [Fig. 21](#).

DATA During the pre-training phase, each training example consists of a symbolic equation e which represents a function $f_e : \mathbb{R}^{d_x} \rightarrow \mathbb{R}^{d_y}$, a set of n input points $X = \{x_i\}_{i=1}^n$ and corresponding outputs $Y = \{f_e(x_i)\}_{i=1}^n$. The distribution, $\mathcal{P}_{e,X}$, from which e

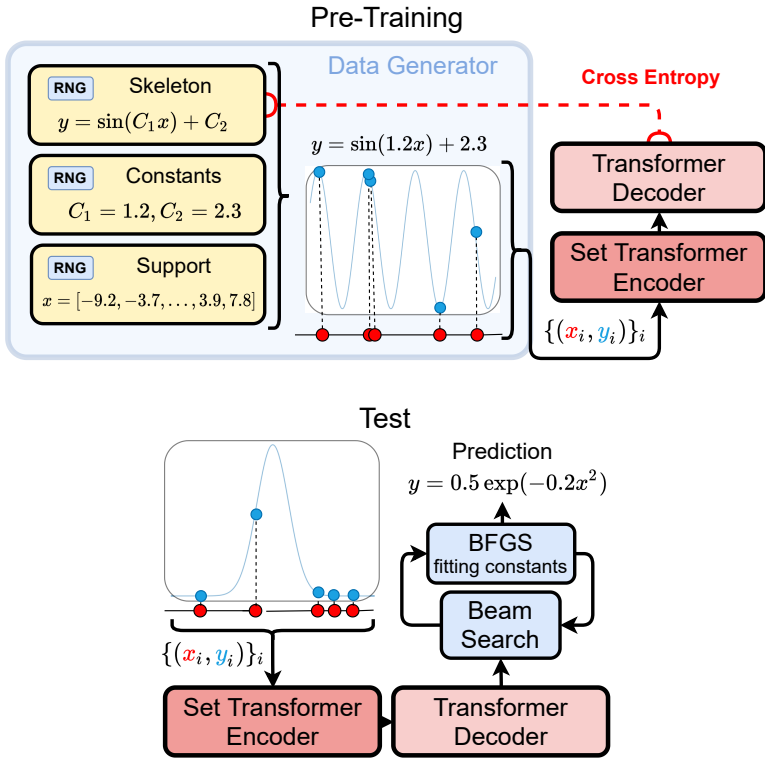


Figure 21: (Above) The data generator produces the input for the Transformer and its target expression. It does so by randomly sampling (i) an equation skeleton (including placeholders for the constants), (ii) numerical constants used to replace the placeholders and (iii) a set of support points $\{x_i\}_i$ to evaluate the previously generated equation and get the corresponding $\{y_i\}_i$. The $\{(x_i, y_i)\}_i$ pairs are fed into the Transformer, which is trained to minimize the cross-entropy loss with the ground-truth skeleton without numerical constants. Both the model output and the targets are expressed in prefix notation. (Below) At test time, given new input data, we sample candidate symbolic skeletons from the model using beam-search. The final candidate equations are obtained by fitting the constants with BFGS.

and the inputs X are sampled will determine the inductive bias of the trained symbolic regressor and should be chosen to resemble the application domain. In particular, X can vary in size (i.e. n is not fixed), and the individual inputs x_i do not have to be *i.i.d* – neither within X nor across examples or batches. For example, $\mathcal{P}_{e,X}$ could be polynomials of degree up to 6, and input sets of up to 100 points sampled uniformly from the range $[0, 1]$. In our experiments, an equation e is represented by a sequence of symbols in prefix notation. An equation e can contain numerical constants that are re-sampled at each batch to increase the diversity of the data seen by the model. In Section 5.5, we describe the details of the data generation process we used in our experiments.

PRE-TRAINING We train a parametric set-to-sequence model S_θ to predict the equation e from the set of input-output points X, Y . In our implementation, S_θ consists of an encoder and a decoder. The encoder maps the (x, y) sequence pairs for each equation into a latent space, resulting in a fixed-size latent representation z . A decoder generates a sequence \bar{e} given z : it produces a probability distribution $P(\bar{e}_{k+1} | \bar{e}_{1:k}, z)$ over each symbol, given the previous symbols and z . The alphabet of \bar{e} is identical to the one used for the original equations e , with one exception: unlike e , \bar{e} does not contain any numerical constants. Instead, it contains a special placeholder symbol ‘ \diamond ’ which denotes the presence of a constant which will be fit at a later stage. For example, if $e = 4.2 \sin(0.3x_1) + x_2$, then $\bar{e} = \diamond \sin(\diamond x_1) + x_2$. We refer to the equation where numerical constants are replaced by placeholders as the “skeleton” of the equation, and use the notation \bar{e} to refer to the symbolic equation that replaces numerical constants with ‘ \diamond ’. The model is trained to reduce the average loss between the predicted \hat{e} and $\text{skeleton}(e)$, i.e. the skeleton of the original equation. Training is performed with mini-batches of B equations each. The overall pre-training algorithm is reported in Algorithm 2.

Algorithm 2 Neural Symbolic Regression pre-training

 Precondition: S_θ , batch size B , training distribution $\mathcal{P}_{e,X}$
while not timeout **do**
 $L \leftarrow 0$
for i in $\{1..B\}$ **do**
 $e, X \leftarrow$ sample an equation and input set from $\mathcal{P}_{e,X}$
 $Y \leftarrow \{f_e(x) \mid x \in X\}$
 $\bar{e} \leftarrow$ skeleton(e)

 $L \leftarrow L - \sum_k \log P_{S_\theta}(\bar{e}_{k+1} \mid \bar{e}_{1:k}, X, Y)$

 Compute the gradient $\nabla_\theta L$ and use it to update θ .

4.3.2 Test time

At test time, given a set of input-output pairs $\{(x_i, y_i)\}_i$ we encode them using the encoder into a latent vector z . From z we iteratively sample candidates skeletons of symbolic equations \hat{e} from the decoder. Finally, for each candidate, we fit the numerical constants \diamond by treating each occurrence as an independent parameter. This can be achieved using a non-linear optimizer, either gradient-based or black-box, by minimizing a loss between the resulting equation applied to the inputs and the targets Y . In our experiments, we used beam-search to sample high-likelihood equation candidates from the decoder, and, like Petersen (2021), BFGS Fletcher, 1987 on the mean squared error to fit the constants.

4.4 EXPERIMENTAL SET-UP

Here, we present the instantiation of the framework described in Section 4.3 that we evaluate empirically, and detail the baselines

and datasets used to test it. For the rest of the chapter, we will refer to our implementation as NeSymReS².

4.4.1 The Model S_θ

For the encoder we opted for the Set Transformer architecture from Lee et al. (2019), using the original publicly available implementation.³ We preferred this to the standard Transformer encoder, as the number n of input-output pairs can grow to large values, and the computation in Set Transformers scales as $\mathcal{O}(nm)$ instead of $\mathcal{O}(n^2)$, where $m \ll n$ is a set of learnable inducing points Snelson and Ghahramani, 2005; Titsias, 2009 we keep constant at $m = 50$. For the decoder we opted for a regular Transformer decoder Vaswani et al., 2017, using the default PyTorch implementation. Encoder and decoder have 11 and 13 million parameters respectively. The hyperparameters chosen for both networks — detailed in Section C.1 — were not fine-tuned for maximum performance.

4.4.2 Pre-training Data Generator

We sample expressions following the framework introduced in Lample and Charton, 2020. A mathematical expression is regarded as a unary-binary tree where nodes are operators and leaves are independent variables or constants. Once an expression is sampled, it is simplified using the rules built in the symbolic manipulation library SymPy Meurer et al., 2017. This sampling method allows us to precisely constrain the search space by controlling the depth of the trees and the set of admissible operators, along with their prior probability of occurring in the generated expression. We opted for scalar functions of up to three independent

² For Neural Symbolic Regression that Scales

³ https://github.com/juho-lee/set_transformer

input variables (i.e. $d_x = 3$ and $d_y = 1$). For convenience, we pre-sampled 10 million skeletons of equations with up to three numerical constants each. At training time, we sample mini-batches of size $B = 150$ of the following elements:

Equation skeletons with constant placeholders placed randomly inside the expressions.

Constants values C_1, C_2, C_3 , each independently sampled from a uniform distribution $\mathcal{U}(1, 5)$.

Support extrema $S_{1,j}, S_{2,j}$, with $S_{1,j} < S_{2,j}$ uniformly sampled from $\mathcal{U}(-10, 10)$ independently for each dimension $j = 1, \dots, d_x$.

Input points for each input dimension $j = 1, \dots, d_x$. A set of n input points, $X_j = \{x_{i,j}\}_{i=1}^n$, is uniformly sampled from $\mathcal{U}(S_{1,j}, S_{2,j}, n)$.

We then evaluate the equations on the input points $X = \{x_i\}_{i=1}^n$ to obtain the corresponding outputs Y .

As Y can take very large or very small values, this can result in numerical instabilities and exploding or vanishing gradients during training. Therefore, we convert every x_i and y_i from float to a multi-hot bit representation according to the half-precision IEEE-754 standard. Furthermore, in order to avoid invalid operations (i.e. dividing by zero, or taking the logarithm of negative values), we drop out input-output pairs containing NaNs.

We train the encoder and decoder jointly to minimize the cross-entropy loss between the ground truth skeleton and the skeleton predicted by the decoder as a regular language model. We use Adam with a learning rate of 10^{-4} , no schedules, and train for 1.5M steps. Overall, this results in about 225M distinct equations seen during pre-training. See Appendix C.2 for more details about training and resulting training curves.

4.4.3 *Symbolic Regression at Test Time*

Given a set of input-output pairs from an unknown equation e , we feed the points into the encoder and use beam-search to sample candidate skeletons from the decoder. We then use BFGS to recover the values of the constants, by minimizing the squared loss between the original outputs and the output from the predicted equations. Our default parameters at test time are beam-size 32, with 4 restarts of BFGS per equation. We select the best equation from the set of resulting candidates based on the in-sample loss with a small penalty of $1e-14$ per token of the skeleton.⁴

4.4.4 *Evaluation*

We evaluate our trained model on five datasets. Unless otherwise specified, for all equations we sample 128 points at test time.

AI-FEYNMAN (AIF) First, we consider all the equations with up to 3 independent variables from the AI-Feynman (AIF) database [Udrescu and Tegmark, 2020](#)⁵. The resulting dataset consists of 52 equations extracted from the popular *Feynman Lectures on Physics* series. We checked our pre-training dataset, and amongst the 10 million equation skeletons, all equations from AIF appear. However, as mentioned in the previous subsection, the support on which they are evaluated, along with the constants and number of points per equation, is continuously sampled at every training iteration, making it impossible to exactly see any of the test data at training time.

⁴ While we found this strategy to work well in practice, a validation set for model selection might offer better performances with noisy data.

⁵ <https://space.mit.edu/home/tegmark/aifeynman.html>

UNSEEN SKELETONS (SOOSE) This dataset of 200 equations is specifically constructed to have zero overlap with the pre-training set, meaning that its equations are all symbolically and numerically different from those included in the pre-training set. We call it SOOSE, for strictly out-of-sample equations. Compared to AIF, these equations are on average significantly longer and more complex (see Table 11). The sampling distribution for the skeletons is the same as the pre-training distribution, but we instantiate three different versions: with up to three constants (same as pre-training distribution, SOOSE-WC); no constants (SOOSE-NC); constants everywhere (SOOSE-FC, for full constants), i.e. one constant term for each term in the equation. The latter is extremely challenging, and since NeSymReS was only pre-trained with up to three constants, it is far from its pre-training distribution.

NGUYEN DATASET This dataset consists of 12 simple equations *without* constants beyond the scalars 1 and 2, each with up to 2 independent variables. Nguyen was the main benchmark used in Petersen, 2021. There are terms that appear in three ground truth equations that are *not* included in the set of equations that our model can fit, specifically x^6 , and x^y , which therefore caps the maximum accuracy that can be reached by our model on this dataset.

4.4.5 Baselines

We compare the performance of our method with the following baselines:

DEEP SYMBOLIC REGRESSION (DSR) Petersen, 2021 Recently proposed RNN-based reinforcement learning search strategy for symbolic regression. We use the open-source implemen-

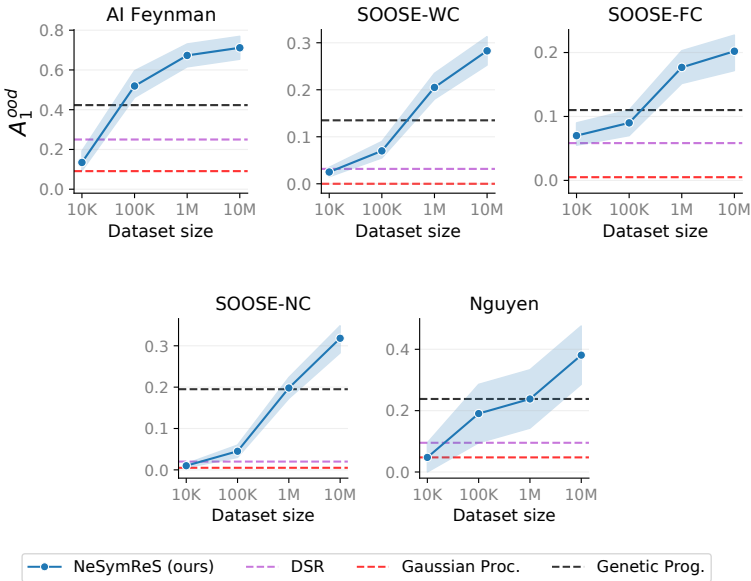


Figure 22: Accuracy as a function of the size of the pre-training dataset, for a fixed computational budget (~ 100 s) at test time. We report reference values for the baselines to emphasize that these approaches do not improve with experience over time.

tation provided by the authors⁶, with the setting that includes the estimation of numerical constants in the final predicted equation.

GENETIC PROGRAMMING [Koza, 1994](#) Standard GP-based symbolic regression based on the open-source Python library [gplearn](#)⁷.

⁶ <https://github.com/brendenpetersen/deep-symbolic-regression>

⁷ <https://gplearn.readthedocs.io/en/stable/>

GAUSSIAN PROCESSES [Rasmussen, 2003](#) Standard Gaussian Process regression with RBF and constant kernel. We use the open source `sklearn` implementation⁸.

All details about baselines are reported in Appendix [C.1](#).

Two notable exclusions are AIF [Udrescu and Tegmark, 2020](#) and EQL [Martius and Lampert, 2016](#). As also noted by Petersen ([2021](#)), in cases where real numerical constants are present or the equations are not separable, the former still requires a complementary symbolic regression method to cope with the discrete search. The latter lacks too many basis functions that appear in the datasets we consider, preventing it from recovering most of the equations. Moreover, its average runtime and number of points required to solve the equations indicated in [Martius and Lampert, 2016](#); [Sahoo et al., 2018](#) are three orders of magnitudes higher than the standards reported by the aforementioned baselines.

4.4.6 Metrics

Evaluating whether two equations are equivalent is a challenging task in the presence of real valued constants.

We distinguish between accuracy *within* the training support (A^{iid}), and outside of the training support (A^{ood}). A^{iid} is computed with 10k points sampled uniformly in the training support. A^{ood} is computed with 10k points in an extended support as detailed in Appendix [C.2](#), and it will be the main metric of interest.

We further distinguish between two metrics, accuracy A_1 and accuracy A_2 , each of which can be either computed *iid* or *ood*. Accuracy A_1 is computed as follows: for every point (x, y) and prediction $f_{\hat{e}}(x) = \hat{y}$, the point is correctly classified if the function `numpy.isclose(y, \hat{y})` returns True.⁹ Then, an equation

⁸ https://scikit-learn.org/stable/modules/gaussian_process.html

⁹ With parameters `atol 1e-3` and `rtol 0.05`.

is correctly predicted if $> 95\%$ of points are correctly classified. For this metric we can keep *all* outputs, including NaNs and $\pm\infty$, which are still representative of whether the symbolic equation was identified correctly. Accuracy A_2 is computed by measuring the coefficient of determination R^2 between y and \hat{y} , excluding NaNs and $\pm\infty$. An equation is correctly identified according to A_2 if the $R^2 > 0.95$. We found the two metrics to correlate significantly, and in the interest of clarity we will use only A_1 in the main text, and show results with A_2 in the Appendix C.3.

4.5 RESULTS

We test three different aspects of the proposed approach: (i) To what extent does performance improve as we increase the size of the pre-training data? (ii) How does our approach compare to state-of-the-art methods in symbolic regression? (iii) What is the impact of the number of input-output pairs available at *test time*?

(i) Accuracy as a Function of Pre-training Data

In order to test the effect of pre-training data on test performance, we trained our NeSymReS model on increasingly larger datasets. More specifically, we consider datasets consisting of 10K, 100K, 1M and 10M equation skeletons. Every aspect of training is the same as described in Section 5.5. We train all models for the same number of iterations, but use early stopping on a held-out validation set to prevent overfitting.

In Figure 43 we report the accuracy on the 5 test sets using a beam size of 32 for NeSymReS, and for all baselines whatever hyperparameter configuration that used comparable (but strictly no less) amount of computing time. In all datasets, increasing the size of the pre-training data results in higher accuracy for NeSymReS. Note that the baselines do not make use of the avail-

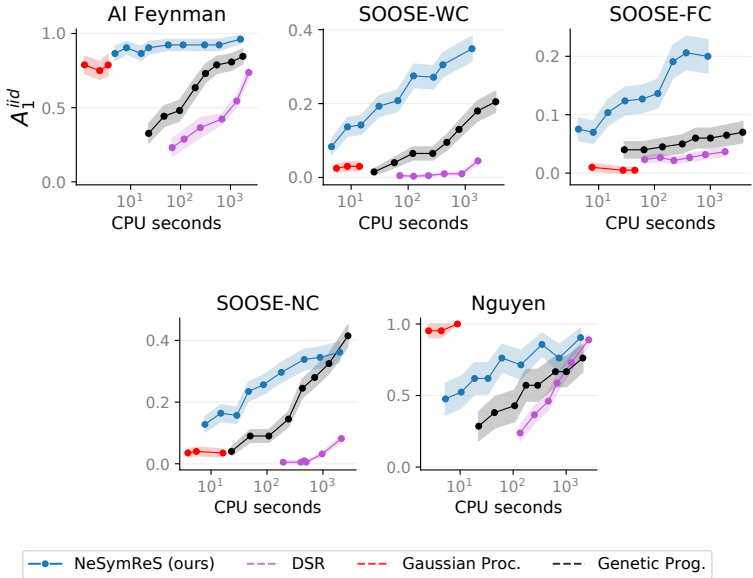


Figure 23: Accuracy *in distribution* as a function of time for all methods ran on a single CPU per equation.

able pre-training data, and as such it does not have any effect on the performance at test time. From here onwards, we will always use the model pre-trained on 10M equation skeletons.

Conclusion: The performance of NeSymReS steadily improves as the size of the pre-training dataset increases, exploiting the feature that symbolic equations can be generated and evaluated extremely quickly and reliably with computers. The trend observed appears to continue for even larger datasets, in accordance to Kaplan et al., 2020, which leaves open interesting avenues for extremely large scale experiments.

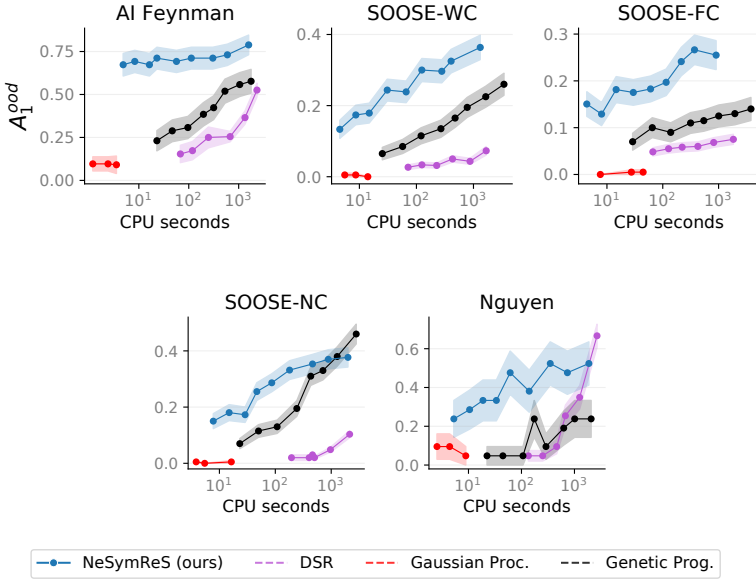


Figure 24: Accuracy *out of distribution* as a function of time for all methods ran on a single CPU per equation.

(ii) Accuracy as a Function of Test-time Compute.

For every method (including baselines), we vary the corresponding hyper-parameter that increases how much time and compute is invested at test time to recover an equation from observing a fixed set of input-output pairs. We report the hyper-parameters and ranges in Table 1.

Making a fair comparison of run-times between different methods is another challenging task. To make the comparison as fair as possible, we decided to run every method on a single CPU at the time. Note that this is clearly a sub-optimal hardware setting for

our 26-million parameters Transformer, which would be highly parallelizable on GPU.

The results on all five datasets are shown in Figure 23 and Figure 24. On all datasets, our method outperforms all baselines both in time and accuracy by a large margin on most budgets of compute. On AIF our NeSymRes is more than three orders of magnitudes faster at reaching the same maximum accuracy as the second-best method, i.e. Genetic Programming, despite running on CPU only. We attribute the low accuracy achieved by Petersen, 2021 to the presence of constants, to the fact that their model does not directly observe the input-output pairs, and the use of REINFORCE Williams, 2004. The Gaussian Process baseline performs extremely well in distribution, reaching high accuracy in a very short amount of time, but poorly out of distribution. This is expected as it does not try to regress the symbolic equation. On Nguyen, NeSymReS achieves relatively high scores more rapidly than the other baselines. For large computation times ($\approx 10^3$ seconds) NeSymReS performs comparably with DSR despite the latter being fine-tuned on two equations of the benchmark (Nguyen-7 and Nguyen-10). The relatively lower performance of NeSymReS on SOOSE-NC can be explained by the fact that both datasets do not have any constants in the equations, while NeSymReS is trained with a large prior on the presence of constants.

Table 1: Hyper-parameters that vary to increase the amount of compute invested by every method.

Method	Hyper-param	Range
Gaussian Proc. (Rasmussen, 2003)	Opt. restarts	$\{8, 16, 32\}$
Genetic Programming (Koza, 1994)	Pop. size	$\{2^{10}, \dots, 2^{17}\}$
DSR (Petersen, 2021)	Epochs	$\{2^2, \dots, 2^7\}$
NeSymReS (ours)	Beam size	$\{2^0, \dots, 2^8\}$

Conclusion: By amortizing the computation performed at pre-training time, NeSymReS is extremely accurate and efficient at test time, even running on CPU.

(iii) *Performance Improves with more Points p*

In practice, depending on the context, a variable number of input-output pairs might be available at test time. In Figure 25, we report the accuracy achieved for a number of input-output points that varies in the range from 1 to 1024. Even though NeSymReS was pre-trained with no more than 500 points, it still performs reliably with fewer points.

Conclusion: NeSymReS is a flexible method and its performance is robust to different numbers of test data, even when such numbers differ significantly from those usually seen during pre-training. Furthermore, its accuracy levels grow with the number of points observed at test time.

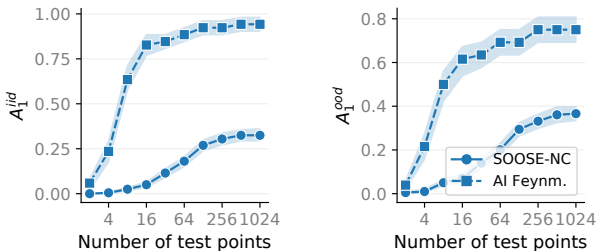


Figure 25: Accuracy as a function of number of input-output pairs observed at test time.

4.6 DISCUSSION

Building on the recent successes of large scale pre-training, we have proposed the first method that *learns the task* of symbolic regression. This approach deviates from the majority of existing techniques in the literature which need to be retrained from scratch on each new equation and does not improve over time with access to data and compute [Sutton, 2019](#). We showed empirically that by pre-training on a large distribution of millions of equations, this simple approach outperforms several strong baselines, and that its performance can be improved by merely increasing the size of the dataset. The key feature that enables this approach is that — unlike for computer vision and natural language — high-quality training data can be generated efficiently and indefinitely using any standard math library and a computer.

In pre-training, the data generation plays a crucial role within our framework. By changing this distribution over equations (including support, constants, number of terms and their interactions), it is possible for the user to finely tune the inductive bias of the model, adapting it to specific applications. In light of its favourable scaling properties and its powerful prior over symbolic expression, we believe that our model could find applications in several domains in the Natural Sciences and engineering, control, and model-based Reinforcement Learning. The scale of our experiments is still relatively small compared to the largest large-scale experiments run to date [Brown et al., 2020](#); [Devlin et al., 2019](#); [Chen et al., 2020c](#), both in terms of dataset and model sizes. Nonetheless, the results we showed already seem to indicate that NeSymReS could improve significantly with access to extremely large scale compute.

TIME AND SPACE COMPLEXITIES The approach we presented scales favorably over several dimensions: computation scales lin-

early in the number of input-output points due to the Set Transformer [Lee et al., 2019](#), and linearly in the number of input dimensions. For future work, it would be interesting to train even larger models on larger datasets with more than three independent variables.

LIMITATIONS Even though our approach can scale to an arbitrary number of input and output dimensions, there are limitations that should be considered. Fitting the constants using a non-linear optimizers like BFGS can prove to be hard if the function to be optimized has several local minima. In this case, other optimization strategies that can deal with non-convex loss surfaces might be beneficial, such as CMA-ES ([Hansen, 2016](#)). One more limitation of our approach is that the pre-trained model as presented cannot be used at test time if the number of input variables is larger than the maximum number of variables seen during pre-training. Finally, one more limitation of the neural network we adopt is that it does not directly interact with the function evaluator available in the math libraries of most computers. If, for example, the first candidate sampled from the network is completely wrong, our current approach cannot adjust its posterior over equations based on this new evidence, but simply sample again.

CONCLUSIONS What are the desirable properties of a strong symbolic regressor? It should:

- *scale* favourably with the number of datapoints observed at test time and with the number of input variables;
- *improve over time* with experience;
- *be targetable* to specific distributions of symbolic equations;
- *be flexible* to accommodate very large or very small values.

In this chapter, we showed that all of these properties can be obtained, and provided a simple algorithm to achieve them in the context of symbolic regression. Our largest pre-trained model can be accessed on our repository.

DIVIDE-AND-CONQUER MONTE CARLO TREE SEARCH

CHAPTER ABSTRACT Standard planners for sequential decision making (including Monte Carlo planning, tree search, dynamic programming, etc.) are constrained by an implicit sequential planning assumption: The order in which a plan is constructed is the same in which it is executed. We consider alternatives to this assumption for the class of goal-directed Reinforcement Learning (RL) problems. Instead of an environment transition model, we assume an imperfect, goal-directed policy. This low-level policy can be improved by a plan, consisting of an appropriate sequence of sub-goals that guide it from the start to the goal state. We propose a planning algorithm, Divide-and-Conquer Monte Carlo Tree Search (DC-MCTS), for approximating the optimal plan by means of proposing intermediate sub-goals which hierarchically partition the initial tasks into simpler ones that are then solved independently and recursively. The algorithm critically makes use of a learned sub-goal proposal for finding appropriate partitions trees of new tasks based on prior experience. Different strategies for learning sub-goal proposals give rise to different planning strategies that strictly generalize sequential planning. We show that this algorithmic flexibility over planning order leads to improved results in navigation tasks in grid-worlds as well as in challenging continuous control environments.

This chapter is based on the paper “*Divide-and-Conquer Monte Carlo Tree Search*”, Giambattista Parascandolo*, Lars Buesing*, Josh Merel, Leonard Hasenclever, John Aslanides, Jessica Hamrick, Nicolas Heess, Alexander Neitz, Theophane Weber (Parascandolo et al., 2021b)

5.1 INTRODUCTION

This is the first sentence of this chapter, but it was not the first one that was written. In fact, the entire introduction section was actually one of the last sections to be added to this chapter. The discrepancy between the order of inception of ideas and the order of their presentation in this chapter probably does not come as a surprise to the reader. Nonetheless, it serves as a point for reflection that is central to the rest of this work, and that can be summarized as “*the order in which we construct a plan does not have to coincide with the order in which we execute it*”.

Most standard planners for sequential decision making problems — including Monte Carlo planning, Monte Carlo Tree Search (MCTS) and dynamic programming — have a baked-in *sequential planning assumption* (Bertsekas et al., 1995; Browne et al., 2012). These methods begin at either the initial or final state and then plan actions sequentially forward or backwards in time. However, this sequential approach faces two main challenges. (i) The transition model used for planning needs to be reliable over long horizons, which is often difficult to achieve when it has to be inferred from data. (ii) Credit assignment to each individual action is difficult: In a planning problem spanning a horizon of 100 steps, to assign credit to the first action, we have to compute the optimal cost-to-go for the remaining problem with a horizon of 99 steps, which is only slightly easier than solving the original problem.

To overcome these two fundamental challenges, here we consider alternatives to the basic assumptions of sequential planners. We focus on goal-directed decision making problems where an agent should reach a goal state from a start state. Instead of a transition and reward model of the environment, we assume a given goal-directed policy (the “low-level” policy) and the associated value oracle that returns its success probability on any given

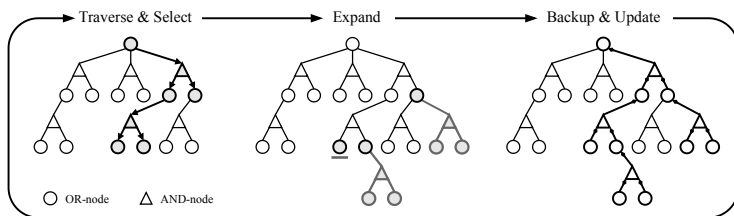


Figure 26: Divide-and-Conquer Monte Carlo Tree Search (DC-MCTS).

task.¹ In general, a low-level policy will not be optimal, e.g. it might be too “myopic” to reliably reach goal states that are far away from its current state. We now seek to improve the low-level policy via a suitable *sequence of sub-goals* that guide it from the start to the final goal, thus maximizing the overall task success probability. This formulation of planning as finding good sub-goal sequences, makes learning of explicit environment models unnecessary, as they are replaced by low-level policies and their value functions.

The sub-goal planning problem can still be solved by a conventional sequential planner that begins by searching for the first sub-goal to reach from the start state, then planning the next sub-goal in sequence, and so on. Indeed, this is the approach taken in most hierarchical RL settings based on options or sub-goals e.g. [Dayan and Hinton, 1993](#); [Sutton et al., 1999](#); [Vezhnevets et al., 2017](#). However, the credit assignment problem mentioned above persists, as assessing if the first sub-goal is useful still requires evaluating the success probability of the remaining plan. Instead, it could be substantially easier to reason about the utility of a sub-goal “in the middle” of the plan, as this breaks the long-horizon problem into two sub-problems with much shorter horizons: how to get to the

¹ As we will observe in Section 5.5, in practice both the low-level policy and value can be learned. Approximating the value oracle with a learned value function was sufficient for DC-MCTS to plan successfully.

sub-goal and how to get from there to the final goal. Based on this intuition, we propose the Divide-and-Conquer MCTS (DC-MCTS) planner that searches for sub-goals to split the original task into two independent sub-tasks of comparable complexity and then recursively solves these, thereby drastically facilitating credit assignment. To search the space of intermediate sub-goals efficiently, DC-MCTS uses a heuristic for proposing promising sub-goals that is learned from previous search results and agent experience.

Humans can plan efficiently over long horizons to solve complex tasks, such as theorem proving or navigation, and some plans even span over decades (e.g. economic measures): In these situations, planning sequentially in terms of next steps – such as what arm to move, or what phone call to make – will cover a tiny proportion of the horizon, neglecting the long uncertainty beyond the last planned step. The algorithm put forward in this chapter is a step in the direction of efficient planners that tackle long horizons by recursively and parallelly splitting them into many smaller and smaller sub-problems. In Section 5.2, we formulate planning in terms of sub-goals instead of primitive actions. In Section 5.3, as our main contribution, we propose the novel Divide-and-Conquer Monte Carlo Tree Search algorithm for this planning problem. In Section 5.4 we position DC-MCTS within the literature of related work. In Section 5.5, we show that it outperforms sequential planners both on grid world and continuous control navigation tasks, demonstrating the utility of constructing plans in a flexible order that can be different from their execution order.

5.2 IMPROVING GOAL-DIRECTED POLICIES WITH PLANNING

Let S and \mathcal{A} be finite sets of states and actions. We consider a multi-task setting, where for each episode the agent has to solve

a new task consisting of a new Markov Decision Process (MDP) \mathcal{M} over \mathcal{S} and \mathcal{A} . Each \mathcal{M} has a single start state s_0 and a special absorbing state s_∞ , also termed the goal state. If the agent transitions into s_∞ at any time it receives a reward of 1 and the episode terminates; otherwise the reward is 0. We assume that the agent observes the start and goal states (s_0, s_∞) at the beginning of each episode, as well as an encoding vector $c_{\mathcal{M}} \in \mathbb{R}^d$. This vector provides the agent with additional information about the MDP \mathcal{M} of the current episode and will be key to transfer learning across tasks in the multi-task setting. A stochastic, goal-directed policy π is a mapping from $\mathcal{S} \times \mathcal{S} \times \mathbb{R}^d$ into distributions over \mathcal{A} , where $\pi(a|s, s_\infty, c_{\mathcal{M}})$ denotes the probability of taking action a in state s in order to get to goal s_∞ . For a fixed goal s_∞ , we can interpret π as a regular policy, here denoted as π_{s_∞} , mapping states to action probabilities. We denote the value of π in state s for goal s_∞ as $v^\pi(s, s_\infty|c_{\mathcal{M}})$; we assume no discounting $\gamma = 1$. Under the above definition of the reward, the value is equal to the success probability of π on the task, i.e. the absorption probability of the stochastic process starting in s_0 defined by running π_{s_∞} :

$$v^\pi(s_0, s_\infty|c_{\mathcal{M}}) = P(s_\infty \in \tau_{s_0}^{\pi_{s_\infty}}|c_{\mathcal{M}}),$$

where $\tau_{s_0}^{\pi_{s_\infty}}$ is the trajectory generated by running π_{s_∞} from state s_0 ². To keep the notation compact, we will omit the explicit dependence on $c_{\mathcal{M}}$ and abbreviate tasks with pairs of states in $\mathcal{S} \times \mathcal{S}$.

5.2.1 Planning over Sub-Goal Sequences

Assume a given goal-directed policy π , which we also refer to as the low-level policy. If π is not already optimal, we can poten-

² We assume MDPs with multiple absorbing states such that this probability is not trivially equal to 1 for most policies, e.g. uniform policy. In experiments, we used a finite episode length.

tially improve it by planning: If π has a low probability of directly reaching s_∞ from the initial state s_0 , i. e. $v^\pi(s_0, s_\infty) \approx 0$, we will try to find a *plan* consisting of a sequence of intermediate sub-goals such that they guide π from the start s_0 to the goal state s_∞ .

Concretely, let $\mathcal{S}^* = \cup_{n=0}^{\infty} \mathcal{S}^n$ be the set of sequences over \mathcal{S} , and let $|\sigma|$ be the length of a sequence $\sigma \in \mathcal{S}^*$. We define for convenience $\bar{\mathcal{S}} := \mathcal{S} \cup \{\emptyset\}$, where \emptyset is the empty sequence representing no sub-goal. We refer to σ as a *plan* for task (s_0, s_∞) if $\sigma_1 = s_0$ and $\sigma_{|\sigma|} = s_\infty$, i. e. if the first and last elements of σ are equal to s_0 and s_∞ , respectively. $s_0 \mathcal{S}^* s_\infty$ denotes the set of plans for this task.

To execute a plan σ , we construct a policy π_σ by conditioning the low-level policy π on each of the sub-goals in order: Starting with $n = 1$, we feed sub-goal σ_{n+1} to π , i. e. we run $\pi_{\sigma_{n+1}}$; if σ_{n+1} is reached, we will execute $\pi_{\sigma_{n+2}}$ and so on. We now wish to do *open-loop planning*, i. e. find the plan with the highest success probability $P(s_\infty \in \tau_{s_0}^{\pi_\sigma})$ of reaching s_∞ . However, this success probability depends on the transition kernels of the underlying MDPs, which might not be known. We can instead define planning as maximizing the following lower bound of the success probability, that can be expressed in terms of the low-level value v^π .

Proposition 3 (Lower bound of success probability). The success probability $P(s_\infty \in \tau_{s_0}^{\pi_\sigma}) \geq L(\sigma)$ of a plan σ is bounded from below by $L(\sigma) := \prod_{i=1}^{|\sigma|-1} v^\pi(\sigma_i, \sigma_{i+1})$, i. e. the product of the success probabilities of π on the sub-tasks defined by (σ_i, σ_{i+1}) .

The straight-forward proof is given in Appendix [D.1.1](#). Intuitively, $L(\sigma)$ is a lower bound for the success of π_σ , as it neglects the probability of “accidentally” (due to stochasticity of the policy or transitions) running into the goal s_∞ before having executed the full plan. We summarize:

Definition 1 (Open-Loop Goal-Directed Planning). Given a goal-directed policy π and its corresponding value oracle v^π , we define

planning as maximizing $L(\sigma)$ over $\sigma \in s_0\mathcal{S}^*s_\infty$, i. e. the set of plans for task (s_0, s_∞) . We define the *high-level (HL) value* $v^*(s_0, s_\infty) := \max_\sigma L(\sigma)$ as the maximum value of the planning objective.

Note the difference between the low-level value v^π and the high-level v^* . $v^\pi(s, s')$ is the probability of the agent *directly* reaching s' from s following π , whereas $v^*(s, s')$ the probability reaching s' from s under *the optimal plan*, which likely includes intermediate sub-goals. In particular, $v^* \geq v^\pi$.

5.2.2 AND/OR Search Tree Representation

In the following we cast the planning problem into a representation amenable to efficient search. To this end, we use the natural compositionality of plans: We can concatenate a plan σ for the task (s, s') and a plan $\hat{\sigma}$ for the task (s', s'') into a plan $\sigma \circ \hat{\sigma}$ for the task (s, s'') . Conversely, we can decompose any given plan σ for task (s_0, s_∞) by splitting it at any sub-goal $s \in \sigma$ into $\sigma = \sigma^l \circ \sigma^r$, where σ^l is the “left” sub-plan for task (s_0, s) , and σ^r is the “right” sub-plan for task (s, s_∞) . Trivially, the planning objective and the optimal high-level value factorize wrt. to this decomposition:

$$\begin{aligned} L(\sigma^l \circ \sigma^r) &= L(\sigma^l)L(\sigma^r) \\ v^*(s_0, s_\infty) &= \max_{s \in \mathcal{S}} v^*(s_0, s) \cdot v^*(s, s_\infty). \end{aligned}$$

This allows us to recursively reformulate planning as:

$$\arg \max_{s \in \mathcal{S}} \left(\arg \max_{\sigma_l \in s_0\mathcal{S}^*s} L(\sigma^l) \right) \cdot \left(\arg \max_{\sigma_r \in s\mathcal{S}^*s_\infty} L(\sigma^r) \right). \quad (6)$$

The above equations are the Bellman equations and the Bellman optimality equations for the classical single pair shortest path problem in graphs, where edge weights are given by $-\log v^\pi(s, s')$. We can represent this planning problem by an AND/OR search

tree (Nilsson, N. J., 1980) with alternating levels of OR and AND nodes. An OR node, also termed an *action node*, is labeled by a task $(s, s'') \in \mathcal{S} \times \mathcal{S}$; the root of the search tree is an OR node labeled by the original task (s_0, s_∞) . A terminal OR node (s, s'') has a value $v^\pi(s, s'')$ attached to it, which reflects the success probability of $\pi_{s''}$ for completing the sub-task (s, s'') . Each non-terminal OR node has $|\mathcal{S}| + 1$ AND nodes as children. Each of these is labeled by a triple (s, s', s'') for $s' \in \mathcal{S}$, which correspond to inserting a sub-goal s' into the overall plan, or not inserting one in case of $s = \emptyset$. Every AND node (s, s', s'') , or *conjunction node*, has two OR children, the “left” sub-task (s, s') and the “right” sub-task (s', s'') .

In this representation, plans are induced by *solution trees*. A solution tree \mathcal{T}_σ is a sub-tree of the complete AND/OR search tree, with the properties that (i) the root $(s_0, s_\infty) \in \mathcal{T}_\sigma$, (ii) each OR node in \mathcal{T}_σ has at most one child in \mathcal{T}_σ and (iii) each AND node in \mathcal{T}_σ has two children in \mathcal{T}_σ . The plan σ and its objective $L(\sigma)$ can be computed from \mathcal{T}_σ by a depth-first traversal of \mathcal{T}_σ . The correspondence of sub-trees to plans is many-to-one, as \mathcal{T}_σ , in addition to the plan itself, contains *the order* in which the plan was constructed. Figure 32 in Section 5.5.3 shows an example for a search and solution tree. Below we will discuss how to construct a favourable search order heuristic.

5.3 BEST-FIRST AND/OR PLANNING

The planning problem from Definition 1 can be solved exactly by formulating it as shortest path problem from s_0 to s_∞ on a fully connected graph with vertex set \mathcal{S} with non-negative edge weights given by $-\log v^\pi$ and applying a classical Single Source or All Pairs Shortest Path (SSSP / APSP) planner. This approach is appropriate if one wants to solve all goal-directed tasks in a *single* MDP. Here, we focus however on the multi-task setting described

Algorithm 3 Divide-and-Conquer MCTS

```

Global low-level value oracle  $v^\pi$ 
Global high-level value function  $v$ 
Global policy prior  $p$ 
Global search tree  $\mathcal{T}$ 

1 procedure TRAVERSE(OR node  $(s, s'')$ )
2   if  $(s, s'') \notin \mathcal{T}$  then
3      $\mathcal{T} \leftarrow \text{EXPAND}(\mathcal{T}, (s, s''))$ 
4     return  $\max(v^\pi(s, s''), v(s, s''))$  ▷ bootstrap
5    $s' \leftarrow \text{SELECT}(s, s'')$  ▷ OR node
6   if  $s' = \emptyset$  or max-depth reached then
7      $G \leftarrow v^\pi(s, s'')$ 
8   else ▷ AND node
9      $G_{\text{left}} \leftarrow \text{TRAVERSE}(s, s')$ 
10     $G_{\text{right}} \leftarrow \text{TRAVERSE}(s', s'')$ 
11    // BACKUP
12     $G \leftarrow G_{\text{left}} \cdot G_{\text{right}}$ 
13   $G \leftarrow \max(G, v^\pi(s, s''))$  ▷ threshold the return
14  // UPDATE
15   $V(s, s'') \leftarrow (V(s, s'')N(s, s'') + G) / (N(s, s'') + 1)$ 
16   $N(s, s'') \leftarrow N(s, s'') + 1$ 
17  return  $G$ 

```

above, where the agent is given a new MDP with a single task (s_0, s_∞) every episode. In this case, solving the SSSP / APSP problem is not feasible: Tabulating all graphs weights $-\log v^\pi(s, s')$ would require $|\mathcal{S}|^2$ evaluations of $v^\pi(s, s')$ for all pairs (s, s') . In practice, approximate evaluations of v^π could be implemented by e.g. actually running the policy π , or by calls to a powerful function approximator, both of which are often too costly to exhaustively evaluate for large state-spaces \mathcal{S} . Instead, we tailor an algorithm for *approximate planning* to the multi-task setting, which we call Divide-and-Conquer MCTS (DC-MCTS). To evaluate v^π as sparsely as possible, DC-MCTS critically makes use of two

learned search heuristics that transfer knowledge from previously encountered MDPs / tasks to new problem instance: (i) a distribution $p(s'|s, s'')$, called the policy prior, for proposing promising intermediate sub-goals s' for a task (s, s'') ; and (ii) a learned approximation v to the high-level value v^* for bootstrap evaluation of partial plans. In the following we present DC-MCTS and discuss design choices and training for the two search heuristics.

5.3.1 *Divide-and-Conquer Monte Carlo Tree Search*

The input to the DC-MCTS planner is an MDP encoding $c_{\mathcal{M}}$, a task (s_0, s_∞) as well as a planning budget, i. e. a maximum number $B \in \mathbb{N}$ of v^π oracle evaluations. At each stage, DC-MCTS maintains a (partial) AND/OR search tree \mathcal{T} whose root is the OR node (s_0, s_∞) corresponding to the original task. Every OR node $(s, s'') \in \mathcal{T}$ maintains an estimate $V(s, s'') \approx v^*(s, s'')$ of its high-level value. DC-MCTS searches for a plan by iteratively constructing the search tree \mathcal{T} with TRAVERSE until the budget is exhausted, see Algorithm 3. During each traversal, if a leaf node of \mathcal{T} is reached, it is expanded, followed by a recursive bottom-up backup to update the value estimates V of all OR nodes visited in this traversal. After this search phase, the currently best plan is extracted from \mathcal{T} by EXTRACTPLAN (essentially depth-first traversal, see Algorithm 5 in the Appendix). In the following we briefly describe the main methods of the search. We illustrate DC-MCTS in Figure 26.

TRAVERSE AND SELECT \mathcal{T} is traversed from the root (s_0, s_∞) to find a promising node to expand. At an OR node (s, s'') , **SELECT** chooses one of its children $s' \in \bar{S}$ to traverse into, including $s = \emptyset$ for not inserting any further sub-goals into this branch. We implemented **SELECT** by the pUCT (Rosin, 2011) rule, which consists of

picking the next node $s' \in \mathcal{S}$ based on maximizing the following score:

$$V(s, s') \cdot V(s', s'') + c \cdot p(s'|s, s'') \cdot \frac{\sqrt{N(s, s'')}}{1 + N(s, s'')}, \quad (7)$$

where $N(s, s')$, $N(s, s', s'')$ are the visit counts of the OR node (s, s') , AND node (s, s', s'') respectively. The first term is the *exploitation* component, guiding the search to sub-goals that currently look promising, i. e. have high estimated value. The second term is the *exploration* term favoring nodes with low visit counts. Crucially, it is explicitly scaled by the policy prior $p(s'|s, s'')$ to guide exploration. At an AND node (s, s', s'') , TRAVERSE traverses into both the left (s, s') and right child (s', s'') .³ As the two sub-problems are solved independently, computation from there on can be carried out in parallel. All nodes visited in a single traversal form a solution tree \mathcal{T}_σ with plan σ .

EXPAND If a leaf OR node (s, s'') is reached during the traversal and its depth is smaller than a given maximum depth, it is expanded by evaluating the high- and low-level values $v(s, s'')$, $v^\pi(s, s'')$. The initial value of the node is defined as max of both values, as by definition $v^* \geq v^\pi$, i. e. further planning should only increase the success probability on a sub-task. We also evaluate the policy prior $p(s'|s, s'')$ for all s' , yielding the proposal distribution over sub-goals used in SELECT. Each node expansion costs one unit of budget B .

BACKUP AND UPDATE We define the return G_σ of the traversal tree \mathcal{T}_σ as follows. Let a refinement \mathcal{T}_σ^+ of \mathcal{T}_σ be a solution tree such that $\mathcal{T}_\sigma \subseteq \mathcal{T}_\sigma^+$, thus representing a plan σ^+ that has all sub-goals of σ with additional inserted sub-goals. G_σ is now defined

³ It is possible to traverse into a single node at the time, we describe several heuristics in Appendix D.1.3

as the value of the objective $L(\sigma^+)$ of the *optimal* refinement of \mathcal{T}_σ , i. e. it reflects how well one could do on task (s_0, s_∞) by starting from the plan σ and refining it. It can be computed by a simple back-up on the tree \mathcal{T}_σ that uses the bootstrap value $v \approx v^*$ at the leafs. As $v^*(s_0, s_\infty) \geq G_\sigma \geq L(\sigma)$ and $G_{\sigma^*} = v^*(s_0, s_\infty)$ for the optimal plan σ^* , we can use G_σ to update the value estimate V . Like in other MCTS variants, we employ a running average operation (line 15-16 in TRAVERSE).

5.3.2 Designing and Training Search Heuristics

Search results and experience from previous tasks can improve DC-MCTS on new problems via adapting the search heuristics, i. e. the policy prior p and the approximate value function v as follows.

BOOTSTRAP VALUE FUNCTION We parametrize $v(s, s' | c_{\mathcal{M}}) \approx v^*(s, s' | c_{\mathcal{M}})$ as a neural network that takes as inputs the current task consisting of (s, s') and the MDP encoding $c_{\mathcal{M}}$. A straight-forward approach to train v is to regress it towards the non-parametric value estimates V computed by DC-MCTS on previous problem instances. However, initial results indicated that this leads to v being overly optimistic, an observation also made in [Kaelbling, 1993](#). We therefore used more conservative training targets, that are computed by backing the low-level values v^π up the solution tree \mathcal{T}_σ of the plan σ return by DC-MCTS. Details can be found in [Appendix D.2.1](#).

POLICY PRIOR Best-first search guided by a policy prior p can be understood as policy improvement of p as described in [Silver et al., 2016](#). Therefore, a straight-forward way of training p is to distill the search results back into the policy prior, e. g. by behavioral cloning. When applying this to DC-MCTS in our setting,

we found empirically that this yielded very slow improvement when starting from an untrained, uniform prior p . This is due to plans with non-zero success probability $L > 0$ being very sparse in \mathcal{S}^* , equivalent to the sparse reward setting in regular MDPs. To address this issue, we propose to apply Hindsight Experience Replay (HER, [Andrychowicz et al., 2017](#)): Instead of training p exclusively on search results, we additionally execute plans σ in the environment and collect the resulting trajectories, i.e. the sequence of visited states, $\tau_{s_0}^{\pi_\sigma} = (s_0, s_1, \dots, s_T)$. HER then proceeds with *hindsight relabeling*, i.e. taking $\tau_{s_0}^{\pi_\sigma}$ as an approximately optimal plan for the “fictional” task (s_0, s_T) that is likely different from the actual task (s_0, s_∞) . In standard HER, these fictitious expert demonstrations are used for imitation learning of goal-directed policies, thereby circumventing the sparse reward problem. We can apply HER to train p in our setting by extracting any ordered triplet $(s_{t_1}, s_{t_2}, s_{t_3})$ from $\tau_{s_0}^{\pi_\sigma}$ and use it as supervised learning targets for p . This is a sensible procedure, as p would then learn to predict optimal sub-goals $s_{t_2}^*$ for sub-tasks $(s_{t_1}^*, s_{t_3}^*)$ under the assumption that the data was generated by an oracle producing optimal plans $\tau_{s_0}^{\pi_\sigma} = \sigma^*$. We have considerable freedom in choosing which triplets to extract from data and use as supervision with HER. In our experiments we use a *temporally balanced* parsing, which creates triplets $(s_t, s_{t+\Delta/2}, s_{t+\Delta})$ such that the resulting policy prior should then preferentially propose sub-goals “in the middle” of the task. In [Appendix D.1.4](#) we discuss this aspect in more detail, and present alternative parsers.

5.3.3 Algorithmic Complexity of DC-MCTS

Denoting an optimal plan as σ^* , the complexity of DC-MCTS with optimal search policy prior $p = p^*$ is $O(|\sigma^*| \cdot |\mathcal{S}|)$. This could potentially be reduced to $O(\log(|\sigma^*|))$ when using progressive widening [Coulom, 2007](#); «[Progressive strategies for monte-carlo](#)

tree search» for fewer evaluations of p and perfect parallelization of tree traversals across multiple workers; for details see Appendix D.1.5.

5.4 RELATED WORK

Goal-directed multi-task learning is an important special case of general RL and has been extensively studied. Universal value functions (Schaul et al., 2015) have been established as compact representation for this setting (Kulkarni et al., 2016; Andrychowicz et al., 2017; Ghosh et al., 2019; Dhiman et al., 2018). This allows to use sub-goals as means for planning, as done in several works such as Kaelbling and Lozano-Pérez, 2017; Gao et al., 2017; Savinov et al., 2018; Stein et al., 2018; Nasiriany et al., 2019, all of which rely on forward sequential planning. Gabor et al., 2019 use MCTS for traditional sequential planning based on heuristics, sub-goals and macro-actions. Zhang et al., 2018 apply traditional graph planners to find abstract sub-goal sequences. We extend this line of work by showing that the abstraction of sub-goals affords more general search strategies than sequential planning. Work concurrent to ours has independently investigated non-sequential sub-goals planning: Jurgenson et al., 2019 propose a top-down policy gradient approach that learns to predict sub-goals in a hierarchical way. Nasiriany et al., 2019 propose gradient-based search jointly over a fixed number of sub-goals for continuous goal spaces. In contrast, DC-MCTS is able to dynamically determine the complexity of the optimal plan.

The proposed DC-MCTS planner is a MCTS (Browne et al., 2012) variant, inspired by recent advances in best-first or guided search, such as AlphaZero (Silver et al., 2018). It can also be understood as a heuristic, guided version of the classic Floyd-Warshall algorithm which exhaustively computes all shortest paths. In the special case of planar graphs, small sub-goal sets, also known as

vertex separators, can be constructed that favourably partition the remaining graph, leading to linear time ASAP algorithms (Hen-zinger et al., 1997). The heuristic sub-goal proposer p that guides DC-MCTS can be loosely understood as a probabilistic version of a vertex separator. Nowak et al., 2018 also consider neural networks that mimic divide-and-conquer algorithms similar to the sub-goal proposals used here. However, while we do policy improvement for the proposals using search and HER, the networks in Nowak et al., 2018 are purely trained by policy gradient methods.

Decomposing tasks into sub-problems has been formalized as pseudo trees (Freuder and Quinn, 1985) and AND/OR graphs (Nilsson, N. J., 1980). The latter have been used especially in the context of optimization (Larrosa et al., 2002; Jégou and Terrioux, 2003; Dechter and Mateescu, 2004; Marinescu and Dechter, 2004). Our approach is related to work on using AND/OR trees for sub-goal ordering in the context of logic inference (Ledeniou and Markovitch, 1998). While DC-MCTS is closely related to the AO^* algorithm (Nilsson, N. J., 1980), which is the generalization of the heuristic A^* search to AND/OR search graphs, interesting differences exist: AO^* assumes a fixed search heuristic, which is required to be lower bound on the cost-to-go. In contrast, we employ learned value functions and policy priors that are not required to be exact bounds. Relaxing this assumption, thereby violating the principle of “optimism in the face of uncertainty”, necessitates explicit exploration incentives in the SELECT method. Alternatives for searching AND/OR spaces include proof-number search, recently applied to chemical synthesis planning (Kishimoto et al., 2019). Very recent work concurrent to ours has focused on relevant research directions: Wang et al., 2020 introduce LA-MCTS as a ‘meta-algorithm’ for black-box optimization, and Chen et al., 2020a propose Retro*, a neural-based A^* -like algorithm for molecule synthesis that is also based on AND/OR trees.

5.5 EXPERIMENTS

We evaluate the proposed DC-MCTS algorithm on navigation in grid-world mazes as well as on a challenging continuous control version of the same problem, comparing it to standard sequential MCTS (in sub-goal space) based on the fraction of “solved” mazes by executing their plans. The MCTS baseline was implemented by restricting the DC-MCTS algorithm to only expand the “right” sub-problem in line 10 of Algorithm 3; the value G_{left} for the “left” sub-problem is computed as in line 7, i.e. using the low-level value v^π . This forces MCTS to plan forward and sequentially, as each next step needs to be reachable from the previous state, as evaluated by v^π . All remaining parameters and design choice were the same for both planners except where explicitly mentioned otherwise.

5.5.1 Grid-World Mazes

Each task consists of a new, procedurally generated maze on a 21×21 grid with start and goal locations $(s_0, s_\infty) \in \{1, \dots, 21\}^2$, see Figure 27 and Figure 28. Task difficulty was controlled by the density of walls d (under connectedness constraint), where the easiest setting $d = 0.0$ corresponds to no walls and the most difficult one $d = 1.0$ implies so-called perfect or singly-connected mazes. The task embedding $c_{\mathcal{M}}$ was given as the maze layout and (s_0, s_∞) encoded together as a feature map of 21×21 categorical variables with 4 categories each (empty, wall, start and goal location). The underlying MDPs have 5 primitive actions: up, down, left, right and NOOP. For sake of simplicity, we first tested our proposed approach by hard-coding a low-level policy π^0 as well as its value oracle v^{π^0} in the following way. If in state s and conditioned on a goal s' , and if s is adjacent to s' , $\pi_{s'}^0$ successfully reaches s' with probability 1 in one step, i.e. $v^{\pi^0}(s, s') = 1$;

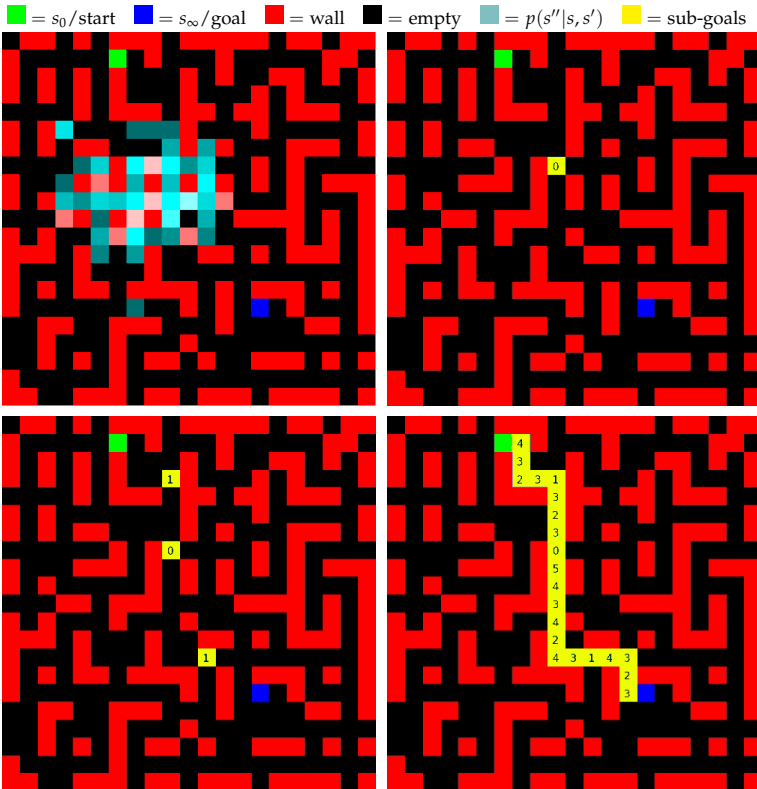


Figure 27: A grid-world maze examples for wall density 0.75. *Top-left*: In light blue, the distribution over sub-goals induced by the policy prior p that guides the DC-MCTS planner. *Top-right* \rightarrow *Bottom-left* \rightarrow *Bottom-right*: The first sub-goal, i.e. at depth 0 of the solution tree, approximately splits the problem in half. Next, the *two* sub-goals at depth 1. Last, the final plan with the depth of each sub-goal shown. See supplementary material for full animations.

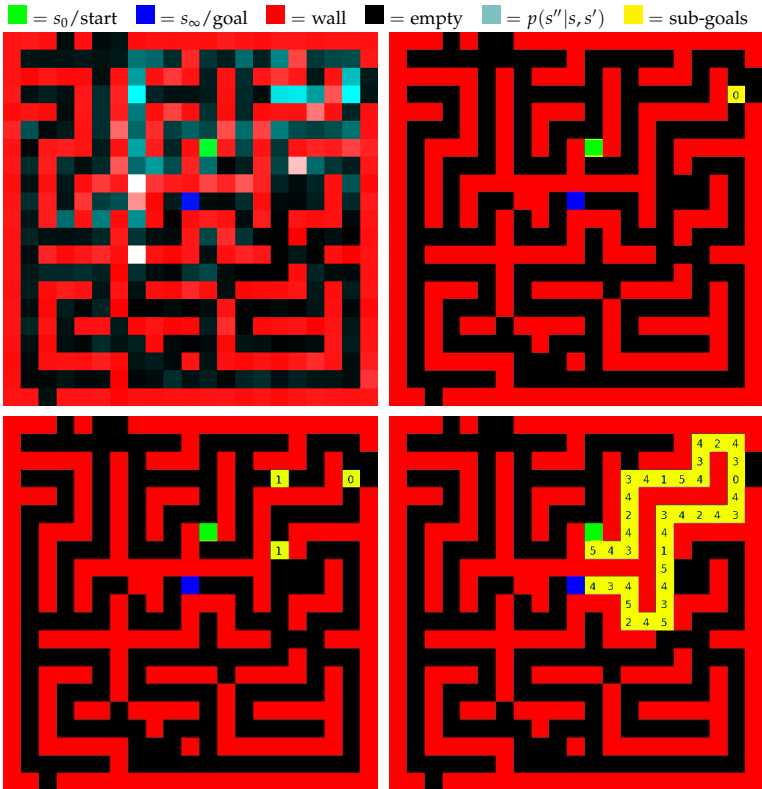


Figure 28: A grid-world maze examples for wall density 0.95. *Top-left*: In light blue, the distribution over sub-goals induced by the policy prior p that guides the DC-MCTS planner. *Top-right* \rightarrow *Bottom-left* \rightarrow *Bottom-right*: The first sub-goal, i.e. at depth 0 of the solution tree, approximately splits the problem in half. Next, the *two* sub-goals at depth 1. Last, the final plan with the depth of each sub-goal shown. See supplementary material for full animations.

otherwise $v^{\pi^0}(s, s') = 0$. If $\pi_{s'}^0$ is nevertheless executed, the agent moves to a random empty tile adjacent to s . Therefore, π^0 is the “most myopic” goal-directed policy that can still navigate everywhere.

For each maze, MCTS and DC-MCTS were given a search budget of 200 calls to the low-level value oracle v^{π^0} . We implemented the search heuristics, i. e. policy prior p and high-level value function v , as convolutional neural networks (CNNs) which operate on input $c_{\mathcal{M}}$; details for the network architectures are given in Appendix D.2.3. With untrained networks, both planners were unable to solve the task (<2% success probability), as shown in Figure 29. This illustrates that a search budget of 200 evaluations of v^{π^0} is insufficient for unguided planners to find a feasible path in most mazes. This is consistent with standard exhaustive SSSP / APSP graph planners requiring $21^4 > 10^5 \gg 200$ evaluations for optimal planning in the worst case on these tasks.

Next, we trained both search heuristics v and p as detailed in Section 5.3.2. In particular, the sub-goal proposal p was also trained on hindsight-reabeled experience data, where for DC-MCTS we used the *temporally balanced parser* and for MCTS the corresponding left-first parser (see Appendix D.1.4). Training of the heuristics greatly improved the performance of both planners. Figure 29 shows learning curves for mazes with wall density $d = 0.75$, as mean and std over 20 different hyperparameters. DC-MCTS exhibits substantially improved performance compared to MCTS, and when compared at equal performance levels, DC-MCTS requires 5 to 10-times fewer training episodes than MCTS. An example of a learned sub-goal proposal p for DC-MCTS is visualized in Figure 27 and Figure 28 (further examples are given in the Appendix in Figure 48). Probability mass concentrates on promising sub-goals that are far from both start and goal, approximately partitioning the task into equally hard sub-tasks.

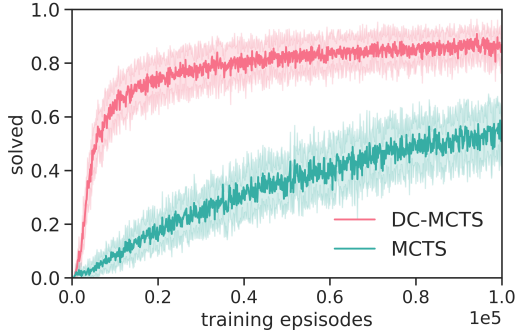


Figure 29: Accuracy on grid-world mazes.

Next, we investigated the performance of both MCTS and DC-MCTS in challenging continuous control environments with non-trivial low-level policies. We embedded the grid-world mazes into a physical 3D environment simulated by MuJoCo (Todorov et al., 2012), rendering each grid-world cell as $4\text{m} \times 4\text{m}$ cell in physical space. The agent is embodied by a quadruped “ant” body; for illustration see Figure 30. For the low-level policy π^m , we pre-trained a goal-directed neural network controller that gets as inputs proprioceptive features (e.g. some joint angles and velocities) of the ant body as well as a 3D-vector pointing from its current position to a target position. π^m was trained to navigate to targets randomly placed less than 1.5 m away in an open area (no walls), using MPO (Abdolmaleki et al., 2018). See Appendix D.2.4 for more details. If unobstructed, π^m can walk in a straight line towards its current goal. However, this policy receives no visual input and thus can only avoid walls when guided with appropriate sub-goals. To establish an interface between the low-level π^m and the planners, we used another CNN to approximate the low-level value oracle $v^{\pi^m}(s_0, s_\infty | c_{\mathcal{M}})$: It was trained to predict whether π^m



Figure 30: The ‘ant’, i.e. the agent, should navigate to the green target.

will succeed in solving the navigation tasks $(s_0, s_\infty), c_M$. Its input is the corresponding discrete grid-world representation c_M of the maze (21×21 feature map of categoricals as described above, details in Appendix). Note that this setting is still challenging: In initial experiments we verified that a model-free baseline (also based on MPO, without HER) with access to state abstraction and low-level controller, only solved about 10% of the mazes after 100 million episodes due to the extremely sparse rewards.

5.5.2 Continuous Control Mazes

We applied MCTS and DC-MCTS to this problem to find symbolic plans consisting of sub-goals in $\{1, \dots, 21\}^2$. The high-level heuristics p and v were trained for 65k episodes, exactly as in Section 5.5.1, except using v^{π^m} instead of v^{π^0} . We again observed that DC-MCTS outperforms by a wide margin the MCTS planner: Figure 31 shows performance of both (with fully trained search

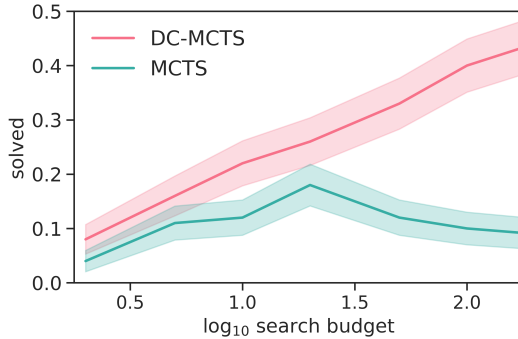


Figure 31: Fraction of solved mazes vs. planning budget.

heuristics) as a function of the search budget for the most difficult mazes with wall density $d = 1.0$. Performance of DC-MCTS with the MuJoCo low-level controller was comparable to that with the hard-coded low-level policy from the grid-world experiment (with same wall density), showing that the abstraction of planning over low-level sub-goals successfully isolates high-level planning from low-level execution. We did not manage to successfully train the MCTS planner on MuJoCo navigation.

This was likely due to HER, which we found — in ablation studies — essential for training DC-MCTS on both settings and MCTS on the grid-world problem, but not appropriate for MCTS on MuJoCo navigation: Left-first parsing for HER consistently biased the MCTS search prior p to propose next sub-goals too close to the previous sub-goal. This led the MCTS planner to “micro-manage” the low-level policy, in particular in long corridors that π^m can solve by itself. DC-MCTS, by recursively partitioning, found an appropriate length scale of sub-goals, leading to drastically improved performance.

5.5.3 Visualizing MCTS and DC-MCTS

To further illustrate the difference between DC-MCTS and MCTS planning we can look at an example search tree from each method in Figure 32. Light blue nodes are part of the final plan: note how in the case of DC-MCTS, the plan is distributed across a *sub-tree* within the search tree, while for the standard MCTS the plan is a *chain*. The first ‘actionable’ sub-goal, i.e. the first sub-goal for the low-level policy, is the left-most leaf in DC-MCTS and the first dark node from the root for MCTS.

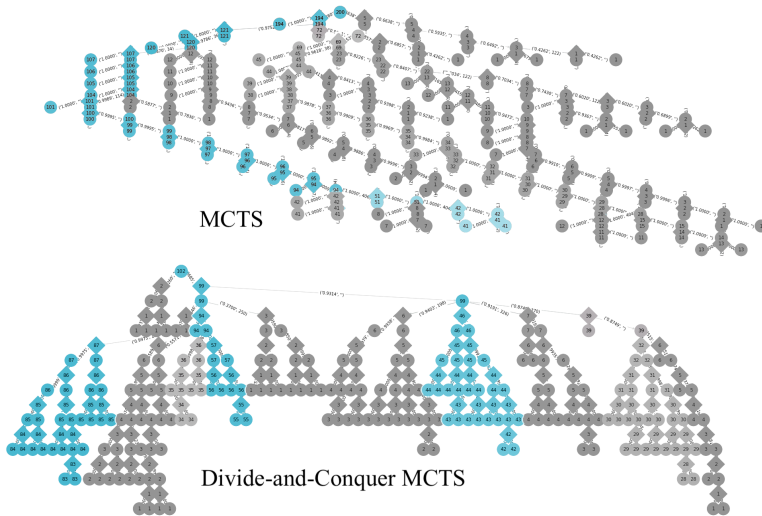


Figure 32: Only colored nodes are part of the final plan: a *sub-tree* for DC-MCTS, a *chain* for MCTS.

5.6 DISCUSSION

To enable guided, divide-and-conquer style planning, we made a few strong assumptions. Sub-goal based planning requires a universal value function oracle of the low-level policy, which often will have to be approximated from data. Overly optimistic approximations can be exploited by the planner, leading to “delusional” plans (Little and Thiébaux, 2007). Joint learning of the high and low-level components can potentially address this issue. In sub-goal planning, at least in its current naive implementation, the “action space” for the planner is the whole state space of the underlying MDPs. Therefore, the search space will have a large branching factor in large state spaces. A solution to this problem likely lies in using learned state abstractions for sub-goal specifications, which is a fundamental open research questions. We also implicitly assumed that low-level skills afforded by the low-level policy need to be “universal”, i.e. if there are states that it cannot reach, no amount of high level search will lead to successful planning outcomes.

In spite of these assumptions and open challenges, we showed that non-sequential sub-goal planning has fundamental advantages over the standard approach of search over primitive actions: (i) *Abstraction and dynamic allocation*: Sub-goals automatically support temporal abstraction as the high-level planner does not need to specify the exact time horizon required to achieve a sub-goal. Plans are generated from coarse to fine, and additional planning is dynamically allocated to those parts of the plan that require more compute. (ii) *Closed & open-loop*: The approach combines advantages of both open- and closed loop planning: The closed-loop low-level policies can recover from failures or unexpected transitions in stochastic environments, while at the same time the high-level planner can avoid costly closed-loop planning. (iii) *Long horizon credit assignment*: Sub-goal abstractions open up new al-

gorithmic possibilities for planning — as exemplified by DC-MCTS — that can facilitate credit assignment and therefore reduce planning complexity. (iv) *Parallelization*: Like other divide-and-conquer algorithms, DC-MCTS lends itself to parallel execution by leveraging problem decomposition made explicit by the independence of the "left" and "right" sub-problems of an AND node. (v) *Reuse of cached search*: DC-MCTS is highly amenable to transposition tables, by caching and reusing values for sub-problems solved in other branches of the search tree. (vi) *Generality*: DC-MCTS is strictly more general than both forward and backward goal-directed planning, both of which can be seen as special cases.

CONCLUSION

6.1 RECAP

Out-of-distribution generalization is one of the main open challenges for artificial intelligence, and in particular for deep learning (Bengio et al., 2019; Schölkopf et al., 2021). As discussed in the introduction, for o.o.d. generalization to be a meaningful objective, we need to make assumptions. In this thesis, we investigated *four* different aspects of out-of-distribution generalization with deep learning, contributing novel algorithms and insights to approach this setting.

The first chapter relied on the notion of *compositionality*. We built on the concept of independence of mechanisms from the literature on causality (Schölkopf et al., 2012; Peters et al., 2017b). Under this assumption of independence, we introduced an algorithm based on a *competition of experts*, which results in a set of independent, reusable modules. We showed in a simple experiment on a standard image dataset, how these modules can then be re-composed at test time, and generalize to novel combination of mechanisms unobserved during training.

The second assumption focused on *invariances*. Assuming that some of the features represent stable mechanisms in the data, a model can generalize out-of-distribution if it learns to rely solely on such invariant features. We showed that training based on empirical risk minimization with gradient descent, using the arithmetic mean of gradients, may lead neural networks to rely on spurious features by minimizing the loss based on every pattern detected. Using the geometric mean of gradients instead, one can

trade off convergence time for invariances, and we derived a simple algorithm called *AND-mask* that builds on this principle. This allows the models to generalize out-of-distribution, and helps prevent memorization in networks with a large capacity.

The third aspect investigated *symbolic equations* as a means to generalize o.o.d., as symbolic equations are at the core of scientific discovery. Instead of hand-designing a symbolic regression algorithm, we aimed at *learning* one, end-to-end, by building on recent successes of large-scale pre-training. We showed that our proposed approach significantly outperforms state-of-the-art algorithms, and most importantly improves over time with experience.

The last chapter focused on planning, i.e. o.o.d. generalization by investing *additional compute* at test time. We presented an algorithm to train an agent capable of goal-directed planning over long horizons. By incorporating a divide-and-conquer approach into Monte Carlo Tree Search (DC-MCTS), our agent learns to solve increasingly more complex problems over long horizons by recursively subdividing them where the uncertainty is higher. Jumping back and forth in time, each additional level of planning depth allows for an exponential decrease of the horizon lengths, trading off for a larger number of (progressively shorter) intervals to estimate.

6.2 OTHER AXES OF O.O.D. GENERALIZATION

There are other axes of o.o.d. generalization that we did not look into. The four different assumptions we investigated all belong to the same overall family of o.o.d. generalization, i.e. where the input data distribution $P_{train}(X) \neq P_{test}(X)$, and even more, the $\text{supp}(P_{train}(X)) \cap \text{supp}(P_{test}(X)) = \emptyset$, but crucially the mechanism $P(Y|X)$ stays the same (Peters et al., 2017b).

Another possibility, sometimes referred to as *concept drift*, is that $P(Y|X)$ changes from training to test. This can occur e.g. as a certain dynamic evolves over time, due to physical or normative changes. In this setting zero-shot o.o.d. generalization cannot be expected without additional knowledge regarding the change in the mechanism, or additional training data that allows to identify it. For a comprehensive overview of the intersection between causality and learning for out-of-distribution generalization I recommend the recent work by [Schölkopf et al., 2021](#).

6.3 I.I.D. OR O.O.D.? A DISTINCTION BLURRED BY SCALE.

In Chapter 4 our neural symbolic regressor NeSymReS was pre-trained on hundreds of millions of equations from a massive distribution. The pre-training distribution is in fact so wide and generic, that almost every well-formed equation has a non-zero probability under the training distribution. When the model correctly recovers an equation that was not in the training set (but inevitably in the training distribution, given how wide that is), it is technically an instance of i.i.d. generalization. But is there even any potential for o.o.d. generalization left? If the training distribution already contains all test examples we might ever be interested in, the distinction between i.i.d. and o.o.d. becomes perhaps redundant.

Two more examples from the recent literature on large-scale pre-training are GPT-3 ([Brown et al., 2020](#)) and DALL-E ([Ramesh et al., 2021](#)). GPT-3 is a language model trained to predict the next token (i.e. roughly a syllable) conditioned on previous tokens, on a corpus of ≈ 500 billion tokens. DALL-E is a generative model that produces images conditioned on a text prompt, and it was pre-trained on 250 million image-text pairs. Both models showed impressive generalization capabilities, generating examples with novel combinations that did not appear in their respective train-

ing sets. It is also remarkable that a model like GPT-3 is a powerful zero- or few-shot learner, despite having no explicit meta-learning objective during training. Again, the line between i.i.d. and o.o.d. is blurred: on the one hand, everything could be now seen as interpolation; on the other hand, it becomes challenging to even find meaningful tasks within these domains that could unambiguously count as extrapolation.

The reason why it is important to ask this question, is that we started this thesis by emphasizing that one of the (currently) unmatched features of human intelligence is its o.o.d. generalization capability. We should not forget, however, that there exists a distribution of tasks we are interested in solving, which is admittedly a very large one. If all we ultimately care about is the model performance on this distribution, *how* we get to such a general agent should not matter, even if it might turn out to be by training on that very same distribution. If the scale of models and data is going to keep increasing, to the point where any relevant test task will be within the training distribution, the focus might shift from questions of i.i.d. vs o.o.d. generalization, to generalization. At this point in time, it is of course unclear whether the current methods simply applied at larger scales will be sufficient to solve any meaningful task. Time (and compute) will tell.

6.4 ON SAMPLE-(IN)EFFICIENCY OF HUMAN AND MACHINE INTELLIGENCE

Finally, I want to take the chance to present a perspective on sample efficiency in deep learning and human intelligence. Large scale training has certainly proven to be a powerful tool, but it has also been the target of skepticism within part of the research community, in particular in the context of deep neural networks as models of human intelligence. The argument that is often made

against these models is that they are *too sample-inefficient* compared to human intelligence. I believe this argument is incorrect.

We often think about learning across multiple timescales. For humans we typically think at least of two: evolution (acting on the DNA) and learning during a lifetime (mostly acting on synaptic plasticity). Two corresponding mechanisms are sometimes seen in artificial neural nets: the iterations that researchers do on architectures roughly play the role of evolution (e.g. Perceptron \rightarrow AlexNet \rightarrow ResNets \rightarrow ...), while the training of neural nets on datasets corresponds to the learning that happens in a lifetime. This analogy for gradient descent is further supported by the common interpretation of back-propagation and gradient descent as learning by slowly adjusting synaptic weights as biological neurons do. If we embraced this classic view of synaptic plasticity, we would have to conclude that neural networks are indeed very sample inefficient compared to human brains within a lifetime.¹

But an alternative view is possible: *pre-training via gradient descent on network weights is not always analogous to the learning that occurs within the lifetime of the individual (i.e. synaptic plasticity), but acts at the same level as evolution.* Note that we are not saying here that gradient descent and evolution are mechanisms that act in a similar way, in fact, they are almost as different as it gets when it comes to *how* they work. Here we are saying that the *role* of training with gradient descent could be seen as equivalent to the *role* evolution had for us: distilling the right inductive biases such that we can then learn in a sample-efficient way. Evolution was a costly and incredibly sample-inefficient process that led to us, and pre-training can be a fast-forward and streamlined alternative to evolution.

For example, while it is true that GPT-3 has been trained on more text than any human could process in a lifetime, a human

¹ For example, current state-of-the-art vision models are pre-trained on up to 3 billion images (Zhai et al., 2021).

of today has implicitly read, heard, and spoken, billions of words through the eyes, ears, and mouths of their ancestors. And indeed, GPT-3 at test time can learn very efficiently without any fine-tuning of the weights. We could look at GPT-3 massive pre-training as serving a similar role to the role evolution had for humans in the development of structures for language in our brain.

Sample efficiency on 'meaningful' tasks is one of the main axes to measure general intelligence, but the *process* that gets to sample-efficient agents *does not need to be sample efficient itself*, just as evolution was not sample-efficient in getting to us.

APPENDIX LICM

A.1 ADDITIONAL RESULTS.

A.1.1 *Too many or too few experts.*

TOO MANY EXPERTS When there are too many experts, for most tasks only one wins all the examples, as shown in Figure 33 where the model has 16 experts for 10 tasks. In this case the remaining experts do not specialize at all and therefore can be removed from the architecture. Had several experts specialized on the same task, they could be combined after determining that they perform the same task. Since the accuracy on the transformed data tested on the pretrained classifier reaches again the upper-bound of the untransformed data, and since the progress is very similar to that illustrated in Figure 6, we omit this plot.

TOO FEW EXPERTS For a committee of 6 experts, the networks do not reconstruct properly most of the digits, which is reflected by an overall low objective function value on the data. Also, the accuracy achieved by the pretrained MNIST classifier does not exceed 72%. A few experts are inevitably assigned to multiple tasks, and by looking at Figure 33 it is interesting to see that the clustering result is still meaningful (e.g. expert 5 is assigned to left, down-left, and up-left translation).

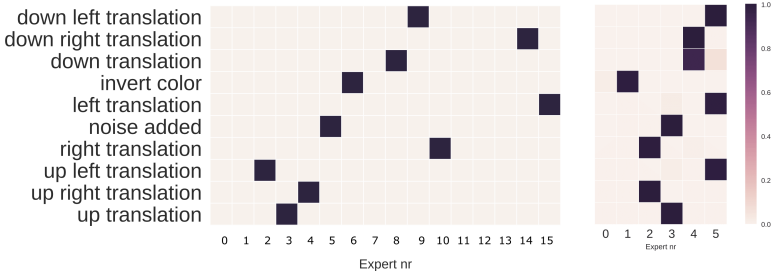


Figure 33: The proportion of data won by each expert for each transformation on the digits from the test set, for the case of 10 mechanisms and more experts (16 on left) or too few (6 on the right). Note how on the left experts 0, 1, 7, 11, 12, 13, do not win any data points, and can therefore be discarded.

A.2 DETAILS OF NEURAL NETWORKS

In Table 2 we report the configuration of the neural networks used in these experiments.

For the approximate identity initialization we train each network for a maximum of 500 iterations, or until the mean squared error of the reconstructed images is below 0.002.

A.3 TRANSFORMATIONS

In our experiments we use the following transformations

- Translations: the image is shifted by 4 pixels in one of the eight directions up, down, left, right and the four diagonals.
- Contrast (or color) inversion: the value of each pixel — originally in the range $[0, 1]$ — is recomputed as $1 -$ the original value.

Table 2: Architectures of the neural networks used in the experiment section. BN stands for Batch normalization, FC for fully connected. All convolutions are preceded by a 1 pixel zero padding.

Expert	Discriminator
Layers	Layers
$3 \times 3, 32, \text{BN}, \text{ELU}$	$3 \times 3, 16, \text{ELU}$
$3 \times 3, 32, \text{BN}, \text{ELU}$	$3 \times 3, 16, \text{ELU}$
$3 \times 3, 32, \text{BN}, \text{ELU}$	$3 \times 3, 16, \text{ELU}$
$3 \times 3, 32, \text{BN}, \text{ELU}$	$2 \times 2, \text{avg pooling}$
$3 \times 3, 32, \text{BN}, \text{ELU}$	$3 \times 3, 32, \text{ELU}$
$3 \times 3, 32, \text{BN}, \text{ELU}$	$3 \times 3, 32, \text{ELU}$
$3 \times 3, 1, \text{sigmoid}$	$2 \times 2, \text{avg pooling}$
	$3 \times 3, 64, \text{ELU}$
	$3 \times 3, 64, \text{ELU}$
	$2 \times 2, \text{avg pooling}$
	$1024, \text{FC}, \text{ELU}$
	$1, \text{FC}, \text{sigmoid}$

- Noise addition: random Gaussian noise with zero mean and variance 0.25 is added to the original image, which is then clamped again to the $[0, 1]$ interval.

A.4 NOTES ON THE FORMALIZATION OF INDEPENDENCE OF MECHANISMS

In this section we briefly discuss the notion of independence of mechanisms as in [Janzing and Schölkopf, 2010](#), where the independence principle is formalized in terms of algorithmic complexity (also known as Kolmogorov complexity). We summarize the main points needed in the present context. We parametrize each *mechanism* by a bit string x . The Kolmogorov complexity $K(x)$ of

x is the length of the shortest program generating x on an a priori chosen universal Turing machine. The **algorithmic mutual information** can be defined as $I(x : y) := K(x) + K(y) - K(x, y)$, and it can be shown to equal

$$I(x : y) = K(y) - K(y|x^*), \quad (8)$$

where for technical reasons we need to work with x^* , the shortest description of x (which is in general uncomputable). Here, the conditional Kolmogorov complexity $K(y|x)$ is defined as the length of the shortest program that generates y from x . The algorithmic mutual information measures the algorithmic information two objects have in common. We define two mechanisms to be **(algorithmically) independent** whenever the length of the shortest description of the two bit strings together is not shorter than the sum of the shortest individual descriptions (note it cannot be longer), i.e., if their algorithmic mutual information vanishes.¹ In view of equation 8, this means that

$$K(y) = K(y|x^*). \quad (9)$$

We will say that two mechanisms x and y are independent whenever the complexity of the conditional mechanism $y|x$ is comparable to the complexity of the unconditional one y . If, in contrast, the two mechanisms were closely related, then we would expect that we can mimic one of the mechanisms by applying the other one followed by a low complexity conditional mechanism.

¹ All statements are valid up to additive constants, linked to the choice of a Turing machine which produces the object (bit string) when given its compression as an input. For details, see Janzing and Schölkopf (2010).

APPENDIX ILC

B.1 APPENDIX TO SECTION 3.2

B.1.1 A classic example of a patchwork solution

Consider a neural network with one hidden layer consisting of two neurons and sigmoidal activations:

$$f_{\theta}(x) = \theta_5 \sigma(\theta_1 x + \theta_2) + \theta_6 \sigma(\theta_3 x + \theta_4), \quad \sigma(z) := 1/(1 + e^{-z}). \quad (10)$$

We want to learn the continuous function $f^* : [0, 1] \rightarrow [0, 2]$ defined as

$$f^*(x) = \begin{cases} 0 & x \in [0, 0.4); \\ 10(x - 0.4) & x \in [0.4, 0.5); \\ 1 & x \in [0.5, 0.7); \\ 10(x - 0.7) + 1 & x \in [0.7, 0.8); \\ 2 & x \in [0.8, 1]. \end{cases}$$

To perform this task, we have access to (noiseless) data from two environments:

$$A : \{(x, f(x)) \mid x \in [0, 0.5)\}, \quad B : \{(x, f(x)) \mid x \in [0.5, 1]\}.$$

There is a simple *constructive* way, provided by the universal function approximation theorem [Cybenko, 1989](#) to fit this function¹ using f_θ up to an arbitrarily small mean squared error $\mathcal{L}_{A+B}(\theta^*)$. Leaving out the details of such a construction ([Cybenko, 1989](#) for details), the reader can check on the left panel of Figure 34 that $\theta^* = (100, -50, 100, -75, 1, 1)$ provides a good fit for *both environments* A and B — both $\mathcal{L}_A(\theta^*)$ and $\mathcal{L}_B(\theta^*)$ are small.

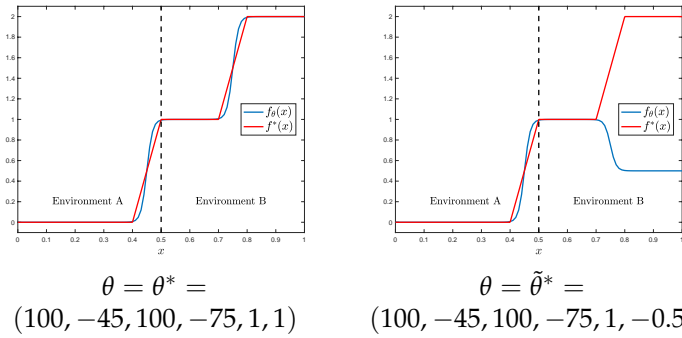


Figure 34: Performance of the neural network in Equation 10 for two different parameters. Any reasonable modification on θ_6 (say ± 1) leaves the performance on environment A unchanged, while the performance on environment B quickly degrades.

However, it is easy to realize that θ^* — while being a solution which can be returned by gradient descent using the pooled data $A+B$ — is not *consistent* (formal definition given in the main paper in Section 3.2). Indeed, it is possible to modify $\tilde{\theta}^*$ such that the loss in environment A remains almost unchanged, while the loss in environment B gets larger. In particular, on the right panel of Figure 34, we show that $\tilde{\theta}^* = (100, -50, 100, -75, 1, -0.5)$ is such that $\mathcal{L}_A(\theta^*) \leq \mathcal{L}_A(\tilde{\theta}^*) + \epsilon$ (with ϵ very small) but $\mathcal{L}_B(\theta^*) \ll$

¹ For a graphical description, the reader can check <http://neuralnetworksanddeeplearning.com/chap4.html>

$\mathcal{L}_B(\tilde{\theta}^*)$. According to our definition in Equation 4 (see main paper), we have $\mathcal{I}^\epsilon(\theta^*) \leq |\mathcal{L}_B(\theta^*) - \mathcal{L}_B(\tilde{\theta}^*)|$ — that is a large number (low consistency).

Remark 1 (Connection to out of distribution generalization). The main point of this analysis was to show an example of where our measure of *consistency* behaves according to expectations: A typical implementation of the universal approximation theorem — which one would *not* expect to generalize out of distribution, due to its ‘*patchwork*’ behavior — leads indeed to a very low consistency score.

B.1.2 Section 3.2.2: Consistency as arithmetic/geometric mean of landscapes

GEOMETRIC MEAN OF MATRICES. Given an n -tuple of $d \times d$ positive definite matrices $(A_j)_{j=1}^n$, the geometric (also called Karcher) mean [Ando et al., 2004](#) is the unique positive definite solution X to the equation $\sum_{i=1}^m \log(A_i^{-1}X) = 0$, where \log is the matrix logarithm. This matrix average has many desirable properties, which make it relevant to signal processing and medical imaging. The Karcher mean can also be written as $\arg \min_{X \in \mathcal{S}^{++}(d)} f(X) = \frac{1}{2m} \sum_{i=1}^m d(A_i, X)^2$, where d is the Riemannian distance in the manifold of SPD matrices $\mathcal{S}^{++}(d)$.

LINK BETWEEN CONSISTENCY AND GEOMETRIC MEANS. Here we show how the consistency score introduced in Equation 4 can be linked (in a simplified setting) to a comparison between the arithmetic and geometric means of the Hessians approximating the landscapes of two separate environments A and B .

At the local minimizer $\theta^* = 0$, we assume that $\mathcal{L}_A = \mathcal{L}_B = 0$ and consider the local quadratic approximations $\mathcal{L}_A(\theta) = \frac{1}{2}\theta^\top H_A\theta$ and $\mathcal{L}_B(\theta) = \frac{1}{2}\theta^\top H_B\theta$. Here, we make the additional simplifying assumption that H_A and H_B are diagonal (or, more broadly, co-

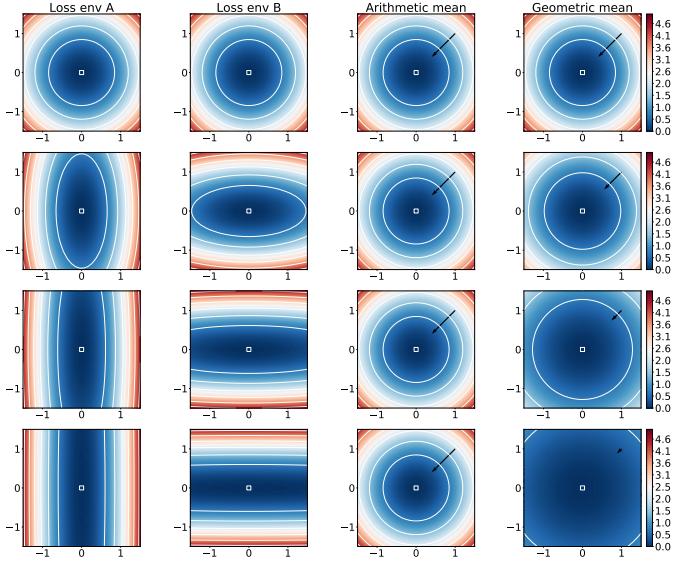


Figure 35: While the arithmetic mean of the two loss surfaces on the left is identical in all three cases (third column), the geometric mean has weaker and weaker gradients (black arrow) the more inconsistent the two loss surfaces become.

diagonalizable): $H_A = \text{diag}(\lambda_1^A, \dots, \lambda_n^A)$, $H_B = \text{diag}(\lambda_1^B, \dots, \lambda_n^B)$, with $\lambda_i^A \geq 0$ and $\lambda_i^B \geq 0$ for all $i = 1, \dots, n$. The *arithmetic* and *geometric* means (noted as H_{A+B} and $H_{A \wedge B}$) of these matrices are defined in this simplified setting as follows:

$$\begin{aligned}
 H_{A+B} &= \text{diag} \left(\frac{1}{2}(\lambda_1^A + \lambda_1^B), \dots, \frac{1}{2}(\lambda_n^A + \lambda_n^B) \right), \\
 H_{A \wedge B} &= \text{diag} \left(\sqrt{\lambda_1^A \lambda_1^B}, \dots, \sqrt{\lambda_n^A \lambda_n^B} \right).
 \end{aligned}
 \tag{11}$$

As motivated in the main paper and in Figure 36, one can link the consistency of two landscapes to a comparison between the geometric and arithmetic means of the corresponding Hessians.

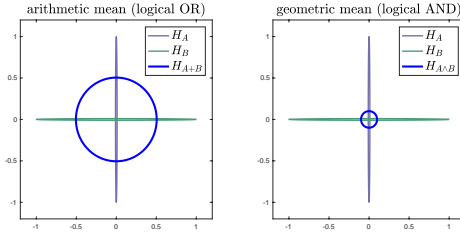


Figure 36: Plotted are contour lines $\theta^\top H^{-1}\theta = 1$ for $H_A = \text{diag}(0.01, 1)$ and $H_B = \text{diag}(1, 0.01)$. It is convenient to provide this visualization because it is linked to the matrix determinant: $\text{Vol}(\{\theta^\top H^{-1}\theta = 1\}) = \pi\sqrt{\det(H)}$. The geometric average retains the volume of the original ellipses, while the volume of H_{A+B} is 25 times bigger. This magnification indicates that landscape A is not consistent with landscape B .

Proposition 4. In the setting we just described, the consistency score in Equation 4 can be estimated as follows:

$$\mathcal{I}^\epsilon(\theta^*) \leq 2\epsilon \left(\frac{\det(H_{A+B})}{\det(H_{A \wedge B})} \right)^2.$$

Before showing the proof, we note that the proposition gives a *lower bound* on the consistency. That is, it provides a *pessimistic* estimate. Yet, as we motivated, this estimate has a nice geometric interpretation. However, as we outline in a remark after the proof, this estimate is tight in two important limit cases.

Proof. In this setting, Equation 4 gives

$$\mathcal{I}^\epsilon(\theta^*) := \max \left\{ \max_{\mathcal{L}_A(\theta) \leq \epsilon} \mathcal{L}_B(\theta), \max_{\mathcal{L}_B(\theta) \leq \epsilon} \mathcal{L}_A(\theta) \right\}.$$

Recall that

$$\mathcal{L}_A(\theta) = \frac{1}{2}\theta^\top H_A \theta = \frac{1}{2} \sum_i \lambda_i^A \theta_i^2.$$

Hence, this is a simple quadratic program with quadratic constraints, and

$$\max_{\mathcal{L}_A(\theta) \leq \epsilon} \mathcal{L}_B(\theta) = \max_{\frac{1}{2} \sum_i \lambda_i^A \theta_i^2 \leq \epsilon} \frac{1}{2} \sum_i \lambda_i^B \theta_i^2.$$

Further, we can change variables and introduce $\tilde{\theta}_i = \theta_i \sqrt{\lambda_i^A/2}$. The problem gets even simpler:

$$\max_{\mathcal{L}_A(\theta) \leq \epsilon} \mathcal{L}_B(\theta) = \max_{\|\tilde{\theta}\|^2 \leq \epsilon} \sum_i \frac{\lambda_i^B}{\lambda_i^A} \tilde{\theta}_i^2 = \epsilon \cdot \max_i \frac{\lambda_i^B}{\lambda_i^A}.$$

All in all, we get

$$\begin{aligned} \mathcal{I}^\epsilon(\theta^*) &= \epsilon \max \left\{ \max_i \frac{\lambda_i^B}{\lambda_i^A}, \max_i \frac{\lambda_i^A}{\lambda_i^B} \right\} \\ &= \epsilon \cdot \max_i \max \left\{ \frac{\lambda_i^B}{\lambda_i^A}, \frac{\lambda_i^A}{\lambda_i^B} \right\} \\ &\leq \epsilon \cdot \max_i \left(\frac{\lambda_i^B}{\lambda_i^A} + \frac{\lambda_i^A}{\lambda_i^B} \right) \\ &= \epsilon \cdot \max_i \left\{ \frac{(\lambda_i^B)^2 + (\lambda_i^A)^2}{\lambda_i^B \lambda_i^A} \right\} \\ &\leq \epsilon \cdot \max_i \left\{ \frac{(\lambda_i^B + \lambda_i^A)^2}{\lambda_i^B \lambda_i^A} \right\}. \end{aligned}$$

This means

$$\begin{aligned} \sqrt{\mathcal{I}^\epsilon(\theta^*)} &\leq \epsilon \max_i \frac{\lambda_i^B + \lambda_i^A}{\sqrt{\lambda_i^B \lambda_i^A}} = 2\epsilon \max_i \frac{(\lambda_i^B + \lambda_i^A)/2}{\sqrt{\lambda_i^B \lambda_i^A}} \leq \\ &\leq 2\epsilon \frac{\prod_i (\lambda_i^B + \lambda_i^A)/2}{\prod_i \sqrt{\lambda_i^B \lambda_i^A}} = 2\epsilon \frac{\det(H_{A+B})}{\det(H_{A \wedge B})}, \end{aligned}$$

where the first inequality comes from the monotonicity of the square root function, and the second inequality comes from the fact that (i) the geometric mean is always smaller or equal than the arithmetic mean and (ii) for any sequence of numbers $\alpha_i > 1$, $\max_i \alpha_i \leq \prod_i \alpha_i$. \square

Remark 2 (Sanity check). There are two important cases where we can test the bound above. First, if $H_A = H_B$, then $\mathcal{I}^\epsilon(\theta^*) = \epsilon$, and the bound returns $\mathcal{I}^\epsilon(\theta^*) \leq 2\epsilon$, since the geometric and arithmetic mean are the same. Next, say $\lambda_i^A = 0$ but $\lambda_i^B > 0$; then, both the bound and the inconsistency score are ∞ (highest possible inconsistency).

B.1.3 Proof of Proposition 1

In this appendix section we consider the AND-masked GD algorithm, introduced at the end of Section 3.2. We recall that the masked gradients at iteration k are $m_t(\theta^k) \odot \nabla \mathcal{L}(\theta^k)$, where $m_t(\theta^k)$ vanishes for any component where there are less than $t \in \{d/2 + 1, \dots, d\}$ agreeing gradient signs across environments, and is equal to one otherwise. In a full-batch setting, the algorithm is

$$\theta^{k+1} = \theta^k - \eta m_t(\theta^k) \odot \nabla \mathcal{L}(\theta^k), \quad (\text{AND-masked GD})$$

where $\eta > 0$ is the learning rate.

Proposition 1. Let \mathcal{L} have L -Lipschitz gradients and consider a learning rate $\eta \leq 1/L$. After k iterations, AND-masked GD visits at least once a point θ where $\|m_t(\theta) \odot \nabla \mathcal{L}(\theta)\|^2 \leq \mathcal{O}(1/k)$.

Proof. Thanks to the component-wise L -smoothness and using a Taylor expansion around θ^i we have

$$\begin{aligned} \mathcal{L}(\theta^{i+1}) &\leq \mathcal{L}(\theta^i) - \eta \langle \nabla \mathcal{L}(\theta^i), m_t(\theta^i) \odot \nabla \mathcal{L}(\theta^i) \rangle + \frac{L\eta^2}{2} \|m_t(\theta^i) \odot \nabla \mathcal{L}(\theta^i)\|^2 \\ &= \mathcal{L}(\theta^i) - \left(\eta - \frac{L\eta^2}{2} \right) \|m_t(\theta^i) \odot \nabla \mathcal{L}(\theta^i)\|^2. \end{aligned}$$

If we seek $\eta - L\eta^2/2 \geq \eta/2$, then $\eta \leq \frac{1}{L}$, as we assumed in the proposition statement. Therefore, $\mathcal{L}(\theta^{i+1}) \leq \mathcal{L}(\theta^i) - (\eta/2) \|m_t(\theta^i) \odot \nabla \mathcal{L}(\theta^i)\|^2$, for all $i \geq 0$. Summing over i from 0 to a desired iteration k , we get

$$\sum_{i=0}^{k-1} (\eta/2) \|m_t(\theta^i) \odot \nabla \mathcal{L}(\theta^i)\|^2 \leq \mathcal{L}(\theta^0) - \mathcal{L}(\theta^k) \leq \mathcal{L}(\theta^0).$$

Therefore,

$$\min_{i=0, \dots, k} \|m_t(\theta^i) \odot \nabla \mathcal{L}(\theta^i)\|^2 \leq \frac{1}{k} \sum_{i=0}^{k-1} (\eta/2) \|m_t(\theta^i) \odot \nabla \mathcal{L}(\theta^i)\|^2 \leq \frac{2\mathcal{L}(\theta^0)}{\eta k}.$$

Hence, there exist an iteration $i^* \in \{0, \dots, k\}$ such that $\|m_t(\theta^{i^*}) \odot \nabla \mathcal{L}(\theta^{i^*})\|^2 \leq \mathcal{O}(1/k)$. \square

B.1.4 Proof of Proposition 2

Here we fix parameters $\theta \in \mathbb{R}^n$ and assume gradients $\nabla \mathcal{L}_e(\theta) \in \mathbb{R}^n$ coming from environments $e \in \mathcal{E}$ are drawn independently from a multivariate Gaussian with zero mean and $\sigma^2 I$ covariance.

We want to show that, in this random setting, the AND-mask introduced in Section 3.2.3 decreases the magnitude of the gradient step.

Proposition 2. Consider the setting we just outlined, with $\mathcal{L} = (1/d) \sum_{e=1}^d \mathcal{L}_e$. While $\mathbb{E} \|\nabla \mathcal{L}(\theta)\|^2 = \mathcal{O}(n/d)$, we have that $\forall t \in \{d/2 + 1, \dots, d\}, \exists c \in (1, 2]$ such that $\mathbb{E} \|m_t(\theta) \odot \nabla \mathcal{L}(\theta)\|^2 \leq \mathcal{O}(n/c^d)$.

Proof. Let us drop the argument θ for ease of notation. First, let us consider $\nabla \mathcal{L}$ (no gradient AND-mask):

$$\mathbb{E} \left\| \frac{1}{d} \sum_{i=1}^d \nabla \mathcal{L}_{e_i} \right\|^2 = \frac{1}{d^2} \sum_{i=1}^d \mathbb{E} \|\nabla \mathcal{L}_{e_i}\|^2 = \frac{n\sigma^2}{d},$$

where in the first equality we used the fact that the $\nabla \mathcal{L}_{e_i}$ are uncorrelated and in the second the fact that $\mathbb{E} [\|\nabla \mathcal{L}_{e_i}\|^2]$ is the trace of the covariance of $\nabla \mathcal{L}_{e_i}$.

Next, assume we apply the element-wise AND-mask m_t to the gradients, which puts to zero the components (dimensions) where there are less than $t \in \{d/2, \dots, d\}$ equal signs. Since Gaussians are symmetric around zero, the probability of having *exactly* u positive j -th gradient component among d environments is $\Pr(p_j = u) = \left(\frac{1}{2}\right)^d \binom{d}{u}$. Hence, the probability to keep the j -th gradient direction (considering also negative consistency) is

$$\begin{aligned} \Pr[[m_t]_j = 1] &= \sum_{u=t}^d \Pr(p_j = u) + \sum_{u=0}^{d-t} \Pr(p_j = u) \\ &= \left(\frac{1}{2}\right)^d \sum_{k=t}^d \binom{d}{k} + \left(\frac{1}{2}\right)^d \sum_{k=0}^{d-t} \binom{d}{k} \\ &= 2 \left(\frac{1}{2}\right)^d \sum_{k=t}^d \binom{d}{k}. \end{aligned} \tag{12}$$

We would now like to compute $\mathbb{E} \left\| m_t \odot \left(\frac{1}{d} \sum_{i=1}^d \nabla \mathcal{L}_{e_i} \right) \right\|^2$. The difficulty lies in the fact that the event $m_t = 1$ makes gradients conditionally *dependent*. Indeed, conditioning on both $m_t = 1$ and $[\nabla \mathcal{L}_e]_j > 0$ changes the distribution of $[\nabla \mathcal{L}_{e'}]_j$: this gradient entry is going to be more likely to be positive or negative, depending on the value of $[\nabla \mathcal{L}_e]_j$ and on the details of the gradient mask. To solve the issue, we our strategy is to reduce the discussion (without loss in generality and with no additional assumption) to the case where gradient entries have all the same sign and hence conditional independence is restored.

We consider the following writing for the quantity we are interested in:

$$\begin{aligned} \mathbb{E} \left\| m_t \odot \left(\frac{1}{d} \sum_{i=1}^d \nabla \mathcal{L}_{e_i} \right) \right\|^2 &= \sum_{j=1}^n \mathbb{E} \left[[m_t]_j \left(\frac{1}{d} \sum_{i=1}^d [\nabla \mathcal{L}_{e_i}]_j \right)^2 \right] \\ &= \sum_{j=1}^n \sum_{\hat{p}_j=0}^d \mathbb{E} \left[[m_t]_j \left(\frac{1}{d} \sum_{i=1}^d [\nabla \mathcal{L}_{e_i}]_j \right)^2 \middle| p_j = \hat{p}_j \right] \Pr[p_j = \hat{p}_j] \\ &= \sum_{j=1}^n \sum_{\hat{p}_j=0}^{(d-t)} \sum_{\hat{p}_j=t}^d \mathbb{E} \left[\left(\frac{1}{d} \sum_{i=1}^d [\nabla \mathcal{L}_{e_i}]_j \right)^2 \middle| p_j = \hat{p}_j \right] \Pr[p_j = \hat{p}_j] \\ &= 2 \sum_{j=1}^n \sum_{\hat{p}_j=t}^d \mathbb{E} \left[\left(\frac{1}{d} \sum_{i=1}^d [\nabla \mathcal{L}_{e_i}]_j \right)^2 \middle| p_j = \hat{p}_j \right] \left(\frac{1}{2} \right)^d \binom{d}{\hat{p}_j}, \end{aligned}$$

where we used the definition of 2-norm, the law of total expectation, and the symmetry of the problem with respect to positive and negative numbers. Finally, since the gradient components within the same environment are conditionally independent, for any $j \in \{1, \dots, n\}$ we can write

$$\mathbb{E} \left\| m_t \odot \left(\frac{1}{d} \sum_{i=1}^d \nabla \mathcal{L}_{e_i} \right) \right\|^2 = 2n \sum_{\hat{p}_j=t}^d \mathbb{E} \left[\left(\frac{1}{d} \sum_{i=1}^d [\nabla \mathcal{L}_{e_i}]_j \right)^2 \middle| p_j = \hat{p}_j \right] \left(\frac{1}{2} \right)^d \binom{d}{\hat{p}_j}.$$

Finally, we note that the following bound holds:

$$\mathbb{E} \left[\left(\frac{1}{\hat{d}} \sum_{i=1}^d [\nabla \mathcal{L}_{e_i}]_j \right)^2 \middle| p_j = \hat{p}_j \leq d \right] \leq \mathbb{E} \left[\left(\frac{1}{\hat{d}} \sum_{i=1}^d [\nabla \mathcal{L}_{e_i}]_j \right)^2 \middle| p_j = d \right].$$

Indeed, if *all* environments lead to positive (or, symmetrically, negative) and *non-interacting* gradients in the j -th direction, the average will be the biggest in norm. Moreover — crucially — conditioned on the event $p_j = d$, gradients coming from different environments are distributed as a positive half-normal distributions. Moreover, they are *conditionally independent*; this because, since they are all positive, the value of a gradient in one environment cannot influence the value of the gradient in another one. We remark that conditional independence on the right-hand side is therefore *not an assumption*, but is intrinsic to the upper bound.

Putting it all together, we have

$$\begin{aligned} \mathbb{E} \left\| m_t \odot \left(\frac{1}{\hat{d}} \sum_{i=1}^d \nabla \mathcal{L}_{e_i} \right) \right\|^2 &\leq 2n \sum_{\hat{p}_j=t}^d \mathbb{E} \left[\left(\frac{1}{\hat{d}} \sum_{i=1}^d [\nabla \mathcal{L}_{e_i}]_j \right)^2 \middle| p_j = d \right] \left(\frac{1}{2} \right)^d \binom{d}{\hat{p}_j} \\ &\leq 2n \sum_{\hat{p}_j=t}^d \sigma^2 \left(\frac{1}{2} \right)^d \binom{d}{\hat{p}_j} \\ &\leq \sigma^2 n (d-t) \binom{d}{t} \left(\frac{1}{2} \right)^{d-1}, \end{aligned}$$

where in the second line we bounded the squared average of a sum of half normal distributions: let $\{X_i\}_{i=1}^d$ be a family of uncorrelated positive half-normal distributions derived from a Gaussians with mean zero and variance σ^2 , we have² that $\mathbb{E}[X_i] =$

² https://en.wikipedia.org/wiki/Half-normal_distribution

$\sigma\sqrt{2/\pi}$ and $\mathbb{E}[X_i^2] = \sigma^2$. Also, $\mathbb{E}[X_i X_j] = \mathbb{E}[X_i]\mathbb{E}[X_j] \leq \sigma^2$. Therefore,

$$\mathbb{E} \left[\left(\frac{1}{d} \sum_{i=1}^d X_i \right)^2 \right] = \frac{1}{d^2} \sum_{i,j=1}^d \mathbb{E}[X_i X_j] \leq \sigma^2.$$

Finally, if we set $r = t/d \in (0.5, 1]$, we have³

$$\binom{d}{t} \sim \left(\frac{1}{r^r(1-r)^{1-r}} \right)^d$$

as $d \rightarrow \infty$ (discarding all polynomial terms). Hence $\binom{d}{t}$ is of the form q^d , with $1 \leq q < 2$. So, the quantity $\sigma^2 n(d-t) \binom{d}{t} \left(\frac{1}{2}\right)^{d-1}$ will be exponentially decreasing at a rate $\mathcal{O}(n/(2-q)^d)$. Notably, if $t = d/2$, then we lose the exponential rate and get back to $\mathcal{O}(n/d)$. \square

B.2 APPENDIX TO SECTION 3.3

We used Pytorch [Paszke et al., 2017](#) to implement all experiments in this paper. Our codebase is publicly available at github.com/gibipara92/learning-explanations-hard-to-vary.

B.2.1 Section 3.3.1

B.2.2 Dataset

Here we report more technical details about the synthetic dataset described in Section 3.3. Each example is constructed as follows: we first choose the label randomly to be either $+1$ or -1 , with

³ Theorem 1 in Burić, Tomislav, and Neven Elezović. "Asymptotic expansions of the binomial coefficients." *Journal of applied mathematics and computing* 46.1-2 (2014): 135-145.

Table 3: Hyperparameter ranges for synthetic data experiments. The regularizers L1 and L2 are never combined; instead, one weight regularization type out of L1, L2 and none is selected and we sample from the respective range afterwards.

Hyperparameter	Ranges
No. hidden units	{256, 512}
No. hidden layers	{3, 5}
Batch-size	{64, 128, 256}
Optimizer	{Adam $_{\beta_1=0.9, \beta_2=0.999}$, SGD + mom $_{0.9}$ }
Learning rate	{1e-3, 1e-2, 1e-1}
Batch-normalization	{Yes, No}
Dropout	{0.0, 0.5}
L2 regularization	{1e-5, 1e-4, 1e-3}
L1 regularization	{1e-6, 1e-5, 1e-4}

equal probability. The example is a vector with $d_S + d_M$ entries, consisting of the *shortcut* and the *mechanism*. In our experiments, $d_M = 2$ and $d_S = 32$.

The Gaussian *shortcuts* are obtained by first sampling one random vector $\mathbf{x}_s \in \mathbb{R}^{d_S}$ per environment. Its components $x_{s,i}$ are sampled independently from a Normal distribution: $x_{s,i} \sim \mathcal{N}(0, 0.1)$. We use \mathbf{x}_s for class 1, and $-\mathbf{x}_s$ for class -1. In the test set, all shortcut components are sampled i.i.d. from the same Normal distribution. Effectively, each example of the test set belongs to a different domain. The *mechanism* is implemented as the two interconnected spirals shown in Figure 37 by sampling the radius $r \sim \text{Unif}(0.08, 1.0)$ and then computing the angle as $\alpha = 2\pi nr$ where n is the number of revolutions of the spiral. We add uniform noise in the range $[-0.02, 0.02]$ to the radii afterwards.

The training dataset consists of 1280 examples per environment and we use $D = 32$ environments unless otherwise mentioned. The training datasets consists of 2000 examples.

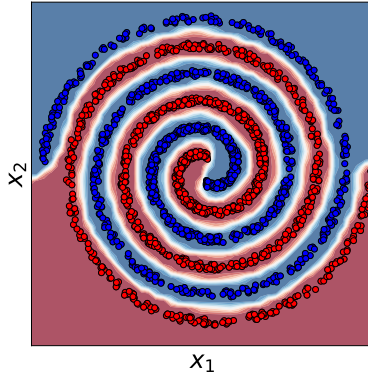


Figure 37: The spirals used as the *mechanism* in the synthetic memorization dataset.

B.2.3 Experiment

We train all networks for $\lfloor 3000/D \rfloor$ epochs, dropping the learning rate by a factor 10 halfway through, and again at three-quarters of training. For computational reason, we stop each trial before completion if the training accuracy exceeds 97% and the test accuracy is below 60%. All networks are MLPs with LeakyReLU activation functions and a cross-entropy loss on the output. We run a hyperparameter search over the ranges shown in Table 3. For IRM and the AND-mask, we select the best-performing run and re-run it 50 times with different random seeds. For DANN and the standard baselines nothing produced results significantly better than chance.

B.2.3.1 Standard regularizers and AND-mask

The networks with the L1, L2, Dropout and Batch-normalization regularizers, have hyperparameters that were randomly selected

from Table 3. For the AND-mask we used the very same ranges. The regularizers L1 and L2 are never combined; instead, one weight regularization type out of L1, L2 and none is selected and we sample from the respective range afterwards. The parameters found to work best from the grid search were: agreement threshold of 1, 256 hidden units, 3 hidden layers, batch size 128, Adam with learning rate $1e-2$, no batch norm, no dropout, L2-regularization with a coefficient of $1e-4$, no L1-regularization. In practice, we often found it helpful to rescale the gradients after masking to compensate for the decreasing overall magnitude. We add the option for gradient rescaling as an additional hyperparameter, as we found it to help in several experiments. It rescales gradient components layer-wise after masking, by multiplying the remaining gradient components by c , where c is the ratio of the number of components in that layer over the number of non-masked components in that layer (i.e. the sum of the binary elements in the mask).⁴ We speculate that for very large layers, a less extreme normalization scheme or the additional use of gradient clipping might be appropriate.

B.2.3.2 Domain Adversarial Neural Networks

The experiments using DANN follow a similar pattern. The model consists of an embedding network, a classification network, and a “domain discrimination” network. All three modules are two-layer multi-layer perceptrons (MLP). The number of hidden units of all MLPs are sampled from the range specified in Table 3, and we trained 100 models. Both label classifier and domain discriminator are applied to the output of the embedding network. The label classifier is trained to minimize the cross-entropy-loss between the predicted and the true label. Similarly, the domain discriminator is trained to minimize the loss between predicted and

⁴ Therefore, c is 1 if the AND-mask has only 1s, and infinite if all components are masked out (which we then keep as 0.)

Table 4: Hyperparameter ranges for IRM.

Hyperparameter	Ranges
No. hidden units	{256, 512}
No. hidden layers	{3, 5}
Batch-size	{64, 128, 256}
Optimizer	{Adam, SGD + momentum _{0,9} }
Batch-normalization	{Yes, No}
Penalty weight	{10.0, 100.0, 1000.0}
Number of annealing iterations	{0, 1, 2, 4, 8}
Learning rate	{1e-3, 1e-2, 1e-1, 1}

true domain-label. The embedding network is trained to minimize the regular task classification loss and at the same time to maximize the the domain-loss achieved by the domain discriminator.

B.2.3.3 *Invariant Risk Minimization*

For the experiments using IRM we used the authors’ PyTorch implementation from <https://github.com/facebookresearch/InvariantRiskMinimization>. We perform a random hyperparameter search over with the ranges shown in Table 4

B.2.3.4 *Curves for all experiments*

In Figure 38 we show the learning curves of training and test accuracy for the different methods.

B.2.3.5 *Correlation plots*

For the correlation plots in Figure 17 we used a randomly initialized MLP with the following configuration: 3 hidden layers, 256

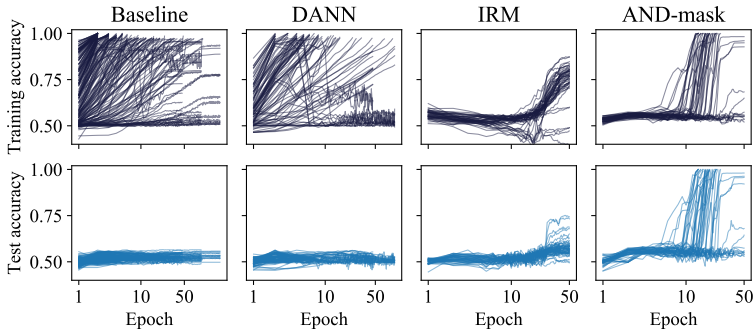


Figure 38: Learning curves for the evaluated methods. The top row shows the accuracy on the training set, the bottom row shows the accuracy on the test set.

hidden units. The dataset was using 16 environments and batches of size 1024. The lines in Figure 17 are linear least-squares regressions to the gradient data shown as scatter plots. We repeat the experiment 10 times with different network weight seeds, resulting in the 10 regression lines. Zero gradients are excluded from the regression computation, as most gradients are masked out by the product mask in both cases.

B.2.4 Further visualizations and experiments

In Figure 39 we show how many environments need to be present for the baseline without AND-mask to switch the decision boundary from the shortcuts to the mechanism. Under the same experimental condition as in the main paper, the baseline first succeeds at 1024 environments.

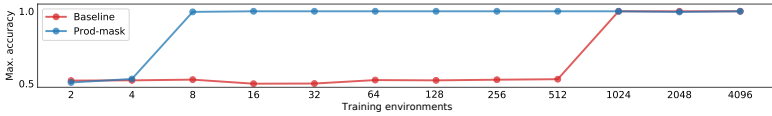


Figure 39: Relationship between number of training environments and test accuracy for the AND-mask method compared to the baseline. We show the best performance out of five runs using the settings that were used for the experiment in the main text.

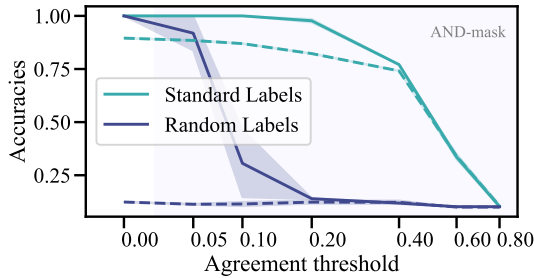


Figure 40: Dashed lines show test acc, solid lines show training acc.

B.2.5 Section 3.3.2: CIFAR-10 memorization and label noise experiments

MEMORIZATION EXPERIMENT In Figure 40, we report the test performance (dashed lines) corresponding to the curves presented in the main paper for the CIFAR-10 memorization experiment. The test performance with standard labels decreases slower than the training performance as the threshold increases, and they eventually reach the same value. This is consistent with the hypothesis that by training on the consistent directions, the AND-mask selects the invariant patterns and prunes out the signals that are not invariant.

NETWORK ARCHITECTURE AND TRAINING DETAILS Each trial trains the ResNet “FastResNet” from the PyTorch-Ignite example⁵ for 80 epochs on the full CIFAR-10 training set. We use the Adam optimizer with a learning rate of 5×10^{-4} , and a 0.1 learning rate decay at epoch 40 and 60. We fix the batch size to 80. We set up 14 trials by evaluating each of the AND-mask-thresholds $\{0, 0.05, 0.1, 0.2, 0.4, 0.6, 0.8\}$ for two datasets: (a) unchanged CIFAR-10, (b) CIFAR-10 with the training labels replaced by random labels. Note that a threshold of 0 corresponds to not using the AND-mask. Each trial is run twice with separate random seeds.

LABEL NOISE EXPERIMENT We trained the same ResNet as for the experiment above, once with and once without the AND-mask. We ran each experiment with three different starting learning rates $\{5 \times 10^{-4}, 1 \times 10^{-3}, 5 \times 10^{-3}\}$ and a learning rate decay at epoch 60. The baseline worked best with a learning rate of 1×10^{-3} , while the AND-mask with 5×10^{-3} , likely to compensate for the masked out gradients. The AND-mask threshold that worked best was 0.2, which is consistent with the results obtain in the experiment above.

B.2.6 Section 3.3.3: Behavioral Cloning on CoinRun

The target policy π^* is obtained by training PPO (Schulman et al., 2017) for 400M time steps using the code⁶ for the paper Cobbe et al., 2020. This policy is trained on the full distribution of levels in order to maximize its generality. We use π^* to generate a behavioral cloning (BC) dataset, consisting of pairs $(s, \pi^*(a|s))$, where s are the input-images (64×64 RGB) and $\pi^*(a|s)$ is the discrete probability distribution over actions output by π^* .

⁵ <https://github.com/pytorch/ignite/blob/master/examples/contrib/cifar10/fastresnet.py>

⁶ <https://github.com/openai/train-procgen>

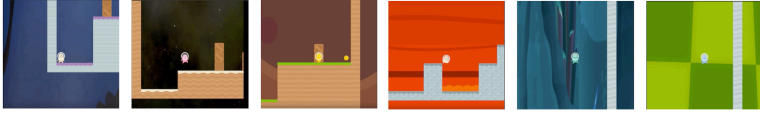


Figure 41: Screenshots of 6 levels of CoinRun (from OpenAI).

Algorithm 4 Temporal AND-mask Adam

```

1  $\mathbf{m} \leftarrow \beta_1 \cdot \mathbf{m} + (1 - \beta_1) \cdot \mathbf{g}$ 
2  $\mathbf{v} \leftarrow \beta_2 \cdot \mathbf{v} + (1 - \beta_2) \cdot (\mathbf{g} \circ \mathbf{g})$ 
3  $\mathbf{a} \leftarrow \beta_3 \cdot \mathbf{a} + (1 - \beta_3) \cdot \text{elementwise\_sign}(\mathbf{g})$ 
4  $\mathbf{b} \leftarrow \mathbb{1}[|\mathbf{a}| \geq \tau]$ 
5  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha(\mathbf{m} \circ \mathbf{b}) \oslash \sqrt{\mathbf{v} + \epsilon}$ 

```

The states are sampled randomly from trajectories generated by π^* . In order to test for generalization performance, the BC training dataset is restricted to 64 distinct levels. We generate 1000 examples per training level. The test set consists of 2000 examples, each from a different level which does not appear in the training set.

A ResNet-18 $\hat{\pi}_\theta$ is trained to minimize the loss $D_{\text{KL}}(\pi^* || \hat{\pi}_\theta)$. We ran two automatic hyperparameter optimization studies using Tree-structured Parzen Estimation (TPE) (Bergstra et al., 2013) of 1024 trials each, with and without the AND-mask. The learning rate was decayed by a factor of 10 half-way at at $3/4$ of the training epochs.

The “temporal” version of the AND-mask used for this experiment is reported in Algorithm 4.

In blue we highlight the additional lines compared to traditional Adam. The threshold τ and β_3 are hyperparameters that we included in the 1024 trials of the search using Tree-structured Parzen Estimators. For the top 10 runs, hyperparameter values that were selected via the TPE search for the AND-mask are the following.

Table 5: Hyperparameters for the 5 best runs using the AND-mask, from the TPE search.

Test KL div	lr	β_1	β_3	τ	weight decay
1.652e-2	0.0078	0.21	0.79	0.36	0.057
1.656e-2	0.0072	0.26	0.86	0.40	0.041
1.662e-2	0.0080	0.23	0.84	0.41	0.045
1.665e-2	0.0068	0.33	0.72	0.47	0.077
1.672e-2	0.0063	0.67	0.65	0.47	0.080

We found that applying weight decay as a second independent update *after* the AND-mask routine improved performance. To keep the comparison fair, we added this as a switch in the hyperparameter search for the Adam baseline as well, and it improved performance there as well.

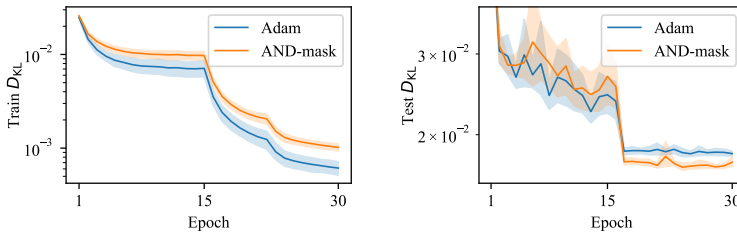


Figure 42: Learning curves for the behavioral cloning experiment on CoinRun. Training loss is shown on the left, test loss is shown on the right. We show the mean over the top-10 runs for each method. The shaded regions correspond to the 95% confidence interval of the mean based on bootstrapping.

B.3 APPENDIX TO SECTION 3.4

B.3.1 *Related work in causal inference*

CAUSAL GRAPHS AND CAUSAL FACTORIZATIONS The formalization of causality through directed acyclic graphs (Pearl, 2009) is a key element informing our exposition. According to such formalization, a causal model gives rise to each observed distribution. It is thereby possible to exploit properties of the causal factorization of the joint probability distribution over the observed variables. Clearly, there are many ways to factorize a joint distribution into conditionals; a distinguishing feature of the causal factorization is that many of the conditionals, which we can think of as physical mechanisms underlying the statistical dependencies represented, are expected to remain *invariant* under interventions or changing external conditions. This postulate has appeared in various forms in the literature (Haavelmo, 1943; Simon, 1953; Hurwicz, 1962; Pearl, 2009; Schölkopf et al., 2012).⁷

CAUSAL MODELS AND ROBUST REGRESSION Based on this insight, it was proposed that regression based on causal features should present desirable invariance and robustness properties (Mooij et al., 2009; Schölkopf et al., 2012; Peters et al., 2016; Rojas-Carulla et al., 2018; Heinze-Deml et al., 2018; Kügelgen et al., 2019; Parascandolo et al., 2018). In this view, the mechanisms can be considered as features of the patterns such that they support stable conditional probabilities. Thus learning the mechanisms may help achieve a stable performance across a number of conditions. Other works connecting causality and learning through invariances are (Subbaswamy et al., 2019; Heinze-Deml and Meinhäusen, 2017), and perhaps – most related to our work – (Ar-

⁷ This would be different for a non-causal factorization of the joint distribution, see Schölkopf, 2019

Arjovsky et al., 2019): we presented a comparison with this method in the following section.

CAUSAL REGULARIZATION Recently (Janzing, 2019) showed that biasing learning towards models of lower complexity might in some cases be beneficial for a notion of generalization from observational to interventional regimes. Our proposed solution is however different, in that we only indirectly deal with penalizing model complexity, and rather focus on our proposed notion of consistency.

B.3.2 Learning invariances in the data

Here we are going to compare ILC to other approaches for learning invariances in the data with neural networks, and in particular to Invariant Risk Minimization (IRM) Arjovsky et al., 2019. The authors of IRM analyze a set up where minimizing training error might lead to models which absorb all the correlations found within the training data, thus failing to recover the relevant causal explanation. They consider a multi-environment setting and focus on the objective of extracting data representations that lead to invariant prediction across environments.

While the high level objective is close to the one we focused on, the differences become clear when considering the definition of *invariant predictors* presented in Arjovsky et al., 2019:

Definition 2. A data representation $\Phi : \mathcal{X} \rightarrow \mathcal{H}$ elicits an invariant predictor $w \circ \Phi$ across environments \mathcal{E} if there is a classifier $w : \mathcal{H} \rightarrow \mathcal{Y}$ simultaneously optimal for all environments, i.e., $w \in \arg \min_{\bar{w}: \mathcal{H} \rightarrow \mathcal{Y}} R^e(\bar{w} \circ \Phi) \forall e \in \mathcal{E}$.

In particular, the objective minimized by IRM is:

$$\min_{\Phi: \mathcal{X} \rightarrow \mathcal{Y}} \sum_{e \in \mathcal{E}_{\text{tr}}} R^e(\Phi) + \lambda \cdot \left\| \nabla_{w|w=1.0} R^e(w \cdot \Phi) \right\|^2 \quad (13)$$

where Φ are the logits predicted by the neural network and w is a dummy scaling variable (see Arjovsky et al., 2019). The relevant part is the penalty term $\lambda \cdot \left\| \nabla_{w|w=1.0} R^e(w \cdot \Phi) \right\|^2$: One way to interpret it, is that the penalty is large on every environment where the distribution outputted by Φ could be made ‘closer’ to the distribution of the labels by either sharpening ($w > 1$) or softening it (i.e., closer to uniform $w < 1$).

Let us consider the example from IRM, where the authors describe two datasets of images that each contain either a cow or a camel: In one of the datasets, there is grass on 80% of the images with cows, while in the other dataset there is grass on 90% of them. IRM then makes the point that we can learn to ignore grass as a feature, because its correlation with the label cow is inconsistent (80% vs 90%). The setting we consider in this paper is slightly different: take our example from the CIFAR-10 experiments. Under our concept of invariance, we expect that (depending on the data generating process) even a single dataset where we treat every image as coming from its own ‘environment’ should be sufficient to discover invariances. Drawing a connection to the setting from IRM, we would argue that the second dataset should not be necessary to learn that ‘grass’ is not ‘cow’. If one treats every example as coming from its own environment, there is already sufficient information in the first dataset to realize that cows are not grass: Grass is predictive of cows only in 80% of the data, so grass cannot be ‘cow’. The actual cow on the other hand, should be present in 100% of the images, and as such it is the invariance we are looking for. Note that this is of course a much more strict definition of invariance: If our dataset contains images labeled as ‘cows’ but that have no cows within them, we might start to discard the features of cows as well.

APPENDIX EQL

C.1 MODEL DETAILS

C.1.1 *NeSymReS Transformer Details*

The model consists of an encoder and a decoder. The encoder takes as input numerical data, $X \in \mathbb{R}^{(d_x+d_y) \times n}$, where d_x is the number of independent variables, d_y the number of dependent variables and n the number of support points. In order to prevent exploding gradient and numerical instabilities, we convert each entry of X into a multi-hot bit representation according to the half precision IEEE-754 standard. This operation yields a new input tensor, $\tilde{X} \in \mathbb{R}^{(d_x+d_y) \times b \times n}$ where $b = 16$ is the dimension of the bit representation. The output of the encoder is a latent vector $z \in \mathbb{R}^{d_z}$, providing a compressed representation of the equation to be modelled. Such latent vector is then used to condition the decoder via a standard Transformer multi-head attention mechanism. During the pre-training phase, the input of the decoder is given by the sequence of tokens representing the ground truth-equation expressed in prefix notation. Such sequence is opportunely masked in order to prevent information leakage in the decoder forward step. The output is then given by a string of symbols, representing the predicted equation, again in prefix notation. During inference, the decoder is only provided with the information from the latent vector z and generates a prediction autoregressively. We use *beam search* to obtain candidate solutions. After removing potentially invalid equations, the remaining equations are modified to include constant placeholders with the procedure

described in Appendix C.2. Using BFGS, we fit these constants. We select the best equation among the so-found candidates, based on the validation loss, with an added regularization penalty of 10^{-14} for each token in the skeleton. Note that BFGS is currently the most time-consuming step of our pipeline. While in all our experiments we run the optimization procedure serially (i.e., one candidate equation at the time), the procedure can be easily parallelized across equations.

Encoder and decoder use the same hidden dimension H and number of heads, h for their multi-head attention modules. In the following, we provide further details about the architectural design choices, hyper-parameters and library of functions used by our model. We trained our model on a single GeForce RTX 2080 GPU for 3 days.

ENCODER For the encoder, we opted for the Set Transformer architecture from Lee et al. (2019). Our choice is motivated in light of the better scaling properties of this method when it comes to input length n , i.e. $\mathcal{O}(nm)$ compared to the standard transformer encoder, $\mathcal{O}(n^2)$, where m is a set of trainable inducing points. Referring to the notation of the original paper, our encoder is formed by n_e Induced Set Attention Blocks (ISABs) and one final component performing Pooling by Multihead Attention (PMA). ISABs differ from the multi-head self attention blocks present in the original Transformer architecture, since they introduce $m < n$ learnable inducing points that reduce the computation burden associated with the self-attention operation. PMA allows us to aggregate the output of the encoder into d_z trainable abstract features representing a compressed representation of the input equation. Overall, the encoder consists of 11M trainable parameters. All the hyper-parameters of the encoder are listed in Table 6.

DECODER The decoder is a standard Transformer decoder. It has n_d layers, hidden dimension H , h attention heads. Input symbols are encoded into the corresponding token embeddings and information about relative and absolute positions of the tokens in the sequence is injected by adding learnable positional encodings to the input embeddings. Two different masks are used to avoid information leakage during the forward step and to make the padding token hidden to the attention modules. Overall, the decoder consists of $15M$ trainable parameters. All the hyper-parameters of the decoder are listed in Table 7.

Table 6: Encoder hyper-params.

Parameter name	Symbol	Value
Num. of ISABs	n_e	5
Hidden dimension	H	512
Num. of heads	h	8
Num. of PMA features	d_z	10
Num. of ind. points	m	50

Table 7: Decoder hyper-params.

Parameter name	Symbol	Value
Num. of layers	n_d	5
Hidden dimension	H	512
Num. of heads	h	8
Embedding dim.	s	32

C.1.2 Baselines

DEEP SYMBOLIC REGRESSION (DSR) For DSR, we use the standard hyper-parameters provided in the open-source implementation of the method, with the setting that includes the estimation of numerical constants in the final predicted equation. DSR depends on two main hyper-parameters, namely the entropy coefficient λ_H and the risk factor ϵ . The first is used to weight a bonus proportional to the entropy of the sampled expression which is added to the main objective. The second intervenes in the definition of the final objective with depends on the $(1 - \epsilon)$ quantile of the distribution of rewards under the current policy. According with the open-source implementation and the results reported in [Petersen](#),

2021, we choose $\epsilon = 0.05$ and $\lambda_H = 0.005$. The set of symbols available to the algorithm to form mathematical expressions is given by $\mathcal{L} = \{+, -, \times, \div, \sin, \cos, \exp, \ln, c\}$, where c stands for the constant placeholder.

GENETIC PROGRAMMING For Genetic Programming, we opt for the open-source Python library `gplearn`. Our choices for the hyper-parameters are listed in Table 8 and mostly reflect the default values indicated in the library documentation. The set of symbols available to the algorithm to form mathematical expressions is the default one and is given by $\mathcal{L} = \{+, -, \times, \div, \sqrt{\cdot}, \ln, \exp, \text{neg}, \text{inv}, \sin, \cos\}$, where *neg* and *inv* stand for «negation» ($x \mapsto -x$), and inversion ($x \mapsto x^{-1}$), respectively.

Table 8: Genetic Programming hyper-parameters. The parameter *Population size* is varied within the range indicated during the experiments reported in Section 4.5.

Parameter name	Value
Population size	$\{2^{10}, \dots, 2^{15}\}$
Selection type	Tournament
Tournament size (k)	20
Mutation probability	0.01
Crossover probability	0.9
Constants range	$(-4\pi, 4\pi)$

GAUSSIAN PROCESSES This is the only baseline that is *not* a symbolic regression method per se, as it learns a mapping from x to y directly. The appealing property of Gaussian Processes is that they are very accurate in distribution, and are very fast to fit in the regime we considered. We opted for the open-source `sklearn` implementation of Gaussian Process regression with

default hyper-parameters. The covariance is given by the product of a constant kernel and an RBF kernel. Diagonal Gaussian noise of variance 10^{-10} is added to ensure positive-definiteness of the covariance matrix. L-BGFS-B is used for the optimization of the marginal likelihood with the number of restarts varied as indicated in Table 1.

A NOTE ABOUT FUNCTION SETS Unfortunately, not all methods support all primitive functions that appear in a given dataset. For example, NeSymReS *could* support x^6 , and x^y — that appear in the Nguyen dataset described in Appendix C.3 — but as we did not include these primitives in the pre-training phase, the version we use in our experiments will not be able to correctly recover these equations. DSR and the implementation of Genetic Programming that we adopted are both lacking arcsin in their function set.

While missing primitives lowers the upper bound in performance that a method can reach for a given dataset, it also makes it easier to fit the other equations that *do not* contain those primitives, as the function set to search is effectively smaller.

C.2 EXPERIMENTAL DETAILS

C.2.1 Training

Operators	+	×	-	÷	√	Pow	ln	exp	sin	cos	tan	arcsin
Freq.	10	10	5	5	4	4	4	4	4	4	4	1

Table 9: Operators and their corresponding un-normalized probabilities of being sampled as parent node.

TRAINING DATASET GENERATION For generating skeletons, we built on top of the method and code proposed in [Lample and Charton, 2020](#), which samples expression trees. For our experiments, each randomly-generated expression tree has 5 or fewer non-leaf nodes. We sample each non-leaf node following the unnormalized weighted distribution shown in Table 9. Each leaf node has a probability of 0.8 of being an independent variable and 0.2 of being an integer. Trees that contain the independent variable x_2 must also have the independent variable x_1 . Those containing the independent variable x_3 must also include the independent variables x_1 and x_2 . We then traverse the tree in pre-order and obtain a semantically equivalent string of the expression tree in a prefix notation. We convert the string from prefix to infix notation and simplify the mathematical expression using the Sympy library. The resulting expression is then modified to include constant placeholders as explained in the following paragraph. This expression is what we refer to as a *skeleton*, as the value of constants has not been determined yet.

For our experiments, we repeat the procedure described above to obtain a pre-compiled dataset of 10M equations. To compile the symbolic equation into a function that can be evaluated by the computer on a given set of input points, we relied on the function *lambdify* from the library Sympy. We store the equations as functions, in order to allow for the support points and values of the constants to be resampled at mini-batch time during pre-training. We opted for a partially pre-generated dataset instead of sampling new equations for every batch in order to speed up the generation of training data for the mini-batches.

TRAINING DETAILS As described in Section, [4.4.2](#), during training we sampled mini-batches of size $B = 150$ from the generated dataset. For each equation, we first choose the number of constants, n_c , that differ from one. n_c is randomly sampled within

the interval ranging from 0 to $\min(3, N_c)$ where N_c is the maximum number of constants that can be placed in the expression. Then, we sample the constants' values from the uniform distribution $\mathcal{U}(1, 5)$. We randomly select n_c among the available placeholders and replace them with the previously obtained numerical values. The remaining constants are set to one. We then generate up to 500 support points by sampling- independently for each dimension - from uniform distributions with varying extrema as described in Section 5.5. If the equation does not contain a given independent variable, such variable is set to 0 for all the support points. For convenience, we drop input-output pairs containing NaNs and entries with an absolute value of y above 1000. Finally we take the equation in the mini-batch with the minimum number of points, and drop valid points from the other equations so that the batch tensor has a consistent length across equations.

TRAINING DATASET DISTRIBUTION The dataset does not consist of unique mathematical expressions. Indeed, some skeletons are repeated, and some skeletons are mathematically equivalent. Overall, within the 10M dataset, we have ~ 1.2 M unique skeletons. Since longer expressions tend to be simplified into shorter expressions during the dataset generation, shorter expressions are the most frequent ones. Of these 1.2M unique skeletons, at least 96.3% represents unique (numerically distinct) mathematical expressions. The counting procedure is described in the next paragraph. The average length of an expression in infix notation is 8.2 tokens. The minimum and the maximum are 1 and 23 respectively, which corresponds in infix notation to the expressions x and $\frac{x^2 \operatorname{asin}^2(x)}{-x_1^2 \operatorname{asin}^2(x)+1}$.

ADDITION OF NUMERICAL CONSTANTS During dataset generation and inference, we introduce constant placeholders by attaching them to the generated or predicted skeletons. This step

is carried out by multiplying all unary operators in the expressions by constant placeholders (except for the «pow» operator). The same procedure is repeated with independent variables for which also additive constants are introduced. Longer expressions tend to have more placeholders in comparison to shorter ones.

C.2.2 Evaluation details

All results reported, i.e. for all methods and datasets, are accuracies over all equations in the dataset. Error bars in all plots denote the standard error of the mean estimate.

METRICS DETAILS As detailed in 4.4.6 we evaluate performances both within the training support (A^{iid}) and outside of the training support (A^{ood}). More specifically, for the latter the support is created as follows: given an equation with in-sample support of (l_o, h_i) , we extend the support of each side by $(h_i - l_o)$ for every variable present in the equation.

CREATING THE SOOBE DATASET The SOOBE (stricly out-of-sample equations) dataset contains entirely different skeletons from the pre-training dataset, which do not overlap numerically nor symbolically. To create it, we list all the different expressions in the training dataset and then randomly sample from this set, excluding the sampled expressions from the training set. We first sample a random support of 500 points from the uniform distribution $\mathcal{U}(-10, 10)$, for each independent variable. Two expressions are different if their images, given the support points, are different. Note that this is a conservative criterion, as two expressions may have the same image in the sampled support (relatively to a fixed tolerance), yet being distinct.

BENCHMARKS As explained in 4.4.4 we evaluate our trained model on five datasets: AI-Feynman, SOOBE-WC, SOOBE-NC, SOOBE-FC, Nguyen. All the equations of AI-Feynman used in our evaluation are listed in table 10. 50 randomly sampled equations out of 200 from the SOOBE dataset, are listed in table 11.

Expression	Support x_1	Support x_2	Support x_3	Expression	Support x_1	Support x_2	Support x_3
$\frac{\sqrt{2} x_1^2}{2\sqrt{\pi}}$	(1, 3)	None	None	$\frac{x_1 x_2^2}{\sqrt{-\frac{x_2^2}{x_3^2}+1}}$	(1, 5)	(1, 2)	(3, 10)
$\frac{\sqrt{2} x_1^2}{2\sqrt{\pi x_1}}$	(1, 3)	(1, 3)	None	$\frac{x_1}{4\pi x_2^2}$	(1, 5)	(1, 5)	None
$\frac{\sqrt{2} x_1^2 (x_2 - x_3)^2}{2\sqrt{\pi x_1} 2x_1^2}$	(1, 3)	(1, 3)	(1, 3)	$\frac{x_1}{4\pi x_2 x_3}$	(1, 5)	(1, 5)	(1, 5)
$\frac{x_1}{\sqrt{-\frac{x_2^2}{x_3^2}+1}}$	(1, 5)	(1, 2)	(3, 10)	$\frac{3x_1^2}{20\pi x_2 x_3}$	(1, 5)	(1, 5)	(1, 5)
$x_1 x_2$	(1, 5)	(1, 5)	None	$\frac{x_1 x_2^2}{2}$	(1, 5)	(1, 5)	None
$\frac{x_1}{4\pi x_2 x_3^2}$	(1, 5)	(1, 5)	(1, 5)	$\frac{x_1}{x_2(x_3+1)}$	(1, 5)	(1, 5)	(1, 5)
$x_1 x_2$	(1, 5)	(1, 5)	None	$\frac{x_1 x_2}{x_1 x_2 + 1} + 1$	(0, 1)	(0, 1)	None
$x_1 x_2 x_3$	(1, 5)	(1, 5)	(1, 5)	$\frac{x_1}{\sqrt{-\frac{x_2^2}{x_3^2}+1}}$	(1, 5)	(1, 2)	(3, 10)
$\frac{x_1^2 x_3}{2}$	(1, 5)	(1, 5)	None	$\frac{x_1 x_2}{\sqrt{-\frac{x_2^2}{x_3^2}+1}}$	(1, 5)	(1, 2)	(3, 10)
$\frac{x_1 x_2}{\sqrt{-\frac{x_2^2}{x_3^2}+1}}$	(1, 5)	(1, 2)	(3, 10)	$-x_1 x_2 \cos(x_3)$	(1, 5)	(1, 5)	(1, 5)
$\frac{x_2 + x_3}{1 + \frac{x_2 x_3}{x_1}}$	(1, 5)	(1, 5)	(1, 5)	$-x_1 x_2 \cos(x_3)$	(1, 5)	(1, 5)	(1, 5)
$x_1 x_2 \sin(x_3)$	(1, 5)	(1, 5)	(0, 5)	$\sqrt{\frac{x_2^2}{x_3^2} - \frac{\pi^2}{x_3^2}}$	(4, 6)	(1, 2)	(2, 4)
$\frac{x_1}{x_2}$	(1, 5)	(1, 5)	None	$x_1 x_2 x_3^2$	(1, 5)	(1, 5)	(1, 5)
$\text{asin}(x_1 \sin(x_2))$	(0, 1)	(1, 5)	None	$x_1 x_2^2$	(1, 5)	(1, 5)	None
$\frac{1}{\frac{x_2}{x_1} + \frac{1}{x_1}}$	(1, 5)	(1, 5)	(1, 5)	$\frac{x_1 x_2}{2\pi x_3}$	(1, 5)	(1, 5)	(1, 5)
$\frac{x_1}{x_2}$	(1, 10)	(1, 10)	None	$\frac{x_1 x_2 x_3}{2}$	(1, 5)	(1, 5)	(1, 5)
$\frac{x_1 \sin^2\left(\frac{2x_2 x_3}{x_1}\right)}{\sin^2\left(\frac{x_2 x_3}{x_1}\right)}$	(1, 5)	(1, 5)	(1, 5)	$\frac{x_2 x_3}{4\pi x_3}$	(1, 5)	(1, 5)	(1, 5)
$\text{asin}\left(\frac{x_1}{x_2 x_3}\right)$	(1, 2)	(2, 5)	(1, 5)	$x_1 x_2 (x_3 + 1)$	(1, 5)	(1, 5)	(1, 5)
$\frac{x_2}{1 - \frac{x_2}{x_1}}$	(3, 10)	(1, 2)	(1, 5)	$\frac{x_1}{x_2 + 2}$	(1, 5)	(1, 5)	None
$\frac{x_3 \left(1 + \frac{x_2}{x_1}\right)}{\sqrt{1 - \frac{x_2^2}{x_1^2}}}$	(3, 10)	(1, 2)	(1, 5)	$\frac{4\pi x_1 x_2}{x_3}$	(1, 5)	(1, 5)	(1, 5)
$\frac{x_1 x_2}{2x_1}$	(1, 5)	(1, 5)	None	$\sin^2\left(\frac{2\pi x_1 x_2}{x_3}\right)$	(1, 2)	(1, 2)	(1, 4)
$x_1 + x_2 + 2\sqrt{x_1 x_2} \cos(x_3)$	(1, 5)	(1, 5)	(1, 5)	$\frac{x_1 x_2}{2\pi}$	(1, 5)	(1, 5)	None
$\frac{3x_1 x_2}{x_2}$	(1, 5)	(1, 5)	None	$2x_1 (1 - \cos(x_2 x_3))$	(1, 5)	(1, 5)	(1, 5)
$\frac{x_2 x_3}{x_1 - 1}$	(2, 5)	(1, 5)	(1, 5)	$\frac{x_2^2}{8\pi^2 x_2 x_3^2}$	(1, 5)	(1, 5)	(1, 5)
$x_1 x_2 x_3$	(1, 5)	(1, 5)	(1, 5)	$\frac{2\pi x_1}{x_2 x_3}$	(1, 5)	(1, 5)	(1, 5)
$\sqrt{\frac{x_1 x_2}{x_3}}$	(1, 5)	(1, 5)	(1, 5)	$x_1 (x_2 \cos(x_3) + 1)$	(1, 5)	(1, 5)	(1, 5)

Table 10: AI-Feynman equation with less than 4 independent variables and the supports as indicated in [Udrescu and Tegmark, 2020](#)

Expression	Expression
$4.931x_1 - x_2 + 4.023 \tan(x_1^2 - 4.027x_3)$	$x_1 \sqrt{-x_3 + \sin(x_2)}$
$\sin(\cos(3.488x_1 \tan(2.798x_1) + 2.938x_1))$	$2.29x_2 \cos(x_2) + \cos\left(\frac{1.044x_1}{x_2}\right)$
$\sqrt{-x_1 + \frac{x_2x_3}{x_1}}$	$\frac{x_3 + \frac{3.797 \sin(x_1)}{x_2}}{x_3}$
$\sin(4.84x_3(2.3x_1 - 3.494x_2 + 1))$	$x_1 - 4.843x_2x_3 + x_2 + \cos(x_3)$
$\sin(x_3) + \sin\left(\frac{x_3}{x_1 - x_2}\right)$	$\cos\left(x_1 + 1.504x_2 + (x_2 + x_3)^2\right)$
$x_1(2.683x_1 + x_2 \cos(x_3)) + 1$	$x_1(-4.641x_1 + \cos^2(4.959\sqrt{x_2}))$
$4.631 \sin\left(4.419 \sin\left(\frac{x_2x_3}{x_1^2}\right)\right)$	$x_2\left(x_2 - \frac{-x_1 - 1}{x_2}\right)$
$3.874x_3 + 4.12 - \frac{1}{x_1 + 4.322x_2x_3}$	$4.47x_1 + 1.193 \cos\left(1 + \frac{1}{x_2}\right)$
$\frac{1.858x_1x_3}{-x_1 + x_2} - 3.661x_3$	$3.63x_1 \cos(1.427x_2^3 + x_2)$
$2.846x_2 + \sin(x_1^5 + 2.258x_3)$	$-x_1(1.196x_1 + \sin(x_1 + x_2))$
$\frac{x_2 + x_3 + \frac{x_2 - 4.615}{x_2}}{x_1}$	$x_3 + \frac{x_3}{x_1 + 2.318x_2 + x_3}$
$-x_3 + \frac{0.221(-x_1 + x_2)}{\log(x_2)} - 1$	$x_1 - \frac{7.74\sqrt{0.383x_1 + x_2}}{x_2}$
$(1.261x_1 + 3.29 \cos(1)) \log(4.169x_2)$	$1 + \frac{0.221 \tan(3.972x_2)}{x_2(-3.549x_1 + x_2)}$
$\frac{x_2}{x_2 + \cos(x_1x_3)}$	$1 - \sin(x_1(x_1 + \sin(x_1)))$
$2.161x_3 \cos^3(x_1^2 + x_2)$	$\cos(x_1) - \sqrt{\cos(x_2)}$
$3.196 \tan(\cos(4.459x_1) - \tan(1)) - 1$	$-2.586x_2 + \frac{0.693 \cos(x_2 - 1)}{x_1}$
$\sqrt{x_2^2 - x_2 - e^{2.103x_1}}$	$-8.802x_1 + 3.379 \log(x_1 + x_2^4)$
$-x_2 + \log\left(x_1\left(-x_2 + \frac{1.513}{e}\right)\right)$	$\sin\left(\frac{2.696x_2}{-x_1 + 2.364x_2}\right) + 1.097$

Table 11: 36 random equations extracted from the SOOBE dataset (version with constants).

C.3 ADDITIONAL RESULTS

C.3.1 Additional Metrics on all Benchmarks

In this section, we show that the conclusions drawn in Section 4.5 with the A_2 metric are consistent when the A_1 metric is considered instead.

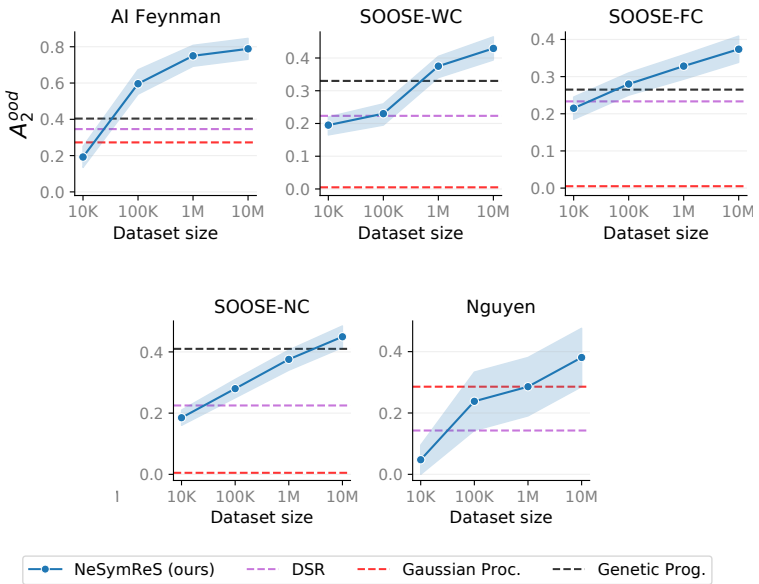


Figure 43: Accuracy as a function of the size of the pre-training dataset, for a fixed computational budget (~ 100 s) at test time.

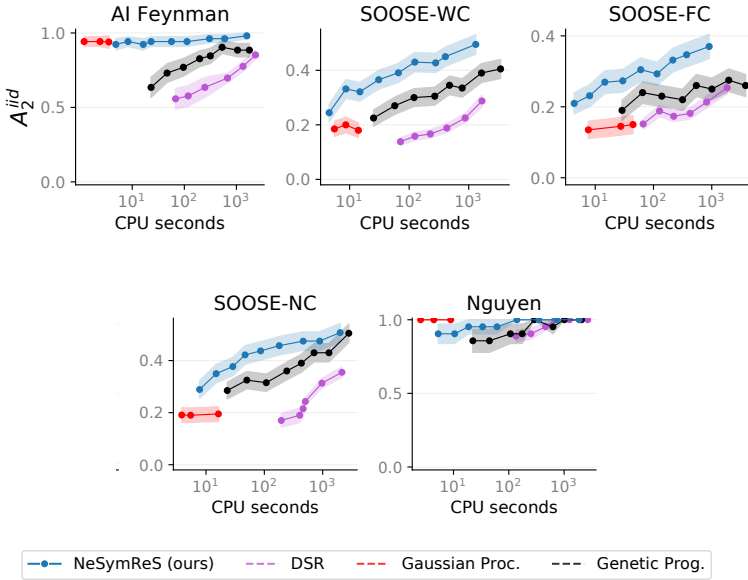


Figure 44: Accuracy in distribution as a function of time for all methods ran on a single CPU per equation.

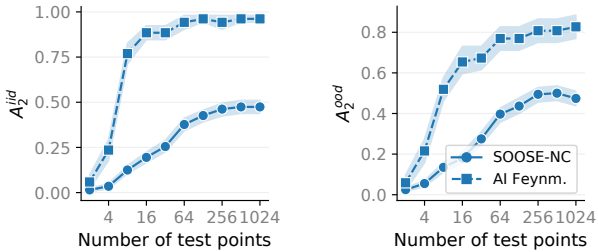


Figure 46: Accuracy as a function of number of input-output pairs observed at test time.

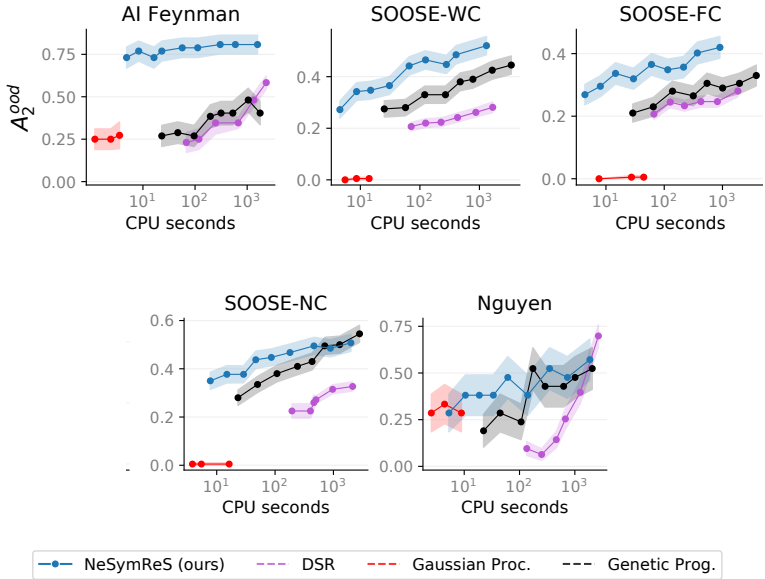


Figure 45: Accuracy out of distribution as a function of time for all methods ran on a single CPU per equation.

APPENDIX DC-MCTS

D.1 ADDITIONAL DETAILS FOR DC-MCTS

D.1.1 Proof of Proposition 1

Proof. The performance of π_σ on the task (s_0, s_∞) is defined as the probability that its trajectory $\tau_{s_0}^{\pi_\sigma}$ from initial state s_0 gets absorbed in the state s_∞ , i.e. $P(s_\infty \in \tau_{s_0}^{\pi_\sigma})$. We can bound the latter from below in the following way. Let $\sigma = (\sigma_0, \dots, \sigma_m)$, with $\sigma_0 = s_0$ and $\sigma_m = s_\infty$. With $(\sigma_0, \dots, \sigma_i) \subseteq \tau_{s_0}^{\pi_\sigma}$ we denote the event that π_σ visits all states $\sigma_0, \dots, \sigma_i$ in order:

$$P((\sigma_0, \dots, \sigma_i) \subseteq \tau_{s_0}^{\pi_\sigma}) = P\left(\bigwedge_{i'=1}^i (\sigma_{i'} \in \tau_{s_0}^{\pi_\sigma}) \wedge (t_{i'-1} < t_{i'})\right),$$

where t_i is the arrival time of π_σ at σ_i , and we define $t_0 = 0$. Obviously, the event $(\sigma_0, \dots, \sigma_m) \subseteq \tau_{s_0}^{\pi_\sigma}$ is a subset of the event $s_\infty \in \tau_{s_0}^{\pi_\sigma}$, and therefore

$$P((\sigma_0, \dots, \sigma_m) \subseteq \tau_{s_0}^{\pi_\sigma}) \leq P(s_\infty \in \tau_{s_0}^{\pi_\sigma}). \quad (14)$$

Using the chain rule of probability we can write the lhs as:

$$P((\sigma_0, \dots, \sigma_m) \subseteq \tau_{s_0}^{\pi_\sigma}) = \prod_{i=1}^m P((\sigma_i \in \tau_{s_0}^{\pi_\sigma}) \wedge (t_{i-1} < t_i) \mid (\sigma_0, \dots, \sigma_{i-1}) \subseteq \tau_{s_0}^{\pi_\sigma}).$$

We now use the definition of π_σ : After reaching σ_{i-1} and before reaching σ_i , π_σ is defined by just executing π_{σ_i} starting from the state σ_{i-1} :

$$P((\sigma_0, \dots, \sigma_m) \subseteq \tau_{s_0}^{\pi_\sigma}) = \prod_{i=1}^m P\left(\sigma_i \in \tau_{\sigma_{i-1}}^{\pi_{\sigma_i}} \mid (\sigma_0, \dots, \sigma_{i-1}) \subseteq \tau_{s_0}^{\pi_\sigma}\right).$$

We now make use of the fact that the $\sigma_i \in \mathcal{S}$ are *states* of the underlying MDP that make the future independent from the past: Having reached σ_{i-1} at t_{i-1} , all events from there on (e. g. reaching σ_j for $j \geq i$) are independent from all event before t_{i-1} . We can therefore write:

$$\begin{aligned} P((\sigma_0, \dots, \sigma_m) \subseteq \tau_{s_0}^{\pi_\sigma}) &= \prod_{i=1}^m P\left(\sigma_i \in \tau_{\sigma_{i-1}}^{\pi_{\sigma_i}}\right) \\ &= \prod_{i=1}^m v^{\pi}(\sigma_{i-1}, \sigma_i). \end{aligned} \quad (15)$$

Putting together equation 14 and equation 15 yields the proposition. \square

D.1.2 Additional algorithmic details

After the search phase, in which DC-MCTS builds the search tree \mathcal{T} , it returns its estimate of the best plan $\hat{\sigma}^*$ and the corresponding lower bound $L(\hat{\sigma}^*)$ by calling the EXTRACTPLAN procedure on the root node (s_0, s_∞) . Algorithm 5 gives details on this procedure.

D.1.3 Descending into one node at the time during search

Instead of descending into both nodes during the TRAVERSE step of Algorithm 3, it is possible to choose only one of the two sub-problems to expand further. This can be especially useful if par-

Algorithm 5 additional Divide-And-Conquer MCTS procedures

Global low-level value oracle v^π
 Global high-level value function v
 Global policy prior p
 Global search tree \mathcal{T}

```

1 procedure EXTRACTPLAN(OR node  $(s, s'')$ )
2    $s' \leftarrow \arg \max_{\hat{s}} V(s, \hat{s}) \cdot V(\hat{s}, s'')$     ▷ choose best sub-goal
3   if  $s = \emptyset$  then                                ▷ no more splitting
4     return  $\emptyset, v^\pi(s, s'')$ 
5   else
6      $\sigma_l, G_l \leftarrow \text{EXTRACTPLAN}(s, s')$     ▷ extract "left" sub-plan
7      $\sigma_r, G_r \leftarrow \text{EXTRACTPLAN}(s', s'')$     ▷ extract "right"
      sub-plan
8   return  $\sigma_l \circ \sigma_r, G_l \cdot G_r$ 

```

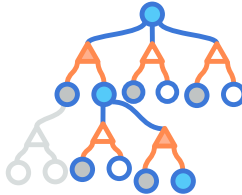
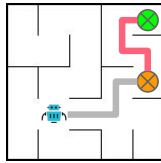
allel computation is not an option, or if there are specific needs e. g. as illustrated by the following three heuristics. These can be used to decide when to traverse into the left sub-problem (s, s') or the right sub-problem (s', s'') . Note that both nodes have a corresponding current estimate for their value V , coming either from the bootstrap evaluation of v or further refined from previous traversals.

- *Preferentially descend into the left node* encourages a more accurate evaluation of the near future, which is more relevant to the current choices of the agent. This makes sense when the right node can be further examined later, or there is uncertainty about the future that makes it sub-optimal to design a detailed plan at the moment.
- *Preferentially descend into the node with a lower value*, following the principle that a chain (plan) is only as good as its

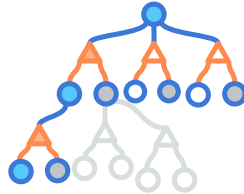
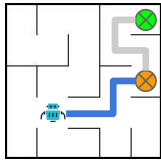
weakest link (sub-problem). This heuristic effectively greedily optimizes for the overall value of the plan.

- Use 2-way UCT on the values of the nodes, which acts similarly to the previous greedy heuristic, but also takes into account the confidence over the value estimates given by the visit counts.

Forward planning is equivalent to expanding only the **right sub-problem**



Backward planning is equivalent to expanding only the **left sub-problem**



Divide and Conquer Tree Search can do both, and also start from the middle, jump back and forth, etc.

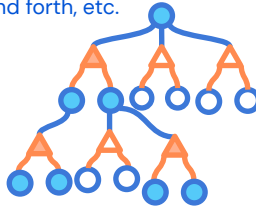
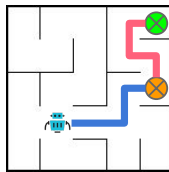


Figure 47: Divide and Conquer Tree Search is strictly more general than both forward and backward search.

The rest of the algorithm can remain unchanged, and during the BACKUP phase the current value estimate V of the sibling sub-problem can be used.

D.1.4 Parsers for Hindsight Experience Replay

Given a task (s_0, s_∞) , the policy prior p defines a distribution over binary partition trees of the task via recursive application (until the terminal symbol \emptyset closes a branch). A sample \mathcal{T}_σ from this distribution implies a plan σ as described above; but furthermore it also contains the order in which the task was partitioned. Therefore, p not only implies a distribution over plans, but also a *search order*: Trees with high probability under p will be discovered earlier in the search with DC-MCTS. For generating training targets for supervised training of p , we need to *parse* a given sequence $\tau_{s_0}^{\pi\sigma} = (s_0, s_1, \dots, s_T)$ into a binary tree. Therefore, when applying HER we are free to choose any deterministic or probabilistic parser that generates a solution tree $\mathcal{T}_{\tau_{s_0}^{\pi\sigma}}$ from re-labeled HER data $\tau_{s_0}^{\pi\sigma}$. As mentioned in the main text, the particular choice of HER-parser will shape the search strategy defined by p . Possible choices for the parsers include:

1. Left-first parsing creates triplets (s_t, s_{t+1}, s_T) . The resulting policy prior will then preferentially propose sub-goals close to the start state, mimicking standard forward planning. Analogously right-first parsing results in approximate backward planning;
2. Temporally balanced parsing creates triplets $(s_t, s_{t+\Delta/2}, s_{t+\Delta})$. The resulting policy prior will then preferentially propose sub-goals “in the middle” of the task. This is the one we used in our experiments;
3. Weight-balanced parsing creates triplets (s, s', s'') such that $v(s, s') \approx v(s', s'')$ or $v^\pi(s, s') \approx v^\pi(s', s'')$. The resulting

policy prior will attempt to propose sub-goals such that the resulting sub-tasks are equally difficult.

D.1.5 Details on Algorithmic Complexity

Let c_{v^π} denote the cost of evaluating the low-level value v^π on any sub-problem (s, s') . We assume c_{v^π} to be independent of (s, s') which holds if e.g. v^π is a fixed size neural network. Denote the cost of evaluating the policy prior p on a sub-goal (s, s', s'') with c_p . Expanding a new OR node in the search tree incurs a cost of $|\mathcal{S}|c_p$ for evaluating p for all children. Assuming the computational cost of tree traversals is negligible, the total cost of running DC-MCTS for N node expansions is thus $N \cdot (|\mathcal{S}|c_p + 2c_{v^\pi})$. The number of expansions to find the optimal (or a sufficiently good) plan strongly depends on the quality of the policy prior (similar to A^* search), making an analysis of the complexity of DC-MCTS challenging for arbitrary p . However, if $p = p^*$ is the optimal policy prior – i. e. $p^*(s'|s, s'') = 1$ if $s' \in \sigma^*$ is in the optimal plan σ^* for (s, s'') and 0 otherwise – DC-MCTS will construct σ^* in the minimal number of step $N = |\sigma^*|$, therefore incurring a cost of $|\sigma^*| \cdot (|\mathcal{S}|c_p + 2c_{v^\pi})$. The dependency on $|\mathcal{S}|$ for the policy prior can be further reduced — in principle down to a constant — using techniques from the literature on MCTS for continuous or large discrete action spaces (e.g. *progressive widening* [Coulom, 2007](#); «[Progressive strategies for monte-carlo tree search](#)»). We can compare this to the cost of unguided, standard SSSP planners, which is $|\mathcal{S}|^2c_{v^\pi}$ as they need to query the low-level value function for all pairs of states (s, s') . Therefore, DC-MCTS can be significantly more cost efficient than conventional SSSP planners if a good policy prior can be learned and the cost of evaluating $c_p \lesssim c_{v^\pi}$ is at least comparable to that of evaluating v^π . Under the assumption $p = p^*$, MCTS and DC-MCTS have the same sample complexity. However MCTS represents the solution as one path

of length $|\sigma^*|$ in the search tree, whereas DC-MCTS presents it as a sub-tree with $|\sigma^*|$ nodes. Therefore, if computation can be carried out in parallel (e.g. by batching independent sub-problems at the same level of the sub-tree), the time complexity of DC-MCTS could be drastically reduced compared to MCTS, in the best case (perfect parallelism and balanced binary solution tree) from $\mathcal{O}(|\sigma^*|)$ to $\mathcal{O}(\log(|\sigma^*|))$.

D.2 TRAINING DETAILS

D.2.1 *Details for training the value function*

In order to train the value network v , that is used for bootstrapping in DC-MCTS, we can regress it towards targets computed from previous search results or environment experiences. A first obvious option is to use as regression target the Monte Carlo return (i.e. 0 if the goal was reached, and 1 if it was not) from executing the DC-MCTS plans in the environment. This appears to be a sensible target, as the return is an unbiased estimator of the success probability $P(s_\infty \in \tau_{s_0}^{\sigma})$ of the plan. Although this approach was used in [Silver et al., 2016](#), its downside is that gathering environment experience is often very costly and only yields little information, i.e. one binary variable per episode. Furthermore no other information from the generated search tree \mathcal{T} except for the best plan is used. Therefore, a lot of valuable information might be discarded, in particular in situations where a good sub-plan for a particular sub-problem was found, but the overall plan nevertheless failed.

This shortcoming could be remedied by using as regression targets the non-parametric value estimates $V(s, s'')$ for all OR nodes (s, s'') in the DC-MCTS tree at the end of the search. With this approach, a learning signal could still be obtained from successful sub-plans of an overall failed plan. However, we empiri-

cally found in our experiments that this lead to drastically over-optimistic value estimates, for the following reason. By standard policy improvement arguments, regressing toward V leads to a bootstrap value function that converges to v^* . In the definition of the optimal value $v^*(s, s'') = \max_{s'} v^*(s, s') \cdot v^*(s', s'')$, we implicitly allow for infinite recursion depth for solving sub-problems. However, in practice, we often used quite shallow trees (depth < 10), so that bootstrapping with approximations of v^* is too optimistic, as this assumes unbounded planning budget. A principled solution for this could be to condition the value function for bootstrapping on the amount of remaining search budget, either in terms of remaining tree depth or node expansions.

Instead of the cumbersome, explicitly resource-aware value function, we found the following to work well. After planning with DC-MCTS, we extract the plan $\hat{\sigma}^*$ with `EXTRACTPLAN` from the search tree \mathcal{T} . As can be seen from Algorithm 5, the procedure computes the return $G_{\hat{\sigma}^*}$ for all OR nodes in the solution tree $\mathcal{T}_{\hat{\sigma}^*}$. For training v we chose these returns $G_{\hat{\sigma}^*}$ for all OR nodes in the solution tree as regression targets. This combines the favourable aspects of both methods described above. In particular, this value estimate contains no bootstrapping and therefore did not lead to overly-optimistic bootstraps. Furthermore, all successfully solved sub-problems given a learning signal. As regression loss we chose cross-entropy.

D.2.2 *Details for training the policy prior*

The prior network is trained to match the distribution of the values of the AND nodes, also with a cross-entropy loss. Note that we did not use visit counts as targets for the prior network — as done in AlphaGo and AlphaZero for example (Silver et al., 2016; Silver et al., 2018)— since for small search budgets visit counts

tend to be noisy and require significant fine-tuning to avoid collapse (Hamrick et al., 2020).

D.2.3 *Neural networks architectures for grid-world experiments*

The shared torso of the prior and value network used in the experiments is a 6-layer CNN with kernels of size 3, 64 filters per layer, Layer Normalization after every convolutional layer, swish (cit) as activation function, zero-padding of 1, and strides [1, 1, 2, 1, 1, 2] to increase the size of the receptive field.

The two heads for the prior and value networks follow the pattern described above, but with three layers only instead of six, and fixed strides of 1. The prior head ends with a linear layer and a softmax, in order to obtain a distribution over sub-goals. The value head ends with a linear layer and a sigmoid that predicts a single value, i.e. the probability of reaching the goal from the start state if we further split the problem into sub-problems.

We did not heavily optimize networks hyper-parameters. After running a random search over hyper-parameters for the fixed architecture described above, the following were chosen to run the experiments in Figure 29. The replay buffer has a maximum size of 2048. The prior and value networks are trained on batches of size 128 as new experiences are collected. Networks are trained using Adam with a learning rate of $1e-3$, the boltzmann temperature of the softmax for the prior network set to 0.003. For simplicity, we used HER with the time-based rebalancing (i.e. turning experiences into temporal binary search trees). UCB constants are sampled uniformly between 3 and 7, as these values were observed to give more robust results.

Table 12: Architectures of the neural networks used in the experiment section for the high-level value and prior. For each convolutional layer we report kernel size, number of filters and stride. LN stands for Layer normalization, FC for fully connected,. All convolutions are preceded by a 1 pixel zero padding.

	Torso	
Value head	$3 \times 3, 64$, stride: 1	Policy head
	swish, LN	
$3 \times 3, 64$, stride: 1	$3 \times 3, 64$, stride: 1	$3 \times 3, 64$, stride: 1
swish, LN	swish, LN	swish, LN
$3 \times 3, 64$, stride: 1	$3 \times 3, 64$, stride: 2	$3 \times 3, 64$, stride: 1
swish, LN	swish, LN	swish, LN
$3 \times 3, 64$, stride: 1	$3 \times 3, 64$, stride: 1	$3 \times 3, 64$, stride: 1
swish, LN	swish, LN	swish, LN
Flatten	$3 \times 3, 64$, stride: 1	Flatten
FC $N_h : 1$	swish, LN	FC $N_h : \#classes$
sigmoid	$3 \times 3, 64$, stride: 2	softmax
	swish, LN	

D.2.4 Low-level controller training details

For physics-based experiments using MuJoCo (Todorov et al., 2012), we trained a low-level policy first and then trained the planning agent to reuse the low-level motor skills afforded by this body and pretrained policy. The low-level policy, was trained to control the quadruped (“ant”) body to go to a randomly placed target in an open area (a “go-to-target” task, essentially the same as the task used to train the humanoid in Merel et al., 2019, which is available at https://github.com/deepmind/dm_control/

`tree/master/dm_control/locomotion`). The task amounts to the environment providing an *instruction* corresponding to a target position that the agent is rewarded for moving to (i.e., a sparse reward when within a region of the target). When the target is obtained, a new target is generated that is a short distance away (<1.5m). What this means is that a policy trained on this task should be capable of producing short, direct, goal-directed locomotion behaviors in an open field. And at test time, the presence of obstacles will catastrophically confuse the trained low-level policy. The policy architecture, consisting of a shallow MLP for the actor and critic, was trained to solve this task using MPO [Abdolmaleki et al., 2018](#). More specifically, the actor and critic had respectively 2 and 3 hidden layers, 256 units each and elu activation function. The policy was trained to a high level of performance using a distributed, replay-based, off-policy training setup involving 64 actors. In order to reuse the low-level policy in the context of mazes, we can replace the environment-provided instruction with a message sent by a high-level policy (i.e., the planning agent). For the planning agent that interfaces with the low-level policy, the action space of the high-level policy will, by construction, correspond to the instruction to the low-level policy.

D.2.5 Pseudocode

We summarize the training procedure for DC-MCTS in the following pseudo-code.

```
def train_DCMCTS():
    replay_buffer = []

    for episode in n_episodes:
        start, goal = env.reset()
        sub_goals = dc_mcts_plan(start, goal) # list of sub
                                             -goals
        replay_buffer.add(sub_goals)
```

```

state = start
while episode.not_over() & len(sub_goals) > 0:
    action = low_level_policy(state, sub_goals[0])
    state = env.step(action)
    visited_states.append(state)

    if state == sub_goals[0]:
        sub_goals.pop(0)

    # Rebalance list of visited states to a binary
    #                               search tree
bst_states = bst_from_states(visited_states)
replay_buffer.add(bst_states) # Hindsight
                               Experience Replay

if replay_buffer.can_sample():
    neural_nets.train(replay_buffer.sample())

```

D.3 MORE SOLVED MAZES

In Figure 48 we show more mazes as solved by the trained Divide and Conquer MCTS.

D.3.1 *Supplementary material and videos*

Additional material, including videos of several grid-world mazes as solved by the algorithm and of MuJoCo low-level policy solving mazes by following DC-MCTS plans, can be found on <https://sites.google.com/view/dc-mcts/home>.

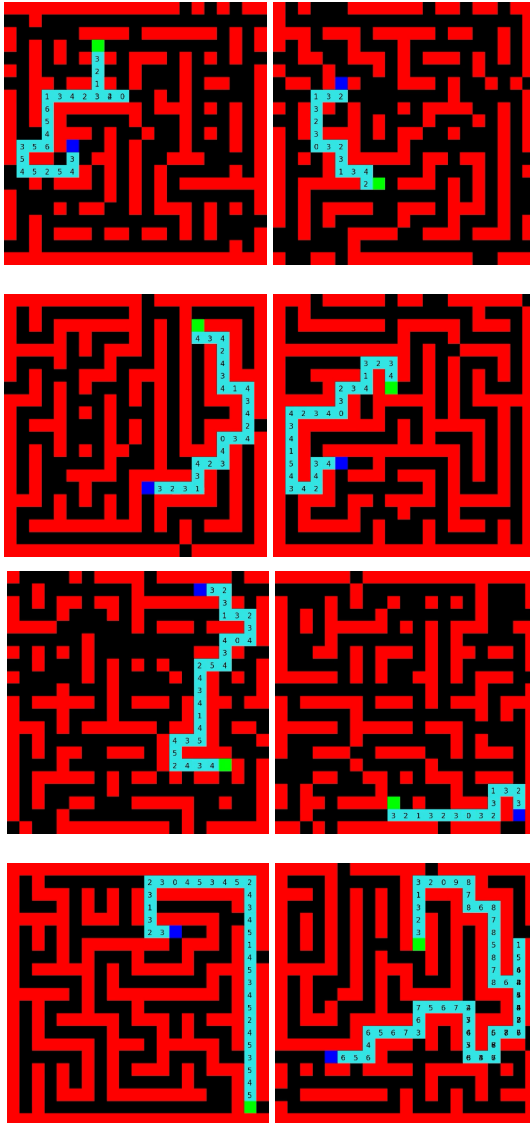


Figure 48: Solved mazes with Divide and Conquer MCTS.
 ■ = start, ■ = goal, ■ = wall, ■ = walkable. Overlapping numbers are due to the agent back-tracking while refining finer sub-goals.

BIBLIOGRAPHY

- Abdolmaleki, Abbas, Jost Tobias Springenberg, Yuval Tassa, Rémi Munos, Nicolas Heess, and Martin A. Riedmiller (2018). «Maximum a Posteriori Policy Optimisation.» In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.
- Adolphs, Leonard, Jonas Kohler, and Aurelien Lucchi (2019). «Ellipsoidal trust region methods and the marginal value of Hessian information for neural network training.» In: *arXiv preprint arXiv:1905.09201 (version 1)*.
- Ahuja, Kartik, Karthikeyan Shanmugam, Kush R. Varshney, and Amit Dhurandhar (2020). «Invariant Risk Minimization Games.» In: *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*. Vol. 119. Proceedings of Machine Learning Research. PMLR.
- Akkaya, Ilge, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, et al. (2019). «Solving rubik’s cube with a robot hand.» In: *arXiv preprint arXiv:1910.07113*.
- Alaa, Ahmed M. and Mihaela van der Schaar (2019). «Demystifying Black-box Models with Symbolic Metamodels.» In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*.
- Aljundi, Rahaf, Punarjay Chakravarty, and Tinne Tuytelaars (2017). «Expert Gate: Lifelong Learning with a Network of Experts.» In: *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*. IEEE Computer Society.

- Ando, Tsuyoshi, Chi-Kwong Li, and Roy Mathias (2004). «Geometric means.» In: *Linear algebra and its applications* 385.
- Andrychowicz, Marcin, Dwight Crow, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba (2017). «Hindsight Experience Replay.» In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*.
- Arjovsky, Martin, Léon Bottou, Ishaan Gulrajani, and David Lopez-Paz (2019). «Invariant risk minimization.» In: *arXiv preprint arXiv:1907.02893*.
- Balaji, Yogesh, Swami Sankaranarayanan, and Rama Chellappa (2018). «MetaReg: Towards Domain Generalization using Meta-Regularization.» In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*.
- Becker, Sue, Yann Le Cun, et al. (1988). «Improving the convergence of back-propagation learning with second order methods.» In: *Proceedings of the 1988 connectionist models summer school*.
- Belkin, Mikhail, Daniel Hsu, Siyuan Ma, and Soumik Mandal (2019). «Reconciling modern machine-learning practice and the classical bias–variance trade-off.» In: *Proceedings of the National Academy of Sciences* 116.32.
- Bengio, Yoshua, Tristan Deleu, Nasim Rahaman, Rosemary Ke, Sébastien Lachapelle, Olexa Bilaniuk, Anirudh Goyal, and Christopher Pal (2019). «A meta-transfer objective for learning to disentangle causal mechanisms.» In: *arXiv 1901.10912*.
- Bergstra, James, Daniel Yamins, and David D. Cox (2013). «Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures.» In: *Proceedings of the 30th International Conference on Machine*

- Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*. Vol. 28. JMLR Workshop and Conference Proceedings. JMLR.org.
- Bertsekas, Dimitri P, Dimitri P Bertsekas, Dimitri P Bertsekas, and Dimitri P Bertsekas (1995). *Dynamic programming and optimal control*. Vol. 1. 2. Athena scientific Belmont, MA.
- Biggio, L., T. Bendinelli, A. Neitz, A. Lucchi, and G. Parascandolo (2021). «Neural Symbolic Regression that Scales.» In: *38th International Conference on Machine Learning*. *equal contribution.
- Bishop, Christopher M et al. (1995). *Neural networks for pattern recognition*. Oxford university press.
- Bottou, Léon, Frank E Curtis, and Jorge Nocedal (2018). «Optimization methods for large-scale machine learning.» In: *Siam Review* 60.2.
- Bousmalis, Konstantinos, Nathan Silberman, David Dohan, Dumitru Erhan, and Dilip Krishnan (2017). «Unsupervised Pixel-Level Domain Adaptation with Generative Adversarial Networks.» In: *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*. IEEE Computer Society.
- Brown, Tom B. et al. (2020). «Language Models are Few-Shot Learners.» In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.
- Browne, Cameron B, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton (2012). «A survey of monte carlo tree search methods.» In: *IEEE Transactions on Computational Intelligence and AI in games* 4.1.
- Chaslot, GMJB, Mark Winands, and Bruno Bouzy. «Progressive strategies for monte-carlo tree search.» In: *In Proceedings of the 10th Joint Conference on Information Sciences (JCIS 2007)*.

- Chen, Binghong, Chengtao Li, Hanjun Dai, and Le Song (2020a). «Retro*: Learning Retrosynthetic Planning with Neural Guided A* Search.» In: *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*. Vol. 119. Proceedings of Machine Learning Research. PMLR.
- Chen, Ting, Simon Kornblith, Mohammad Norouzi, and Geoffrey E. Hinton (2020b). «A Simple Framework for Contrastive Learning of Visual Representations.» In: *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*. Vol. 119. Proceedings of Machine Learning Research. PMLR.
- Chen, Ting, Simon Kornblith, Kevin Swersky, Mohammad Norouzi, and Geoffrey E. Hinton (2020c). «Big Self-Supervised Models are Strong Semi-Supervised Learners.» In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.
- Chen, Xi, Yan Duan, Rein Houthoofd, John Schulman, Ilya Sutskever, and Pieter Abbeel (2016). «InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets.» In: *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*.
- Clevert, Djork-Arné, Thomas Unterthiner, and Sepp Hochreiter (2016). «Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs).» In: *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*.
- Cobbe, Karl, Oleg Klimov, Christopher Hesse, Taehoon Kim, and John Schulman (2019). «Quantifying Generalization in Reinforcement Learning.» In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019,*

- Long Beach, California, USA*. Vol. 97. Proceedings of Machine Learning Research. PMLR.
- Cobbe, Karl, Christopher Hesse, Jacob Hilton, and John Schulman (2020). «Leveraging Procedural Generation to Benchmark Reinforcement Learning.» In: *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*. Vol. 119. Proceedings of Machine Learning Research. PMLR.
- Coulom, Rémi (2007). «Computing “ELO ratings” of move patterns in the game of Go.» In: *ICGA journal* 30.4.
- Cranmer, Miles D., Alvaro Sanchez-Gonzalez, Peter W. Battaglia, Rui Xu, Kyle Cranmer, David N. Spergel, and Shirley Ho (2020). «Discovering Symbolic Models from Deep Learning with Inductive Biases.» In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.
- Cybenko, George (1989). «Approximation by superpositions of a sigmoidal function.» In: *Mathematics of control, signals and systems* 2.4.
- Daniusis, Povilas, Dominik Janzing, Joris M. Mooij, Jakob Zscheischler, Bastian Steudel, Kun Zhang, and Bernhard Schölkopf (2010). «Inferring deterministic causal relations.» In: *UAI 2010, Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence, Catalina Island, CA, USA, July 8-11, 2010*. AUAI Press.
- Dayan, Peter and Geoffrey E Hinton (1993). «Feudal reinforcement learning.» In: *Advances in neural information processing systems*.
- Dechter, Rina and Robert Mateescu (2004). «Mixtures of deterministic-probabilistic networks and their AND/OR search space.» In: *Proceedings of the 20th conference on Uncertainty in artificial intelligence*. AUAI Press.

- Deutsch, David (2011). *The beginning of infinity: Explanations that transform the world*. Penguin UK.
- Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova (2019). «BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.» In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Association for Computational Linguistics.
- Dhiman, Vikas, Shurjo Banerjee, Jeffrey M Siskind, and Jason J Corso (2018). «Floyd-Warshall Reinforcement Learning Learning from Past Experiences to Reach New Goals.» In: *arXiv preprint arXiv:1809.09318*.
- Du, Yunshu, Wojciech M Czarnecki, Siddhant M Jayakumar, Razvan Pascanu, and Balaji Lakshminarayanan (2018). «Adapting auxiliary losses using gradient similarity.» In: *arXiv preprint arXiv:1812.02224*.
- Dubčáková, Renáta (2011). *Eureqa: software review*.
- Eshratifar, Amir Erfan, David Eigen, and Massoud Pedram (2018). «Gradient Agreement as an Optimization Objective for Meta-Learning.» In: *arXiv preprint arXiv:1810.08178*.
- Fletcher, Roger (1987). *Practical Methods of Optimization*. Second. John Wiley & Sons.
- Forrest, Stephanie (1993). «Genetic algorithms: principles of natural selection applied to computation.» In: *Science* 261.5123.
- Fort, Stanislav, Paweł Krzysztof Nowak, and Srini Narayanan (2019). «Stiffness: A New Perspective on Generalization in Neural Networks.» In: *arXiv preprint arXiv:1901.09491*.
- Freuder, Eugene C and Michael J Quinn (1985). «Taking Advantage of Stable Sets of Variables in Constraint Satisfaction Problems.» In: *IJCAI*. Vol. 85. Citeseer.
- Gabor, Thomas, Jan Peter, Thomy Phan, Christian Meyer, and Claudia Linnhoff-Popien (2019). «Subgoal-Based Temporal Ab-

- traction in Monte-Carlo Tree Search.» In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*. ijcai.org.
- Ganin, Yaroslav, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky (2016). «Domain-adversarial training of neural networks.» In: *The Journal of Machine Learning Research* 17.1.
- Gao, Wei, David Hsu, Wee Sun Lee, Shengmei Shen, and Karthikk Subramanian (2017). «Intention-net: Integrating planning and deep learning for goal-directed autonomous navigation.» In: *arXiv preprint arXiv:1710.05627*.
- Gebhard, T., N. Kilbertus, G. Parascandolo, I. Harry, and B. Schölkopf (2017). «ConvWave: Searching for Gravitational Waves with Fully Convolutional Neural Nets.» In: *Workshop on Deep Learning for Physical Sciences (DLPS) at the 31st Conference on Neural Information Processing Systems*.
- Ghosh, Dibya, Abhishek Gupta, and Sergey Levine (2019). «Learning Actionable Representations with Goal Conditioned Policies.» In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.
- Goodfellow, Ian J., Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio (2014). «Generative Adversarial Nets.» In: *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*.
- Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep learning*. MIT press.
- Haavelmo, T. (1943). «The statistical implications of a system of simultaneous equations.» In: *Econometrica* 11.1.

- Hamrick, Jessica B., Victor Bapst, Alvaro Sanchez-Gonzalez, Tobias Pfaff, Theophane Weber, Lars Buesing, and Peter W. Battaglia (2020). «Combining Q-Learning and Search with Amortized Value Estimates.» In: *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.
- Hansen, Nikolaus (2016). «The CMA evolution strategy: A tutorial.» In: *arXiv preprint arXiv:1604.00772*.
- Heinze-Deml, Christina and Nicolai Meinshausen (2017). «Conditional variance penalties and domain shift robustness.» In: *arXiv preprint arXiv:1769*.
- Heinze-Deml, Christina, Jonas Peters, and Nicolai Meinshausen (2018). «Invariant causal prediction for nonlinear models.» In: *Journal of Causal Inference* 6.2.
- Henzinger, Monika R, Philip Klein, Satish Rao, and Sairam Subramanian (1997). «Faster shortest-path algorithms for planar graphs.» In: *journal of computer and system sciences* 55.1.
- Hermann, Katherine L. and Andrew K. Lampinen (2020). «What shapes feature representations? Exploring datasets, architectures, and training.» In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.
- Higgins, Irina, Loïc Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner (2017). «beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework.» In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.
- Hoover, Kevin D (1990). «The logic of causal inference: Econometrics and the conditional analysis of causation.» In: *Economics & Philosophy* 6.2.

- Hurwicz, L (1962). «On the structural form of interdependent systems.» In: *Logic, Methodology and Philosophy of Science, Proceedings of the 1960 International Congress*. Stanford University Press.
- Hyvärinen, Aapo and Hiroshi Morioka (2016). «Unsupervised Feature Extraction by Time-Contrastive Learning and Nonlinear ICA.» In: *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*.
- Ioffe, Sergey and Christian Szegedy (2015). «Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.» In: *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*. Vol. 37. JMLR Workshop and Conference Proceedings. JMLR.org.
- Jacobs, Robert A, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton (1991). «Adaptive mixtures of local experts.» In: *Neural computation* 3.1.
- Janzing, D. and B. Schölkopf (2010). «Causal inference using the algorithmic Markov condition.» In: *IEEE Transactions on Information Theory* 56.10.
- Janzing, Dominik (2019). «Causal Regularization.» In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*.
- Jastrzębski, Stanisław, Zachary Kenton, Devansh Arpit, Nicolas Ballas, Asja Fischer, Yoshua Bengio, and Amos Storkey (2017). «Three factors influencing minima in sgd.» In: *arXiv preprint arXiv:1711.04623*.
- Jégou, Philippe and Cyril Terrioux (2003). «Hybrid backtracking bounded by tree-decomposition of constraint networks.» In: *Artificial Intelligence* 146.1.

- Jordan, Michael I and Robert A Jacobs (1994). «Hierarchical mixtures of experts and the EM algorithm.» In: *Neural computation* 6.2.
- Jumper, John, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. (2021). «Highly accurate protein structure prediction with AlphaFold.» In: *Nature*.
- Jurgenson, Tom, Edward Groshev, and Aviv Tamar (2019). «Sub-Goal Trees—a Framework for Goal-Directed Trajectory Prediction and Optimization.» In: *arXiv preprint arXiv:1906.05329*.
- Kaelbling, Leslie Pack (1993). «Learning to achieve goals.» In: *IJ-CAI*. Citeseer.
- Kaelbling, Leslie Pack and Tomás Lozano-Pérez (2017). «Learning composable models of parameterized skills.» In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE.
- Kaplan, Jared, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei (2020). «Scaling laws for neural language models.» In: *arXiv preprint arXiv:2001.08361*.
- Kilbertus*, N., G. Parascandolo*, and B. Schölkopf* (2018). «Generalization in anti-causal learning.» In: *NeurIPS 2018 Workshop on Critiquing and Correcting Trends in Machine Learning*. *authors are listed in alphabetical order.
- Kilbertus, Niki, Mateo Rojas-Carulla, Giambattista Parascandolo, Moritz Hardt, Dominik Janzing, and Bernhard Schölkopf (2017). «Avoiding Discrimination through Causal Reasoning.» In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*.
- Kingma, Diederik P. and Jimmy Ba (2015). «Adam: A Method for Stochastic Optimization.» In: *3rd International Conference on*

- Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings.*
- Kingma, Diederik P. and Max Welling (2014). «Auto-Encoding Variational Bayes.» In: *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings.*
- Kishimoto, Akihiro, Beat Buesser, Bei Chen, and Adi Botea (2019). «Depth-First Proof-Number Search with Heuristic Edge Cost and Application to Chemical Synthesis Planning.» In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada.*
- Kolesnikov, Alexander, Lucas Beyer, Xiaohua Zhai, Joan Puigcerver, Jessica Yung, Sylvain Gelly, and Neil Houlsby (2020). *Big Transfer (BiT): General Visual Representation Learning.* arXiv: 1912.11370 [cs.CV].
- Koza, John R (1994). «Genetic programming as a means for programming computers by natural selection.» In: *Statistics and computing* 4.2.
- Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton (2012). «ImageNet Classification with Deep Convolutional Neural Networks.» In: *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States.*
- Krueger, David, Ethan Caballero, Joern-Henrik Jacobsen, Amy Zhang, Jonathan Binas, Remi Le Priol, and Aaron Courville (2020). «Out-of-distribution generalization via risk extrapolation (rex).» In: *arXiv preprint arXiv:2003.00688.*
- Kügelgen, Julius von, Alexander Mey, and Marco Loog (2019). «Semi-Generative Modelling: Covariate-Shift Adaptation with Cause and Effect Features.» In: *The 22nd International Conference on Artificial Intelligence and Statistics, AISTATS 2019, 16-18*

- April 2019, Naha, Okinawa, Japan*. Vol. 89. Proceedings of Machine Learning Research. PMLR.
- Kulkarni, Tejas D., Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum (2016). «Hierarchical Deep Reinforcement Learning: Integrating Temporal Abstraction and Intrinsic Motivation.» In: *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*.
- Kusner, Matt J., Brooks Paige, and José Miguel Hernández-Lobato (2017). «Grammar Variational Autoencoder.» In: *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*. Vol. 70. Proceedings of Machine Learning Research. PMLR.
- Lake, Brenden M, Ruslan Salakhutdinov, and Joshua B Tenenbaum (2015). «Human-level concept learning through probabilistic program induction.» In: *Science* 350.6266.
- Lample, Guillaume and François Charton (2020). «Deep Learning For Symbolic Mathematics.» In: *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.
- Larrosa, Javier, Pedro Meseguer, and Marti Sánchez (2002). «Pseudo-tree search with soft constraints.» In: *ECAI*.
- LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton (2015). «Deep learning.» In: *nature* 521.7553, pp. 436-444.
- Ledeniov, Oleg and Shaul Markovitch (1998). «The divide-and-conquer subgoal-ordering algorithm for speeding up logic inference.» In: *Journal of Artificial Intelligence Research* 9.
- Lee, Jason D, Max Simchowitz, Michael I Jordan, and Benjamin Recht (2016a). «Gradient descent converges to minimizers.» In: *arXiv preprint arXiv:1602.04915*.
- Lee, Juho, Yoonho Lee, Jungtaek Kim, Adam Kosioerek, Seungjin Choi, and Yee Whye Teh (2019). «Set transformer: A framework for attention-based permutation-invariant neural net-

- works.» In: *International Conference on Machine Learning*. PMLR, pp. 3744–3753.
- Lee, Stefan, Senthil Purushwalkam, Michael Cogswell, Viresh Ranjan, David J. Crandall, and Dhruv Batra (2016b). «Stochastic Multiple Choice Learning for Training Diverse Deep Ensembles.» In: *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5–10, 2016, Barcelona, Spain*.
- Little, Iain and Sylvie Thiébaux (2007). «Probabilistic planning vs. replanning.» In: *In ICAPS Workshop on IPC: Past, Present and Future*. Citeseer.
- Mandt, Stephan, Matthew D Hoffman, and David M Blei (2017). «Stochastic gradient descent as approximate bayesian inference.» In: *The Journal of Machine Learning Research* 18.1.
- Marinescu, Radu and Rina Dechter (2004). «AND/OR tree search for constraint optimization.» In: *Proc. of the 6th International Workshop on Preferences and Soft Constraints*. Citeseer.
- Martius, Georg and Christoph H Lampert (2016). «Extrapolation and learning equations.» In: *arXiv preprint arXiv:1610.02995*.
- Merel, Josh, Arun Ahuja, Vu Pham, Saran Tunyasuvunakool, Siqi Liu, Dhruva Tirumala, Nicolas Heess, and Greg Wayne (2019). «Hierarchical Visuomotor Control of Humanoids.» In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6–9, 2019*. OpenReview.net.
- Meurer, Aaron et al. (2017). «SymPy: symbolic computing in Python.» In: *PeerJ Computer Science* 3. ISSN: 2376-5992.
- Mooij, Joris M., Dominik Janzing, Jonas Peters, and Bernhard Schölkopf (2009). «Regression by dependence minimization and its application to causal inference in additive noise models.» In: *Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009, Montreal, Quebec, Canada, June 14–18, 2009*. Vol. 382. ACM International Conference Proceeding Series. ACM.

- Muandet, Krikamol, David Balduzzi, and Bernhard Schölkopf (2013). «Domain Generalization via Invariant Feature Representation.» In: *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*. Vol. 28. JMLR Workshop and Conference Proceedings. JMLR.org.
- Nasiriany, Soroush, Vitchyr Pong, Steven Lin, and Sergey Levine (2019). «Planning with Goal-Conditioned Policies.» In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*.
- Neitz, A., G. Parascandolo, and B. Schölkopf (2021). «A teacher-student framework to distill future trajectories.» In: *9th International Conference on Learning Representations (ICLR)*. *equal contribution.
- Neitz, Alexander, Giambattista Parascandolo, Stefan Bauer, and Bernhard Schölkopf (2018). «Adaptive Skip Intervals: Temporal Abstraction for Recurrent Dynamical Models.» In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*.
- Nilsson, N. J. (1980). *Principles of Artificial Intelligence*. Morgan Kaufmann Publishers Inc. ISBN: 0-934613-10-9.
- Nocedal, Jorge and Stephen Wright (2006). *Numerical optimization*. Springer Science & Business Media.
- Nowak, Alex, David Folqué, and Joan Bruna (2018). «Divide and Conquer Networks.» In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.
- Oord, Aaron van den, Yazhe Li, and Oriol Vinyals (2018a). «Representation learning with contrastive predictive coding.» In: *arXiv preprint arXiv:1807.03748*.
- Oord, Aäron van den et al. (2018b). «Parallel WaveNet: Fast High-Fidelity Speech Synthesis.» In: *Proceedings of the 35th Interna-*

- tional Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. Vol. 80. Proceedings of Machine Learning Research. PMLR.
- Parascandolo, G., A. Neitz, A. Orvieto, L. Gresele, and B. Schölkopf (2021a). «Learning explanations that are hard to vary.» In: *9th International Conference on Learning Representations (ICLR)*. *equal contribution.
- Parascandolo, Giambattista, Niki Kilbertus, Mateo Rojas-Carulla, and Bernhard Schölkopf (2018). «Learning Independent Causal Mechanisms.» In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. Vol. 80. Proceedings of Machine Learning Research. PMLR.
- Parascandolo, Giambattista, Lars Holger Buesing, Josh Merel, Leonard Hasenclever, John Aslanides, Jessica B Hamrick, Nicolas Heess, Alexander Neitz, and Theophane Weber (2021b). *Divide-and-Conquer Monte Carlo Tree Search*.
- Paszke, Adam, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer (2017). *Automatic differentiation in pytorch*.
- Pearl, J. (2009). *Causality: Models, Reasoning, and Inference*. 2nd. Cambridge University Press.
- Pearl, Judea. (2000). *Causality*. Cambridge University Press.
- Peters, J., D. Janzing, and B. Schölkopf (2017a). *Elements of Causal Inference - Foundations and Learning Algorithms*. MIT Press.
- Peters, Jonas, Peter Bühlmann, and Nicolai Meinshausen (2016). «Causal inference by using invariant prediction: identification and confidence intervals.» In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 78.5.
- Peters, Jonas, Dominik Janzing, and Bernhard Schölkopf (2017b). *Elements of causal inference: foundations and learning algorithms*. The MIT Press.

- Petersen, Brenden K (2021). «Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients.» In: *International Conference on Learning Representations*.
- Polu, Stanislas and Ilya Sutskever (2020). *Generative Language Modeling for Automated Theorem Proving*. arXiv: 2009.03393 [cs.LG].
- Quionero-Candela, Joaquin, Masashi Sugiyama, Anton Schwaighofer, and Neil D Lawrence (2009). *Dataset shift in machine learning*. The MIT Press.
- Radford, Alec et al. (2021). *Learning Transferable Visual Models From Natural Language Supervision*. arXiv: 2103.00020 [cs.CV].
- Ramesh, Aditya, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever (2021). «Zero-shot text-to-image generation.» In: *arXiv preprint arXiv:2102.12092*.
- Rasmussen, Carl Edward (2003). «Gaussian processes in machine learning.» In: *Summer School on Machine Learning*. Springer.
- Rojas-Carulla, Mateo, Bernhard Schölkopf, Richard Turner, and Jonas Peters (2015). «Causal transfer in machine learning.» In: *arXiv:1507.05333*.
- (2018). «Invariant models for causal transfer learning.» In: *The Journal of Machine Learning Research* 19.1.
- Rosin, Christopher D (2011). «Multi-armed bandits with episode context.» In: *Annals of Mathematics and Artificial Intelligence* 61.3.
- Sahoo, Subham S., Christoph H. Lampert, and Georg Martius (2018). «Learning Equations for Extrapolation and Control.» In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. Vol. 80. Proceedings of Machine Learning Research. PMLR.
- Sajjadi, Mehdi S. M., Giambattista Parascandolo, Arash Mehrjou, and Bernhard Schölkopf (2018). «Tempered Adversarial Networks.» In: *Proceedings of the 35th International Conference on*

- Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. Vol. 80. Proceedings of Machine Learning Research. PMLR.
- Savinov, Nikolay, Alexey Dosovitskiy, and Vladlen Koltun (2018). «Semi-parametric topological memory for navigation.» In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.
- Saxton, David, Edward Grefenstette, Felix Hill, and Pushmeet Kohli (2019). «Analysing Mathematical Reasoning Abilities of Neural Models.» In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.
- Schaul, Tom, Daniel Horgan, Karol Gregor, and David Silver (2015). «Universal Value Function Approximators.» In: *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*. Vol. 37. JMLR Workshop and Conference Proceedings. JMLR.org.
- Schmidt, Michael and Hod Lipson (2009). «Distilling free-form natural laws from experimental data.» In: *science* 324.5923.
- Schölkopf, B. (2019). *Causality for Machine Learning*. arXiv:1911.10500.
- Schölkopf, Bernhard, Dominik Janzing, Jonas Peters, Eleni Sgouritsa, Kun Zhang, and Joris M. Mooij (2012). «On causal and anti-causal learning.» In: *Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012*. icml.cc / Omnipress.
- Schölkopf, Bernhard, Francesco Locatello, Stefan Bauer, Nan Rosemary Ke, Nal Kalchbrenner, Anirudh Goyal, and Yoshua Bengio (2021). «Toward causal representation learning.» In: *Proceedings of the IEEE* 109.5, pp. 612–634.
- Schulman, John, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov (2017). «Proximal policy optimization algorithms.» In: *arXiv preprint arXiv:1707.06347*.

- Shazeer, Noam, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc V. Le, Geoffrey E. Hinton, and Jeff Dean (2017). «Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer.» In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.
- Shimodaira, Hidetoshi (2000). «Improving predictive inference under covariate shift by weighting the log-likelihood function.» In: *Journal of statistical planning and inference* 90.2.
- Siegelmann, Hava T and Eduardo D Sontag (1995). «On the computational power of neural nets.» In: *Journal of computer and system sciences* 50.1.
- 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. (2016). «Mastering the game of Go with deep neural networks and tree search.» In: *nature* 529.7587.
- Silver, David, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharsan Kumaran, Thore Graepel, et al. (2017). «Mastering chess and shogi by self-play with a general reinforcement learning algorithm.» In: *arXiv preprint arXiv:1712.01815*.
- (2018). «A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play.» In: *Science* 362.6419, pp. 1140–1144.
- Simon, H. A. (1953). «Causal Ordering and Identifiability.» In: *Studies in Econometric Methods*. Cowles Commission for Research in Economics, Monograph No. 14. John Wiley & Sons.
- Singh, Sidak Pal and Dan Alistarh (2020). «WoodFisher: Efficient second-order approximations for model compression.» In: *arXiv preprint arXiv:2004.14340*.
- Snelson, Edward and Zoubin Ghahramani (2005). «Sparse Gaussian Processes using Pseudo-inputs.» In: *Advances in Neural*

- Information Processing Systems 18 [Neural Information Processing Systems, NIPS 2005, December 5-8, 2005, Vancouver, British Columbia, Canada].*
- Stein, Gregory J, Christopher Bradley, and Nicholas Roy (2018). «Learning over Subgoals for Efficient Navigation of Structured, Unknown Environments.» In: *Conference on Robot Learning*.
- Subbaswamy, Adarsh, Peter Schulam, and Suchi Saria (2019). «Preventing Failures Due to Dataset Shift: Learning Predictive Models That Transport.» In: *The 22nd International Conference on Artificial Intelligence and Statistics, AISTATS 2019, 16-18 April 2019, Naha, Okinawa, Japan*. Vol. 89. Proceedings of Machine Learning Research. PMLR.
- Sugiyama, Masashi and Motoaki Kawanabe (2012). *Machine learning in non-stationary environments: Introduction to covariate shift adaptation*. MIT press.
- Sugiyama, Masashi, Matthias Krauledat, and Klaus-Robert Müller (2007). «Covariate shift adaptation by importance weighted cross validation.» In: *Journal of Machine Learning Research* 8.May.
- Sutton, Richard S, Doina Precup, and Satinder Singh (1999). «Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning.» In: *Artificial intelligence* 112.1-2.
- Sutton, Richard (2019). «The Bitter Lesson.» In: URL: <http://incompleteideas.net/IncIdeas/BitterLesson.html>.
- Titsias, Michalis (2009). «Variational learning of inducing variables in sparse Gaussian processes.» In: *Artificial intelligence and statistics*. PMLR.
- Todorov, Emanuel, Tom Erez, and Yuval Tassa (2012). «Mujoco: A physics engine for model-based control.» In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE.
- Tzeng, Eric, Judy Hoffman, Kate Saenko, and Trevor Darrell (2017). «Adversarial Discriminative Domain Adaptation.» In: *2017*

- IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*. IEEE Computer Society.
- Udrescu, Silviu-Marian and Max Tegmark (2020). «AI Feynman: A physics-inspired method for symbolic regression.» In: *Science Advances* 6.16.
- Udrescu, Silviu-Marian, Andrew Tan, Jiahai Feng, Orisvaldo Neto, Tailin Wu, and Max Tegmark (2020). «AI Feynman 2.0: Pareto-optimal symbolic regression exploiting graph modularity.» In: *arXiv preprint arXiv:2006.10782*.
- Vapnik, Vladimir N (1999). «An overview of statistical learning theory.» In: *IEEE transactions on neural networks* 10.5.
- Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin (2017). «Attention is All you Need.» In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*.
- Vezhnevets, Alexander Sasha, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu (2017). «FeUdal Networks for Hierarchical Reinforcement Learning.» In: *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*. Vol. 70. Proceedings of Machine Learning Research. PMLR.
- Wang, Linnan, Rodrigo Fonseca, and Yuandong Tian (2020). «Learning Search Space Partition for Black-box Optimization using Monte Carlo Tree Search.» In: *NeurIPS*.
- Williams, R. J. (2004). «Simple statistical gradient-following algorithms for connectionist reinforcement learning.» In: *Machine Learning* 8.

- Zhai, Xiaohua, Alexander Kolesnikov, Neil Houlsby, and Lucas Beyer (2021). «Scaling vision transformers.» In: *arXiv preprint arXiv:2106.04560*.
- Zhang, Amy, Sainbayer Sukhbaatar, Adam Lerer, Arthur Szlam, and Rob Fergus (2018). «Composable Planning with Attributes.» In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. Vol. 80. Proceedings of Machine Learning Research. PMLR.
- Zhang, Chiyuan, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals (2017). «Understanding deep learning requires rethinking generalization.» In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.
- Zhang, Guodong, Lala Li, Zachary Nado, James Martens, Sushant Sachdeva, George E. Dahl, Christopher J. Shallue, and Roger B. Grosse (2019a). «Which Algorithmic Choices Matter at Which Batch Sizes? Insights From a Noisy Quadratic Model.» In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*.
- Zhang, K., J. Zhang, and B. Schölkopf (2015). «Distinguishing Cause from Effect Based on Exogeneity.» In: *Fifteenth Conference on Theoretical Aspects of Rationality and Knowledge*.
- Zhang, Yunbo, Wenhao Yu, and Greg Turk (2019b). «Learning Novel Policies For Tasks.» In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. Vol. 97. Proceedings of Machine Learning Research. PMLR.

Giambattista Parascandolo

Sept 2021

giambattista.parascandolo@tue.mpg.de | sites.google.com/view/giambattista-parascandolo/

EDUCATION

- PhD – Max Planck Institute for Intelligent Systems & ETH Zurich** Tübingen & Zurich
PhD in Computer Science *Mar 2017 → Sep 2021*
- Topic: *Deep Learning Beyond the Training Distribution*
 - PhD defense committee: Prof. Bernhard Schölkopf and Prof. Thomas Hofmann (advisors), Prof. Yoshua Bengio
 - *European Laboratory for Learning and Intelligent Systems (ELLIS) PhD Student*
 - Doctoral fellow of the *MPI-ETH Center for Learning Systems*
 - Seminars: *Machine Learning; Deep Reinforcement Learning; Robotics, Systems and Controls;*
- Center for Brains Minds and Machines, Virtual Summer Course** Cambridge, MA
Massachusetts Institute of Technology (MIT) *Attended remotely due to CoV-2 pandemic* *June 2020*
- MSc – Tampere University of Technology** Tampere, Finland
Master's degree in Information Technology *2014 – 2015*
- *GPA – 5.0/5.0*, graduated with distinction (highest)
 - Major: Signal processing. Minor: Learning and intelligent systems
- BSc – Università di Roma Tor Vergata** Rome, Italy
Bachelor's degree dpt. of Mathematics. Media and Technology Sciences *2010 – 2013*
- *Grade – 110/110*, summa cum laude
 - Major: Mathematics

EXPERIENCE

- OpenAI** San Francisco, California
Research Scientist *October 2021 →*
- Reinforcement Learning team with John Schulman.
- DeepMind** London, UK
Research Scientist Intern *May 2019 – October 2019*
- Worked with Lars Buesing on goal-directed RL via Divide-and-Conquer MCTS.
- Google X, the Moonshot Factory** Mountain View, CA
Machine Learning Resident *June 2018 – Sept 2018*
- Machine Learning expert on a project exploring automated design for electromagnetic devices, with applications ranging from integrated photonics to metasurfaces, designed on ultra-large-scale simulators.
- Teacher Assistant**
- *Causal Representation Learning* – ETH Zürich *2020 - 2021*
 - *Deep Learning* – ETH Zürich *2020 - 2021*
 - *Computational Intelligence Lab* – ETH Zürich *2019 - 2020*
 - *Analysis of Audio, Speech and Music Signals* – Tampere University of Technology *2014 - 2015*
 - *Mathematical Methods in Computer Graphics* – University of Rome Tor Vergata *2013 - 2014*
- M.Sc. thesis advisor**
- *Topographic Neural Networks*, Federico Pirovano *2020 - 2021*
 - *Acoustic scene classification with deep neural networks*, Michele Valenti *2015 - 2016*
- Project Researcher, Research Assistant** Tampere, Finland
Audio Research Group – Tampere University of Technology *Feb 2015 – March 2017*
- Deep learning and applications to audio processing. Funded by Google Faculty Research Award to Prof. Virtanen.

OTHER

Academic work: Reviewer for NeurIPS, ICML, ICLR. PhD admission interviewer for MPI & ETH.
Languages: English, Italian, Python – *fluent* German, French, Matlab, C++ – *basic*

INVITED TALKS AND LECTURES

- MIT – Seminar talk to the *College of Computing* and *Brain and Cognitive Sciences* departments *March 2021*
- University of Rome **La Sapienza** – Invited Lecture for graduate course on Deep Learning *April 2021*
- Politecnico di Milano – Invited Talk *April 2021*
- MILA | Montreal Institute for Learning Algorithms *Nov 2020*
- Continual AI *Oct 2020*
- Google X *Dec 2018*

SELECTED PUBLICATIONS AND PREPRINTS — [GOOGLE SCHOLAR](#) FOR UP-TO-DATE LIST

Research impact: > 1500 citations, h-index 13.

* denotes equal contribution

Topographic Neural Networks

Federico Pirovano, *GP*
preprint

Divide-and-Conquer Monte Carlo Tree Search

*GP**, L. Buesing*, J. Merel, L. Hasenclever, A. Neitz, T. Weber, J. Hamrick, J. Aslanides, N. Heess
preprint – work done while at DeepMind

Neural Symbolic Regression that Scales

L. Biggio, T. Bendinelli, A. Neitz, A. Lucchi, *GP*
[ICML 2021](#)

NeurIPS 2020 Workshop on “Learning Meets Combinatorial Algorithms”

NeurIPS 2020 Workshop on “Knowledge Representation & Reasoning Meets Machine Learning”

Learning explanations that are hard to vary

*GP**, A. Neitz*, A. Orvieto, L. Gresele, B. Schölkopf
[ICLR 2021](#)

A teacher-student framework to distill future trajectories

A. Neitz*, *GP**, B. Schölkopf
[ICLR 2021](#)

Adaptive Skip Intervals: Temporal Abstraction for Recurrent Dynamical Models

A. Neitz, *GP*, S. Bauer, B. Schölkopf
[NeurIPS 2018](#)

Learning Independent Causal Mechanisms

*GP**, N. Kilbertus, M. Rojas-Carulla, B. Schölkopf
[ICML 2018](#)

NeurIPS 2017 Workshop on Learning Disentangled Features: from Perception to Control – *Spotlight*

Generalization in anti-causal learning

N. Kilbertus*, *GP**, B. Schölkopf*

NeurIPS 2018 Workshop on Critiquing and correcting trends in machine learning

Tempered Adversarial Networks

M. S. M. Sajjadi, *GP*, A. Mehrjou, B. Schölkopf

[ICML 2018](#)

ICLR 2017 Workshop

Avoiding Discrimination through Causal Reasoning

N. Kilbertus, M. Rojas-Carulla, *GP*, M. Hardt, D. Janzing, B. Schölkopf

[NeurIPS 2017](#)

ConvWave: Searching for Gravitational Waves with Fully Convolutional Neural Nets

T. Gebhard, N. Kilbertus, *GP*, I. Harry, B. Schölkopf

NeurIPS 2017 Workshop on Deep Learning for Physical Sciences

Convolutional recurrent neural networks for polyphonic sound event detection

E. Cakir*, *GP**, T. Heittola, H. Huttunen, T. Virtanen

[IEEE Transaction on Audio, Speech and Language Processing, Special issue on Sound Scene and Event Analysis – journal](#)

Recurrent neural networks for polyphonic sound event detection in real life recordings

GP, H. Huttunen, T. Virtanen

[ICASSP 2016 – Oral – Award for the best student paper of the Special Session](#)

Taming the waves: sine as activation function in deep neural networks

GP, H. Huttunen, T. Virtanen

preprint - 2016

Sound Event Detection in Multichannel Audio Using Spatial and Harmonic Features

S. Adavanne, *GP*, P. Pertilä, T. Heittola, T. Virtanen

Workshop on Detection and Classification of Acoustic Scenes and Events 2016 – *1st place in the challenge*

Acoustic Scene Classification Using Convolutional Neural Networks

M. Valenti, A. Diment, *GP*, S. Squartini, T. Virtanen

[IJCNN 2017 – Best Student Poster Intel Award](#)

Workshop on Detection and Classification of Acoustic Scenes and Events 2016 – *Oral*

Low-latency sound source separation using deep neural networks

G. Naithani, *GP*, T. Barker, N. H. Pontoppidan, T. Virtanen

[IEEE Global Conference on Signal and Information Processing \(GlobalSIP\) 2016](#)