On Knowledge Transfer in Deep Learning: The Roles of Disentanglement, Modularity, and Meta-learning

On Knowledge Transfer in Deep Learning: The Roles of Disentanglement, Modularity, and Meta-learning

Dissertation

der Mathematisch-Naturwissenschaftlichen Fakultät

der Eberhard Karls Universität Tübingen

zur Erlangung des Grades eines

Doktors der Naturwissenschaften

(Dr. rer. nat.)

vorgelegt von
M.Sc. Muhammad Waleed Gondal
aus Multan/ Pakistan

Tübingen
2025

Gedruckt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Eberhard Karls Universität Tübingen.

Tag der mündlichen Qualifikation:	23.09.2025
Dekan:	Prof. Dr. Thilo Stehle
1. Berichterstatter:	Prof. Dr. Bernhard Schölkopf
2. Berichterstatter:	Prof. Dr. Gerard Pons-Moll

To my family

Abstract

The remarkable capacity of human brains to seamlessly adapt to new situations by using prior knowledge, and applying learned concepts across a diverse range of contexts demonstrates its exceptional prowess in knowledge transfer capabilities. This hallmark of human intelligence has inspired the development of artificial neural networks that seek to emulate human-like learning and knowledge transfer capabilities. Although deep neural networks (DNNs) have achieved remarkable performance across a wide range of tasks, they often struggle to generalize and transfer knowledge across diverse data distributions and tasks. Inductive biases, inspired by cognitive mechanisms and underlying structure within data are believed to be crucial for enhancing such generalization capabilities. This dissertation investigates the role of three specific inductive biases - disentanglement, modularity, and meta-learning in facilitating knowledge transfer in DNNs.

First, we examine disentanglement, exploring how learning structured representations of data by isolating the underlying factors of variation enable improved generalization and sample efficient transfer learning. We investigate disentangled representation learning for sequential data and images. For sequential data, we introduce a new type of state space model, Disentangled State Space Models (DSSM), which explicitly separates domain-invariant state dynamics from domain-specific information. Using simulated data, we investigate how disentangled representations of sequences improve knowledge transfer, sequence manipulation, and domain characterization. For image-based tasks, we introduce the first real-world disentanglement dataset, MPI3D, captured in a controlled setting with known ground-truth factors. We also provide two more datasets simulating the experimental setup. We subsequently, benchmark the performance of unsupervised disentanglement algorithms on the new datasets and systematically investigate how simulated datasets can be used to construct improved representations of real-world data. Next, we delve into modularity, which entails developing reusable, independent neural network components inspired by independent causal mechanisms. We introduce Neural Interpreters (NI), a novel attention-based architecture designed for compositional reasoning and knowledge reuse. We demonstrate its effectiveness in image classification and abstract reasoning tasks, showcasing improved sample-efficient transfer learning and systematic generalization capabilities. Lastly, we explore the inductive bias of meta-learning and contrastive learning, aiming to learn transferable meta-representations of data-generating functions. We propose Function Contrastive Representation Learning (FCRL), a method that leverages contrastive learning to meta-learn function representation independently from downstream prediction tasks. The learned representations are then used to solve a range of downstream tasks defined for a given function.

Throughout the thesis, we conduct comprehensive experimental evaluations on simulated and real-world datasets, demonstrating the effectiveness of the proposed inductive biases in promoting knowledge transfer and generalization in DNNs. We also discuss the challenges and limitations of incorporating inductive biases and suggest future research directions to further advance the capabilities of deep learning models for better knowledge transfer and out-of-distribution generalization.

Kurzfassung

Die bemerkenswerte Fähigkeit des menschlichen Gehirns, sich nahtlos an neue Situationen anzupassen, indem es auf vorhandenes Wissen zurückgreift und gelernte Konzepte in einer Vielzahl von Kontexten anwendet, demonstriert seine außergewöhnliche Stärke in der Fähigkeit zum Wissenstransfer. Diese Eigenschaft der menschlichen Intelligenz hat die Entwicklung künstlicher neuronaler Netzwerke inspiriert, die versuchen, menschenähnliche Lern- und Wissenstransferfähigkeiten zu emulieren. Obwohl tiefe neuronale Netzwerke (DNNs) beeindruckende Leistungen bei verschiedenen Aufgaben gezeigt haben, kämpfen sie oft damit, Wissen über verschiedene Datendistributionen hinweg zu generalisieren und zu übertragen. Induktive Verzerrungen, inspiriert von kognitiven Mechanismen und der zugrundeliegenden Struktur innerhalb der Daten, gelten als entscheidend für die Verbesserung solcher Generalisierungsfähigkeiten. Diese Dissertation untersucht die Rolle von drei spezifischen induktiven Verzerrungen - Entflechtung, Modularität und Meta-Lernen - bei der Erleichterung von Wissenstransfer und Generalisierung in DNNs.

Zuerst untersuchen wir die Entflechtung, indem wir erforschen, wie das Lernen strukturierter Datenrepräsentationen durch Isolierung der zugrundeliegenden Variationsfaktoren verbesserte Generalisierung und effizientes Transferlernen ermöglicht. Wir untersuchen das Lernen entflechteter Repräsentationen für sequenzielle Daten und Bilder. Für sequenzielle Daten führen wir eine neue Klasse von Zustandsraummodellen ein, die Entflechteten Zustandsraummodelle (DSSMs), die explizit domänenunabhängige Zustandsdynamiken von domänenspezifischen Informationen trennen. Mit simulierten Daten untersuchen wir, wie entflechtete Repräsentationen von Sequenzen den Wissenstransfer, die Sequenzmanipulation und die Domänencharakterisierung verbessern. Für bildbasierte Aufgaben führen wir den ersten realweltlichen Entflechtungsdatensatz, MPI3D, ein, der in einer kontrollierten Umgebung mit bekannten Grundwahrheitsfaktoren aufgenommen wurde. Wir stellen außerdem zwei weitere Datensätze vor, die das experimentelle Setup simulieren. Anschließend bewerten wir die Leistung von Algorithmen zum unüberwachten Entflechten anhand der neuen Datensätze und untersuchen systematisch, wie simulierte Datensätze genutzt werden können, um bessere Repräsentationen der realen Welt zu erstellen.

Als Nächstes befassen wir uns mit der Modularität, die die Entwicklung wiederverwendbarer, unabhängiger Komponenten neuronaler Netzwerke umfasst, inspiriert von unabhängigen kausalen Mechanismen. Wir stellen Neural Interpreters (NI) vor, eine neuartige aufmerksamkeitsbasierte Architektur, die für kompositionelles Denken und Wiederverwendung von Wissen konzipiert ist. Wir demonstrieren ihre Wirksamkeit in

Aufgaben der Bildklassifikation und abstrakten Denkaufgaben und zeigen verbessertes transfer-effizientes Lernen und systematische Generalisierungsfähigkeiten. Zuletzt erkunden wir die induktive Verzerrung des Meta-Lernens und des kontrastiven Lernens mit dem Ziel, übertragbare Meta-Repräsentationen von datengenerierenden Funktionen zu lernen. Wir schlagen das Function Contrastive Representation Learning (FCRL) vor, eine Methode, die kontrastives Lernen nutzt, um unabhängig von nachgelagerten Vorhersageaufgaben Funktionenrepräsentationen durch Meta-Lernen zu erlernen. Die gelernten Repräsentationen werden dann verwendet, um eine Vielzahl von nachgelagerten Aufgaben zu lösen, die auf einer gegebenen Funktion basieren.

Im Verlauf der Arbeit führen wir umfassende experimentelle Bewertungen an simulierten und realweltlichen Datensätzen durch, die die Wirksamkeit der vorgeschlagenen induktiven Verzerrungen bei der Förderung des Wissenstransfers und der Generalisierung in DNNs demonstrieren. Wir diskutieren auch die

Herausforderungen und Grenzen der Einbeziehung induktiver Verzerrungen und schlagen zukünftige Forschungsrichtungen vor, um die Fähigkeiten von Deep-Learning-Modellen für einen besseren Wissenstransfer und Generalisierung außerhalb der Verteilung weiterzuentwickeln.

Acknowledgments

I am extremely grateful to Bernhard Schölkopf for providing me with the opportunity to pursue my Ph.D. under his guidance within the exceptional group at the Max Planck Institute. Your trust and advice have been very important to my growth, providing me with the freedom to follow my research interests. Working under your guidance, has been both an honor and an exciting experience.

I am deeply thankful to Gerard Pons-Moll, for agreeing to review this thesis and for serving on my examination committee. I am also grateful to Seong Joon Oh and Maria Knobelsdorf for their willingness to be in my thesis defense committee.

Special thanks to Michael Hirsch, a mentor whose faith and support enabled me to be where I am today; to Luitpold Staudigl, for giving me exposure to industry, for teaching me to think about applying machine learning in scalable ways that can impact and improve the experiences of millions of people, and for his constant support during crucial years of my Ph.D.; and to Stefan Bauer, for his constant guidance and support throughout this journey.

A big thanks to Sabrina Rehbaum for your kindness and unwavering support through the daily administrative challenges over the years.

I would like to thank Nasim Rahaman for the countless hours of collaboration, brainstorming, coding, and for teaching me new concepts. Our discussions around MPI whiteboards, followed by impromptu pizza parties, have been some of the best times. I am grateful for your friendship and for being such a great collaborator.

To Manuel Wüthrich, I deeply appreciate your invaluable assistance with projects, papers, and finding an apartment. To Annika Buchholz, for your support and fun discussions. I am also thankful to Krikamol Muandet, Wittawat Jitkrittum, and Antonio Vergari for broadening my scientific horizons and helping me in this journey. I would also like to thank all the amazing colleagues at the Empirical Inference group for the interesting discussions and the fun we had: thank you Alexander Neitz, Anant Raj, Julius von Kügelgen, Timothy Gebhard, Luigi Gresele, Diego Agudelo-España, Frederik Träuble, Simon Buchholz, Amir Karimi, Shruti Joshi, Arash Mehrjou, Giambattista Parascandolo, Francesco Locatello, Yassine Nemmour, Maximilian Dax and Jonas Kübler. A big thanks to everyone else in the department for teaching me the ways of research, and inspiring me to improve every day.

To Hamd ul Moqeet and Rohma Saeed, you guys made my time in Tübingen unforgettable. Our weekly board game nights, the great food, and those deep dives into Coke Studio discussions were the highlights of my week.

To my long-time friend Talha Badr, who has stood by me through thick and thin, thank

you for your unwavering loyalty and steady presence over the years. And to Waleed Ahmed, thank you for always opening your door to me, your place was a refuge where I could unwind and laugh.

To my parents and siblings, whose constant love and quiet sacrifices have shaped who I am, and to whom I owe more than words can express for the opportunities they created for me, for the standards they instilled in me, and for the steady guidance that has carried me through every stage of my life.

Lastly, to my loving wife Aima, whose support, belief, and encouragement have been pivotal in realizing this endeavor. I am deeply thankful for your sacrifices and excited for the future adventures we will share together.

Contents

Symbols

x	Scalar data point
$\mathbf{x}$	Vector data point for function inputs
$\mathbf{y}$	Vector data point for function outputs
$\mathbf{X}$	Random variable $\mathbf{X}$
$\bar{\mathbf{X}}$	Sequence of non-i.i.d. random variables, i.e., $(\mathbf{X}_1, \mathbf{X}_2, ..., \mathbf{X}_T)$
$p(x)$	Probability density function
$p(y\|x)$	Conditional probability density function of Y given $X = x$, evaluated at y
i.i.d.	Independent and identically distributed
$\mathcal{N}(\mu, \Sigma)$	Gaussian distribution with mean μ and covariance Σ
$\mathbb{E}$	Expectation operator
$\mathcal{L}$	Evidence lower bound (ELBO)
$\mathcal{F}$	Set of functions in the Neural Interpreter
σ	Learnable parameter in the type matching mechanism
$\mathbf{W}$	Weight matrix
$\mathbf{b}$	Bias vector
$sim(\mathbf{x}, \mathbf{y})$	Cosine similarity measure between vectors $\mathbf{x}$ and $\mathbf{y}$

Chapter 1

Introduction

We live in the world of perpetual novelty where we come across a variety of new experiences on daily basis. Our brains excel at identifying familiar patterns hidden within these variations, enabling us to rapidly adapt our prior knowledge and skills to tackle new tasks. This knowledge and skill adaptation occurs at multiple levels of abstractions. For instance, a person who is proficient in riding a bicycle on flat roads can quickly learn to navigate rough terrains on a mountain bike. Likewise, this individual has an advantage when learning to ride a motorcycle. The ability to construct knowledge from one task and transfer it effectively to solve novel tasks in unseen domains is a hallmark of human intelligence. In cognitive science, this transfer is often attributed to cognitive mechanisms such as knowledge compilation, error correction, and analogical transfer (Nokes, 2009; Gick and Holyoak, 1987). Although a comprehensive theory of transfer is still elusive, these mechanisms are believed to be intricately linked with human learning, memory, categorization, reasoning, and problem-solving skills (Koedinger and Roll, 2012).

Inspired by the human ability to learn and transfer knowledge across experiences, researchers have endeavored to create artificial agents that can replicate this ability. This pursuit has led to the development of artificial neural networks (ANNs) (Rosenblatt, 1958), computational models inspired by the biological neural networks in the human brain. These networks consist of interconnected artificial neurons that emulate the structure and function of biological neurons. Like in brain, activations across these neurons determine how signals propagate and interact to process information and learn from data. In more complex models, known as deep neural networks (DNNs) (LeCun *et al.*, 2015; Goodfellow *et al.*, 2016), numerous layers of neurons are used to extract hierarchical and abstract representations of the data. This enables DNNs to capture intricate patterns and dependencies, distributing knowledge across multiple levels of abstraction. Consequently, DNNs have demonstrated impressive results on a wide range of tasks that includes computer vision (Krizhevsky *et al.*, 2012; Radford *et al.*, 2021), drug discovery (Gawehn *et al.*, 2016), protein unfolding (Jumper *et al.*, 2021), natural language processing (Young *et al.*, 2018; Brown *et al.*, 2020; Chowdhery *et al.*, 2022), speech recognition (Graves *et al.*, 2013), robotics (Sünderhauf *et al.*, 2018; Agarwal *et al.*, 2023), and reinforcement learning (Mnih *et al.*, 2015; Silver *et al.*, 2016) among others.

While DNNs can achieve remarkable performance on complex tasks when trained

with copious amounts of data, their performance is fundamentally constrained by the assumption that the evaluation samples come from the same distribution as the training examples, a condition often referred to as i.i.d. generalization (Vapnik, 1999; Bishop *et al.*, 2001). In practice, this assumption is frequently violated when models are applied to real-world tasks involving diverse data distributions. Consequently, models may lack robustness to distribution shifts and struggle to generalize effectively when tackling tasks involving previously unseen data distributions (Peters *et al.*, 2017; Geirhos *et al.*, 2020; Hendrycks and Dietterich, 2019; Barbu *et al.*, 2019). Recent advances in large-scale pre-training have shown promise to mitigate this issue (Bommasani *et al.*, 2021; Brown *et al.*, 2020), but there is a growing consensus that attaining human-like transfer learning and generalization across distributions (out-of-distribution generalization) necessitates the incorporation of inductive biases that are more closely aligned with our comprehension of the world and human cognition (Schölkopf *et al.*, 2021; Bengio *et al.*, 2013; Goyal and Bengio, 2022). These inductive biases, often inspired by cognitive mechanisms, encapsulate our priors or assumptions about tasks and guide the learning process. The incorporation of these biases can potentially help deep models to better understand the underlying data structures and adapt more effectively to new, unseen tasks. To develop meaningful high-level inductive biases that enable knowledge transfer across tasks, we must consider the assumptions regarding the types of data distributions a learning algorithm will encounter in real-world situations. For instance, if a DNN is proficient in a set of tasks (e.g., recognizing various animals), it should be able to generalize its knowledge and perform well on a new, related task (e.g., identifying a specific breed of an animal). This generalization is only possible if there is a shared structure between the training tasks and the new task. Therefore, it is essential to ask:

- *How do source and target distributions differ, and what kind of structure, if any, is shared across tasks that can be leveraged for effective knowledge transfer?*
- *Can DNNs efficiently learn this structure through appropriate inductive biases?*
- *What are the limitations of current state-of-the-art models in terms of modularizing knowledge for reuse, and meta-learning representations that are both transferable and robust to noise?*

Addressing these questions is critical because, without a mechanism to capture the latent structure underlying diverse tasks, a model's capacity to generalize is inherently limited. Causality (Peters *et al.*, 2016) provides a framework with the necessary assumptions for representing structural knowledge about the data-generating process, which aligns closely with structural causal models (Pearl, 2009).

In this dissertation, we posit that a natural structure underlies some data distribution and that by integrating this structure into DNNs through suitable inductive biases, we can facilitate knowledge transfer across tasks. This knowledge transfer may range from extreme zero-shot out-of-distribution generalization to sample-efficient transfer learning

with a few labeled examples. More specifically, we examine the role of three inductive biases: disentanglement, modularity, and meta-learning. We not only introduce novel methods for efficiently implementing these inductive biases in DNNs but also assess the knowledge transfer capabilities they bestow in simulated and real-world scenarios. For the above mentioned inductive biases, the knowledge of the natural structure within data distributions comes either from prior beliefs and assumptions (in the case of modularity and disentanglement, inspired by independent causal mechanisms (Schölkopf *et al.*, 2021)) or from the data itself (in the case of meta-learning).

1.1 Inductive Biases

Inductive biases refer to the set of assumptions or constraints embedded within a learning algorithm that guide it to prioritize certain hypotheses or solutions over others. These biases are essential for enabling algorithms to generalize beyond the training distribution (Wolpert and Macready, 1997). In deep learning, these biases are integrated in various ways, such as through novel architectures (e.g., self-attention in transformers), specialized regularization methods, or carefully designed objective functions. Such approaches have been critical in addressing challenges associated with both in-distribution and out-of-distribution generalization (Goyal and Bengio, 2020; Schölkopf *et al.*, 2021; Bengio *et al.*, 2013). In this dissertation, we primarily focus on three key inductive biases: disentanglement, modularity, and meta-learning. Each of these biases offers a distinct perspective on how knowledge can be factorized, organized, and efficiently transferred across different tasks and domains.

Disentanglement. Disentangled representation learning aims to encode data into latent variables that correspond to independent factors of variation (Bengio *et al.*, 2013; Kim and Mnih, 2018; Locatello *et al.*, 2018). In theory, such factorized representations should make it easier to generalize across tasks where a subset of these factors change in an unseen manner e.g., novel environments or interventions. However, contemporary disentanglement methods heavily rely on synthetic benchmarks (e.g., dSprites (Higgins *et al.*, 2017a), Shapes3D (Kim and Mnih, 2018)), where ground-truth factors are explicitly defined and controlled. Real-world data, by contrast, rarely permits such clean separations i.e., factors are often confounded e.g., correlated lighting and object pose in natural images, thus making evaluation and algorithm development challenging. To bridge this gap, we introduce the first real-world disentanglement dataset that combines the complexity of real-world data (e.g., noise, occlusions, and natural correlations) with controlled, known factors of variation. The novel dataset facilitates rigorous evaluation of disentanglement algorithms in near-real-world settings while preserving the ground-truth annotations necessary for benchmarking.

Modularity. Closely related to disentanglement, modularity is an inductive bias that emphasizes compositional factorization of knowledge into reusable components or modules. Drawing from the insights of independent causal mechanisms (Parascandolo *et al.*, 2017; Peters *et al.*, 2017; Besserve *et al.*, 2020; Goyal *et al.*, 2019), modularity posits that complex tasks can be decomposed into smaller and reusable components or modules. These modules enable systematic generalization by recombining learned components in novel configurations i.e., models can adapt to unseen tasks without retraining (Lake and Baroni, 2018; Keysers *et al.*, 2019). However, traditional deep neural networks often learn monolithic representations (Krizhevsky *et al.*, 2012; Andreas *et al.*, 2016), where entangled parameters hinder flexible reuse of knowledge (e.g., transferring "object detection" skills to a new domain requires retraining the entire model). To address this, we propose a modular self-attention architecture that instantiates explicit, hierarchical modules at multiple abstraction levels. Unlike prior work, the proposed architecture allows dynamic, context-dependent recombination of modules, enabling efficient adaptation to novel tasks while preserving task-specific expertise.

Meta-learning. While disentanglement and modularity focus on structural and compositional factorization of knowledge, meta-learning (also referred as "learning to learn") targets the fast adaptation of learned representations to new tasks (Sohn, 2016; Finn *et al.*, 2017; Vinyals *et al.*, 2016; Ravi and Larochelle, 2016). These methods leverage task-distribution priors (e.g., shared dynamics in robotics or common visual concepts in few-shot image classification) to learn meta-representations that encode the shared structure. Learning such representations facilitate sample-efficient adaptation i.e., solving novel tasks with minimal data by reusing the shared representations (Garnelo *et al.*, 2018a,b). However, a key challenge arises from the joint optimization of meta-representations and task-specific predictors which can lead to learning task-specific meta-representations, limiting their transferability to out-of-distribution tasks. To address this, we propose a decoupled training framework that separates meta-representation learning from task adaptation. By leveraging contrastive learning, we enforce invariance to task-irrelevant variations (e.g., domain shifts) within meta-representation, resulting in more robust and generalizable meta-representations.

1.2 Outline and contributions

This dissertation is composed of four primary chapters, with each chapter examining the role of a specific inductive bias in knowledge transfer across tasks or domains. The final chapter provides conclusions and offers insights into potential directions for future research. Each chapter includes the necessary technical background, related work, algorithms, theory, and experiments for a comprehensive understanding. The dissertation is organized as follows.

In Chapter 2, we assume that sequential data is generated from diverse and heterogeneous domains. We define a domain as an environment with invariant properties within a sequence but changing across different sequences. To learn representations for sequences across different domains, we propose Disentangled State Space Models (DSSM) - a new class of state space models that explicitly separate domain-invariant state dynamics from the domain-specific factors that influence them (Miladinović *et al.*, 2019b,a). The inductive bias of disentanglement is enforced through a novel Variational Autoencoder (VAE) based training procedure that leverages temporal coherence and domain stationarity to regulate the learning process. We utilize simulated data to examine how these structured sequence representations enhance knowledge transfer across domains, support robust predictions, enable sequence manipulation, and facilitate domain characterization while incorporating our assumptions about task structure.

In Chapter 3, we examine the transfer of disentangled representations in static images from simulations to the real world, where domains represent different levels of realism in the images (simulated toy, simulated realistic, and real-world images) (Gondal *et al.*, 2019a). Here, we make an assumption that the underlying independent factors of variation can be disentangled. The premise of this chapter is that studying disentanglement in real-world images is challenging due to expensive data collection, imperfections, and uncontrollable factors. Conversely, simulated datasets are inexpensive, straightforward to produce, and provide precise control over independent generative factors. As a result, the development of disentanglement models have largely depended on simulated toy datasets. To thoroughly evaluate these models' performance on real-world data, we introduce MPI3D, the first real-world disentanglement dataset captured in a controlled setting with known independent ground-truth factors. We developed an experimental platform using a robotic arm to manipulate objects and created two additional simulation datasets. These datasets facilitate a systematic investigation of disentanglement methods' performance on real versus simulated data and generate insights on leveraging simulated data for improved real-world representations. Our study indicates poor model transfer, but effective model and hyperparameter selection for transferring information to the real-world disentanglement.

In Chapter 4, we introduce Neural Interpreters (NIs) (Rahaman *et al.*, 2021), a novel attention-based architecture inspired by typed programming languages. Our proposed architecture decomposes the inference process in a self-attention network into a collection of modules, which we refer to as functions. By assuming a shared underlying structure between tasks, NIs have been designed to enable compositional reasoning and knowledge reuse, thus improving systematic generalization to unseen but related data distributions. Additionally, the architecture's modular inductive bias allows for non-disruptive addition and fine-tuning of functions when faced with new tasks. We analyze the performance of Neural Interpreters in three settings: multi-task image classification, learning fuzzy boolean expressions and abstract reasoning. Our results demonstrate sample-efficient adaptation to new tasks, systematic generalization, and a graceful trade-off between performance and compute at inference time. Additionally, we provide ab-

lation studies and qualitative visualizations, revealing patterns in input routing through specialized (learned) modules.

In Chapter 5, we study the generalization of meta-representations of data-generating functions concerning their transferability to downstream tasks (Gondal *et al.*, 2021). Meta-representations are fixed-dimensional embeddings of a function, derived from just a few input-output examples. They enable shared learning between functions by assuming a common underlying structure between them (the inductive bias of meta-learning). However, ensuring these representations are both transferable to various downstream tasks and robust against noise is challenging. Our findings indicate that jointly optimizing meta-representations alongside prediction tasks can undermine their transferability. To tackle this problem, we propose a decoupled encoder-decoder training scheme with a novel contrastive learning framework called Function Contrastive Representation Learning (FCRL). By leveraging the inductive bias of contrastive learning, FCRL improves transferability of meta-representations to downstream tasks and their noise robustness while retaining the benefits of shared learning and sample efficiency. We conduct extensive experiments on various downstream problems and datasets to assess the effectiveness of our proposed method. Our results reveal that a downstream predictor trained with our pre-trained encoder outperforms relevant methods, including instances where the predictor is trained in conjunction with the encoder.

The thesis expects an understanding of Deep Learning basics, extensively introduced in (Goodfellow *et al.*, 2016). Furthermore, each chapter individually introduces the specific background needed to grasp the technical discussions.

1.3 Related publications

This dissertation is based upon the following publications Miladinović *et al.* (2019a); Gondal *et al.* (2019a, 2021); Rahaman *et al.* (2021):

1. Đorđe Miladinović*, Muhammad Waleed Gondal*, Bernhard Schölkopf, Joachim Buhmann, Stefan Bauer. “Disentangled State Space Models: Unsupervised Learning of Dynamics across Heterogeneous environment”. In: Workshop on Deep Generative Models for Highly Structured Data at ICLR 2019.

2. Muhammad Waleed Gondal, Manuel Wüthrich, Đorđe Miladinović, Francesco Locatello, Martin Breidt, Valentin Volchkov, Joel Akpo, Olivier Bachem, Bernhard Schölkopf, Stefan Bauer. “On the Transfer of Inductive Bias from Simulation to the Real World.” In: Advances in Neural Information Processing Systems (NeurIPS), 2019.

3. Muhammad Waleed Gondal, Shruti Joshi, Nasim Rahaman, Stefan Bauer, Manuel Wüthrich, Bernhard Schölkopf. "Function Contrastive Learning of Transferable Meta-representations." In: International Conference on Machine Learning (ICML), 2021.

4. Nasim Rahaman*, Muhammad Waleed Gondal*, Shruti Joshi, Peter Gehler, Yoshua Bengio, Francesco Locatello, Bernhard Schölkopf. "Dynamic Inference with Neural Interpreters." In: Advances in Neural Information Processing Systems (NeurIPS), 2021.

I also worked on the following publications Rahaman *et al.* (2020); Jitkrittum *et al.* (2019); Gondal *et al.* (2019b) during my Ph.D, however, they are not covered in this dissertation.

1. Muhammad Waleed Gondal, Bernhard Schölkopf, Michael Hirsch. "The Unreasonable Effectiveness of Texture Transfer for Single Image Super Resolution." In: European Conference on Computer Vision (ECCV) 2018 Workshops.

2. Nasim Rahaman, Anirudh Goyal, Muhammad Waleed Gondal, Manuel Wüthrich, Stefan Bauer, Yash Sharma, Yoshua Bengio, Bernhard Schölkopf. "Spatially Structured Recurrent Modules." In: International Conference on Learning Representations (ICLR), 2021.

3. Wittawat Jitkrittum*, Patsorn Sangkloy*, Muhammad Waleed Gondal, Amit Raj, James Hays, Berhard Schölkopf. "Kernel Mean Matching for Content Addressability of GANs." In: International Conference on Machine Learning (ICML), 2019.

Chapter 2

Disentangled State Space Models

"The diversity of the phenomena of nature is so great, and the treasures hidden in the heavens so rich, precisely in order that the human mind shall never be lacking in fresh nourishment."

— Johannes Kepler

Sequential data often originates from diverse environments. Across them exist both shared regularities and environment specifics. To learn robust cross-environment descriptions of sequences, in this chapter, we introduce *disentangled state space models* (DSSM). In the latent space of DSSM, environment-invariant state dynamics is explicitly disentangled from environment-specific information governing that dynamics. We empirically show that such separation enables robust prediction, sequence manipulation and environment characterization. We also propose an unsupervised VAE-based training procedure to learn DSSM as Bayesian filters. In our experiments, we demonstrate that DSSM is able to achieve controlled generation and prediction of synthetic bouncing balls video sequences across varying gravitational influences.

2.1 Introduction

Learning dynamical models from sequential data is a central task in various domains of science (Durbin and Koopman, 2012). This includes managing input of diverse complexity e.g., natural language (Graves, 2013), videos (Srivastava *et al.*, 2015) or financial time-series (Øksendal, 2003). It is also crucial for building interactive agents which use reinforcement and control algorithms on top (Finn and Levine, 2017). Traditional choice in engineering are state space models (SSM) (Koller *et al.*, 2009), typically found in form of Kalman filters (Gelb, 1974) where well-crafted, relatively simple state representations

Based on (Miladinović *et al.*, 2019a). Đorđe Miladinović*, Muhammad Waleed Gondal*, Bernhard Schölkopf, Joachim Buhmann, Stefan Bauer. "Disentangled State Space Models: Unsupervised Learning of Dynamics across Heterogeneous environment". In: Workshop on Deep Generative Models for Highly Structured Data at ICLR 2019.

Figure 2.1: **Sequential systems across environments.** Examples include, from left to right: (left) Michaelis-Menten model for enzyme kinetics, governed by reaction rate constants $\vec{k}$. (center) bouncing ball kinematics, determined by ball weight and playground characteristics. (right) bat swinging motion, influenced by the person performing it. In each example, environments are defined differently, depending on what governs sequence dynamics.

and (normally linear) functional forms are used. To improve flexibility, new solutions rather learn model-free SSM from scratch. Due to their non-autoregressive architecture they make an attractive alternative to recurrent neural networks.

Several recent works have already recognized the benefits of introducing additional structure into SSM: the requirement of separating confounders from actions, observations and rewards (Lu *et al.*, 2018) or content from dynamics (Yingzhen and Mandt, 2018; Fraccaro *et al.*, 2017), especially for transfer learning and extrapolation (Kansky *et al.*, 2017). Complementary to these approaches, we focus on learning structured SSM to decouple system dynamics into its generic (environment-invariant) and environment-specific components. Some examples of sequential data which naturally admit this structure are given in Figure 2.1. Dynamics of these are defined by some constant external factors which we jointly refer to as environment. More concretely, we explore a panel data setting in which we are given multiple sequences describing the same time-evolving phenomena, one or more per environment E. We would like to learn a robust non-parametric SSM to represent the dynamics of that phenomena across these environments, and robustly extrapolate to the unseen ones. To do so, we explicitly model E as a learnable static element of the latent space. Our idea is based on the assumption that one can decouple sequence dynamics to: (i) the generic part which is invariant across environments; and (ii) the environment-specific part. In other words, true E integrates all unobserved environment-specific influences which bias generic system dynamics. Our hypothesis is that considering disentangled, implicitly causal structure of SSM enhances predictive robustness, domain adaptation, and allows for environment characterization and reasoning under interventions e.g., counterfactual inference. The main contributions of this chapter are as follows.

- We introduce a class of non-parametric SSM tailored to exploit invariance from sequential data originating from heterogeneous environments. Disentangled state space models (DSSM) (see Figure 2.2d) form a joint environment model while explicitly decoupling what is generic in sequence dynamics from what is environment-specific.

- We extend on recent advances in amortized variational inference to design an unsupervised training procedure and implement DSSM in form of Bayesian filters. In the spirit of (Karl *et al.*, 2016), well-established reparameterization trick is applied such that the gradient propagates through time. While VAE heuristic provides no convergence guarantees, it is fast, robust and allows end-to-end training.

- We analyze video sequences of a bouncing ball, influenced by varying gravity (environment). We outperform state-of-the-art K-VAE (Fraccaro *et al.*, 2017) in predictions, and also do interventions by "swapping environments" i.e. we enforce a specific dynamic behaviour by using an environment from another sequence which exhibits the desired behaviour. Example videos are available at: `https://sites.google.com/view/dssm`.

2.2 Related work

Closely related to our proposal are approaches which consider structured and disentangled representation of videos, separating the pose from the content (Denton *et al.*, 2017; Tulyakov *et al.*, 2017; Villegas *et al.*, 2017; Yingzhen and Mandt, 2018) or the object appearance from its dynamics (Fraccaro *et al.*, 2017). Proposed models were shown to improve the prediction (Villegas *et al.*, 2017; Denton *et al.*, 2017) and enable controlled generation with "feature swapping" (Tulyakov *et al.*, 2017; Yingzhen and Mandt, 2018). This so called content-based disentanglement was also performed in speech analysis where the structure imposed the explicit separation of sequence- and segment-level attributes (Hsu *et al.*, 2017).

VAE frameworks have already been extended to sequence modeling (Marino *et al.*, 2018), and applied to speech (Bayer and Osendorfer, 2014; Chung *et al.*, 2015; Fraccaro

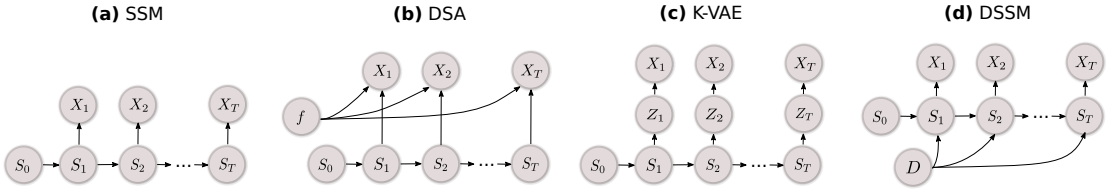


Figure 2.2: **DSSM and related architectures.** **(a)** Traditional SSM architecture was used e.g., in (Karl *et al.*, 2016). **(b)** Disentangled sequential autoencoder (DSA) (Yingzhen and Mandt, 2018) decouples time-invariant *content* from the time-varying features. **(c)** Kalman-VAE (Fraccaro *et al.*, 2017) separates object (content) representation from its dynamics (we did not depict control input here). **(d)** DSSM introduce environments E to model environment-specific effects on sequence *dynamics*.

et al., 2016; Goyal *et al.*, 2017), videos (Yingzhen and Mandt, 2018) and text (Bowman *et al.*, 2015). However, these (mainly) recurrent neural network-based approaches are autoregressive and hence not always suitable e.g., for planning and control from raw pixel space (Watter *et al.*, 2015; Hafner *et al.*, 2018). To "image the world" (Ha and Schmidhuber, 2018) from the latent space directly and circumvent the autoregressive feedback, alternative methods learn SSM instead (Karl *et al.*, 2016; Fraccaro *et al.*, 2017; Krishnan *et al.*, 2017). In DVBF (Karl *et al.*, 2016) SSM is trained using VAE-based learning procedure which allows gradient to propagate through time during training. K-VAE by Fraccaro *et al.* (2017) is a two-layered model which decomposes object's representation from its dynamics. DKF (Krishnan *et al.*, 2015), and very closely related DMM (Krishnan *et al.*, 2017), admit SSM structure but the state inference is conditioned on both past and future observations, so the structure of a filter is not preserved. This is problematic as noted by Karl *et al.* (2016). Similar issues can be found in (Yingzhen and Mandt, 2018).

As opposed to content-based methods which focus on the observation model, our work is focused on dynamics-based disentanglement. This makes our approach complementary to existing strategies (see Figure 2.2). For example, while DSA can represent and manipulate the shape or color of a bouncing ball, our method can manipulate its trajectory. To implement our Bayesian filter, we blended some recent ideas in amortized variational inference (Karl *et al.*, 2016; Marino *et al.*, 2018) and adapted them to fit our novel DSSM architecture.

2.3 Problem setting

We consider a setting in which we are given N sequential observations $X_i^{\,k=1..N}$ of a dynamical system influenced by the underlying environmental factor E. We assume the dynamical system of the following form:

$$X_i = g(S_i) + \omega_i; \quad \omega_i \sim N(0, \Sigma_\omega) \tag{2.1}$$

$$S_{i+1} = f(S_i, E) + \beta_i; \quad \beta_i \sim N(0, \Sigma_\beta) \tag{2.2}$$

where f and g represent non-linear functions. $X_i \in \mathbb{R}^D$ represents the observation in time step i and $S_i \in \mathbb{R}^N$ is the corresponding latent state. Σ_β and Σ_ω are noise covariances which we for simplicity assume are isotropic Gaussian. We further assume that the latent process noise β and observation noise ω are both uncorrelated in time.

Our goal is to jointly learn the generative model which consists of the transition function f and observation function g, together with the corresponding networks ϕ_β^{enc}, ϕ_E^{enc} and ϕ_S^{enc} which infer the process noise residual β_i, the environment E, and the initial state S_0 respectively. The framework overview is given in Figure 2.3.

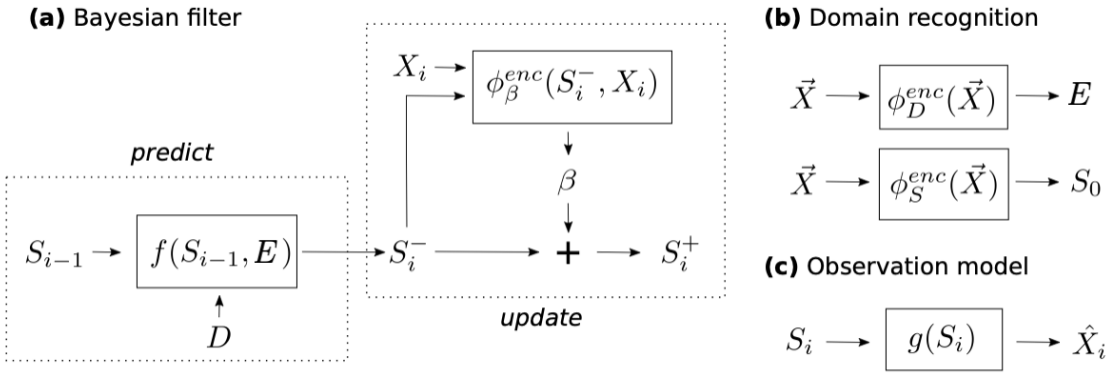


Figure 2.3: **Variational Bayesian filtering framework. (a)** predict-update equations of proposed Bayesian filer: Here, predict step generates the a priori state estimate S_i^-. The update step corrects the initial estimate (S_i^-) using observation X_i through the residual vector β_i. **(b)** in environment recognition phase, the environment E and the initial state S_0 are inferred. **(c)** function g maps the states to observations.

2.4 Variational Bayesian filtering for DSSM

We now describe the full architecture of the DSSM, including the generation and inference models. We also give the training procedure and discuss the optimization challenge.

Generative model. Given an observed sequence $\vec{X}$ of length T, the joint distribution of DSSM factorizes as:

$$p(\vec{X},\vec{S},E,\vec{\beta}) = p_0(S_0)p_0(E)\prod_{i=1}^{T} p(X_i|S_i)p(S_i|S_{i-1},E,\beta_i)p_0(\beta_i) \tag{2.3}$$

This factorization follows the graphical model in Figure 2.2d and the assumption that the process noise is serially uncorrelated. For our experiments, we set the prior probabilities of the initial state $p_0(S_0)$, environment $p_0(E)$ and process noise $p_0(\beta_i) = p_0(\beta)$ to be zero-mean unit-variance Gaussian. Conditioned on β_i and E, state transition is deterministic and the probability $p(S_i|S_{i-1},E,\beta_i)$ is a Dirac function with the peak defined by Equation (2.2). The emission probability $p(X_i|S_i)$ is defined by Equation (2.1).

Inference. Joint variational distribution over the unobserved random variables E, $\vec{S}$ and $\vec{\beta}$, for a sequence of observation $\vec{X}$ of length T factorizes as:

$$q(\vec{S},E,\vec{\beta}|\vec{X}) = q(E|\vec{X})q(S_0|\vec{X})\prod_{i=1}^{T} q(S_i|\beta_i,S_i^-)q(\beta_i|S_i^-,X_i)q(S_i^-|S_{i-1},E) \tag{2.4}$$

Here, the conditionals $S_i^-|S_{i-1},E$ and $S_i|\beta_i,S_i^-$ are deterministic and defined by Equation (2.2). The remaining factors are given as follows:

$$q(\beta_i|S_i^-,x_i) = \mathcal{N}(\mu^{\beta_i},\Sigma^{\beta_i}), \quad [\mu^{\beta_i},\Sigma^{\beta_i}] = \phi_\beta^{enc}(S_i^-,x_i) \tag{2.5}$$

$$q(S_0|\vec{X}) = \mathcal{N}(\mu^s,\Sigma^S), \quad [\mu^S,\Sigma^S] = \phi_S^{enc}(\vec{x}) \tag{2.6}$$

$$q(E|\vec{X}) = \mathcal{N}(\mu^E,\Sigma^E), \quad [\mu^E,\Sigma^E] = \phi_E^{enc}(\vec{x}) \tag{2.7}$$

Learning. The generative and inference schemes are fully defined by the functions [f, g, ϕ_s^{enc}, ϕ_θ^{enc}, ϕ_β^{enc}] which are learned simultaneously. In the experiments, each of these functions is parameterized by a neural network, but in a more general scenario, they may be any differentiable functions since we apply SGD-based training. To match the posterior distributions of E, S_0 and $\vec{\beta}$ to the assigned prior probabilities $p_0(E)$, $p_0(S_0)$ and $p_0(\beta)$, we utilize reparametrization trick (Kingma and Welling, 2013; Rezende *et al.*, 2014) which enables end-to-end training. In order to define the objective function for our training, similar to (Kingma and Welling, 2013) we derive the variational lower bound $\mathcal{L}$ which we consequently attempt to maximize during the training. We start from the well-known equality (Kingma and Welling, 2013):

$$\mathcal{L} = \mathbb{E}_{q(\vec{S}|\vec{X})}[\log\, p(\vec{X}|\vec{S})] - \mathrm{KL}(q(\vec{S}|\vec{X})||p_0(\vec{S})) \tag{2.8}$$

Due to the conditional independence of the observations given the latent states, we can decompose the first term as:

$$\mathbb{E}_{q(\vec{S}|\vec{X})}[\log\, p(\vec{X}|\vec{S})] = \sum_{i=1}^{T} \mathbb{E}_{q(S_i|\vec{X})}[\log\, p(X_i|S_i)] \tag{2.9}$$

The KL term can be shown to simplify into a sum of the following KL terms:

$$\begin{aligned} \mathrm{KL}(q(\vec{S}|\vec{X})||p_0(\vec{S})) = &\ \mathrm{KL}(q(E|\vec{X})||p_0(E)) \\ &+ \mathrm{KL}(q(s_0|\vec{X})||p_0(S_0)) \\ &+ \sum_{i=1}^{T} \mathop{\mathbb{E}}_{q(\beta_i,E,S_{i-1}|\vec{X})} \mathrm{KL}(q(\beta_i)||p_0(\beta)) \end{aligned} \tag{2.10}$$

where we dropped the conditional dependency $\beta_i|S_{i-1},X_i,E$ in q to ease the notation. The exact steps of the $\mathcal{L}$ derivation are given in the Appendix A.0.2. Algorithm 1 shows the details of the training procedure for one iteration, for a batch of size one. The extension to the full batch training is straight forward.

Implementation. All learnable functions of DSSM i.e., [f, g, ϕ_S^{enc}, ϕ_E^{enc}, ϕ_β^{enc}] are modelled as deep neural networks. In our experiments, g and ϕ_β^{enc} are deconvolutional

Algorithm 1 One iteration of DSSM training procedure

Input: sequence $\vec{X}$ of length T
$[\mu^E, \Sigma^E] = \phi_E^{enc}(\vec{X}), \quad E \sim \mathcal{N}(\mu^E, \Sigma^E)$
$[\mu^S, \Sigma^S] = \phi_S^{enc}(\vec{X}), \quad s_0 \sim \mathcal{N}(\mu^S, \Sigma^S)$
for $i = 1$ **to** T **do**
    Predict step: $S_i^- = f(S_{i-1}, E)$
    Estimate residual: $[\mu_i^\beta, \Sigma_i^\beta] = \phi_\beta^{enc}(S_i^-, X_i), \quad \beta_i \sim \mathcal{N}(\mu_i^\beta, \Sigma_i^\beta)$
    Update step: $S_i = S_i^- + \beta_i$
    Predict observation: $\hat{X}_i = g(S_i)$
end for
ll_loss = LL$(\vec{X}, \vec{\hat{X}})$ (see Eq (2.9))
kl_loss = KL$(E, S_0, \vec{\beta})$; (see Eq (2.10))
mm_loss = MM$(\phi_E^{enc}(\vec{X}))$; (see Eq (2.11))
Backpropagate(ll_loss, kl_loss, mm_loss)

and convolutional neural networks respectively. ϕ_S^{enc} and ϕ_E^{enc} are implemented as bi-directional LSTM followed by a multilayer perceptron to convert LSTM-based sequence embedding into S_0 and E, and f as an LSTM cell. More information on the architecture are included in Appendix A.0.3.

Optimization challenges. Some performance improvements were observed with an additional heuristic regularization term, which ensures the consistency during the inference of environment E. The main idea is: since E is designed to capture time-invariant, global information of the state dynamics which by design influences state transition in each time step, we penalize the step-wise change in time embeddings produced by bi-directional LSTM used to model ϕ_E^{enc}, in order to enforce E to remain time-invariant. To that end, we add an additional term to our objective function, the moment matching regularization term defined as:

$$MM(\phi_E^{enc}(\vec{X})) = \sum_{i=2}^{T} ||h_i - h_{i-1}||^2 \tag{2.11}$$

where h_i is the hidden state of the ϕ_E^{enc} LSTM cell in step i. This idea is related to the approaches based on the maximum mean discrepancy (Gretton *et al.*, 2012). Namely, enforcing equality of consecutive cell states corresponds to matching of their first moments. Furthermore, similarly to (Bowman *et al.*, 2015; Karl *et al.*, 2016) we used a KL annealing scheme. This was helpful for circumventing local minimum and preventing the KL term to converge to zero too early during the training. The exact details are given in our experiments below.

2.5 Experiments

Dataset. We test our framework on a 2D bouncing ball problem where the ball kinematics is affected by a varying gravity vector (E). The idea is to evaluate model robustness across environments. Here each environment is characterized by different gravitational settings. See also visualizations at: `https://sites.google.com/view/dssm`. Using the physics engine library pymunk [1], we simulate video sequences of a bouncing ball. During generation, we randomly change the gravity such that it remains constant within a sequence, but may vary across sequences. The gravity vector takes 4 values, depending on whether the gravity points up, down, left or right. Each video frame is a 32x32 binary image. We generate 16'000 trajectories across 40 time steps for training, and another 2000 trajectories across 70 time steps for testing.

Network details. To get compressed representation of each frame, the images are first passed through a shallow convolutional network. Kernel size was set to 3x3, while the network depth was 64. The step size was 1 in both directions. We used ReLU activation units. For environment variable E, we took the last hidden representation of the network as the output, whereas for s_0 we took the first hidden representation as the output. All of the hidden latent states were equal to 64. To parameterize g we used a deconvolutional network with transposed convolutions. The kernel size was set to 5. See Appendix A.0.1 for further training details.

2.5.1 Prediction and controlled generation of video sequences

We first perform long-term forecasting analysis comparing our method against one of the state-of-the-art approaches, the K-VAE from (Fraccaro *et al.*, 2017). Next, we demonstrate controlled generation, by manipulating video sequences. Effectively, we perform interventions by "swapping environments" between video sequences, and similarly we swap initial states. Furthermore, we show the ability to perform uncontrolled generation where both initial state and gravity vector are sampled from a prior. Finally we visualize the environment embeddings to provide further intuition.

Forecasting. We evaluate the forecasting performance by comparing our method against K-VAE Fraccaro *et al.* (2017). We use OpenCV [2] inbuilt functions to detect ground truth ball position p_t in each time frame. The exact algorithm for the position extraction is provided in the Appendix A.0.4. Following (Hsieh *et al.*, 2018; Chang *et al.*, 2016) we define the ball velocity as $v_t = p_{t+1} - p_t$ and compute the relative error in the predicted velocities of the balls for forecasting. Evaluated models observe the first 25 frames of a test sequence and then forecast the next 45. The results for relative error in magnitude

[1] https://www.pymunk.org/en/latest/
[2] https://opencv.org/

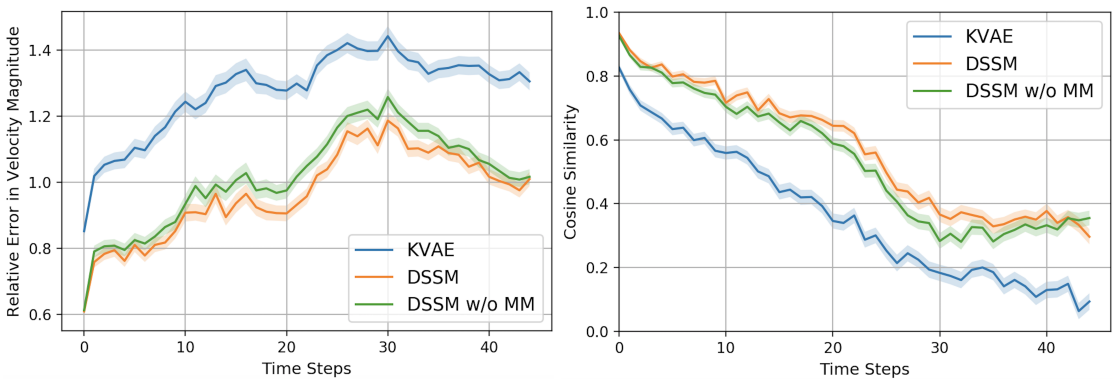


Figure 2.4: **Bouncing ball trajectory forecasting.** DSSM-based Bayesian filter against K-VAE on the task of long-term forecasting: **Left:** velocity magnitude; **Right:** cosine similarity. Here, MM corresponds to moment-matching regularization. Shown error curves are the test set averages.

and cosine similarity of the velocities are then averaged across all test sequences. This is shown in Figure 2.4. We observe an increase in prediction quality with respect to both metrics in comparison to the bench-marking model K-VAE.

Controlled generation. Next, we experiment by extracting embeddings of the gravity context vector E and the initial state s_0 from different test sequences in order to perform feature swapping and controlled generation, similarly to (Yingzhen and Mandt, 2018). Two scenarios are considered. First, the initial state which consists of the velocity vector and the ball position, is extracted from the baseline video sequence and injected into a series of other test sequences. Similarly, we performed the context replacement for the gravity vector E. After feature swapping, we rolled-out the sequences effectively performing controlled generation. Figure 2.5 shows qualitative results.

Environment identification. While the controlled generation in Figure 2.5 serves more as a qualitative analysis of the disentanglement, we additionally evaluate the disentanglement through the quantitative analysis. More specifically, we trained an auxiliary multi-layer perceptron classifier to map E to true gravity value. The cross-validation results performed on the training set rendered accuracy of 99.15%. In Figure 2.6 (left), we can see that DSSM environment variable $E \in \mathbb{R}^3$ contains information of the underlying gravity dynamics. Moreover, we provide 2D t-SNE visualizations of the embeddings (for $E \in \mathbb{R}^3$) in Figure 2.6 (right). Well-defined clusters can be observed, indicating that E indeed represents the true gravity. It is interesting to visualize that the discovered environment values follow the real-world geometric directional aspects i.e. the embeddings of downwards directed gravity *(negative y-axis)* lies opposite to the upwards directed

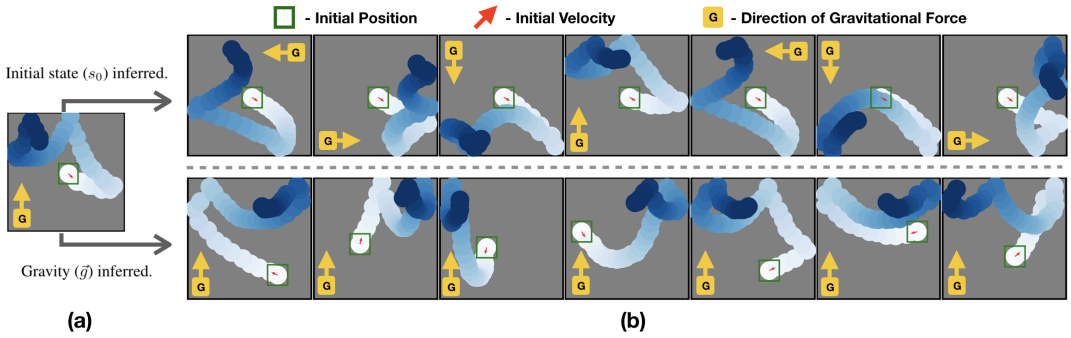


Figure 2.5: **Environment swapping and controlled generation. (a)** base sequence. **(b)** test sequences in which we injected environment gravity value (bottom-row) and initial state (top-row) of the base sequence inferred using networks ϕ_E^{enc} and ϕ_S^{enc} respectively.

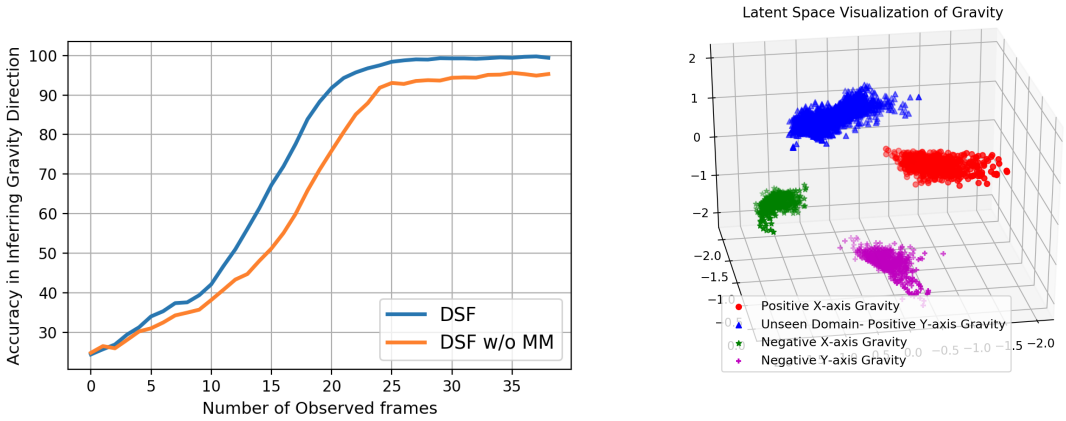


Figure 2.6: **Identifying the environment E. Left:** Quantitative analysis of inferring the correct gravitational direction from only a few observations. **Right:** Visualization of the t-SNE embeddings of the learned environment E. It can be seen that the corresponding environments cluster together which indicates that the model can correctly infer and map the environments in the latent space.

gravity *(positive y-axis)*. The same is true for the horizontally directed gravities.

Uncontrolled generation. We further demonstrate the uncontrolled generation of the sequences where s_0 and E are sampled from the priors $p_0(E)$ and $p_0(S_0)$ respectively. In Figure 2.7 we show how the generated sequences preserve natural bouncing ball dynamics. We refer readers to `https://sites.google.com/view/dssm` for the actual visualizations of these generated videos.

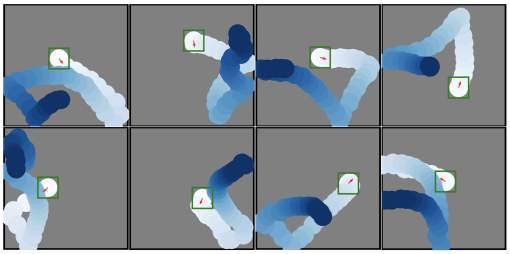


Figure 2.7: **Uncontrolled video generation.** Here both the initial state of the ball and the gravity are sampled from the prior

2.6 Conclusion

This chapter proposes a novel view on data-driven learning of dynamics from diverse environments. We proposed a new class of state space models particularly crafted to exploit this kind of a setting. In disentangled state space models one separates generic system dynamics which is assumed to be invariant across environments and environment-specific information which governs this dynamics. We showed that such separation is beneficial and allows us to learn robust cross-environment models which hold promise to generalize on unseen environments. Our particular application was learning of the toy video dynamics of a bouncing ball affected by varying gravitational influences where we achieved state-of-the-art results.

Chapter 3

On the Transfer of Inductive Bias from Simulation to the Real World

"What we observe as material bodies and forces are nothing but shapes and variations in the structure of space. Particles are just schaumkommen (appearances)."

— Erwin Schrödinger

Learning meaningful and compact representations of data with disentangled semantic aspects are considered to be of key importance in representation learning Bengio *et al.* (2013). Since real-world data is notoriously costly to collect, many recent state-of-the-art disentanglement models have heavily relied on synthetic toy datasets. In this chapter, we propose a novel dataset which consists of over one million images of physical 3D objects with seven factors of variation, such as object color, shape, size and position. In order to be able to control all the factors of variation precisely, we built an experimental platform where the objects are being moved by a robotic arm. In addition, we provide two more datasets which consist of simulations of the experimental setup. These datasets provide for the first time the possibility to systematically investigate how well different disentanglement methods perform on real data in comparison to simulation, and how simulated data can be leveraged to build better representations of the real world. We provide a first experimental study of these questions and our results indicate that learned models transfer poorly, but that model and hyperparameter selection is an effective means of transferring information to the real world.

Based on (Gondal *et al.*, 2019a). Muhammad Waleed Gondal, Manuel Wüthrich, Đorđe Miladinović, Francesco Locatello, Martin Breidt, Valentin Volchkov, Joel Akpo, Olivier Bachem, Bernhard Schölkopf, Stefan Bauer. "On the Transfer of Inductive Bias from Simulation to the Real World." In: Advances in Neural Information Processing Systems (NeurIPS), 2019.

3.1 Introduction

In representation learning it is commonly assumed that a high-dimensional observation $\mathbf{X}$ is generated from low-dimensional factors of variation $\mathbf{G}$. The goal is usually to revert this process by searching for a latent embedding $\mathbf{Z}$ which replicates the underlying generative factors $\mathbf{G}$, e.g. shape, size or color. Learning well-*disentangled* representations of complex sensory data has been identified as one of the key challenges in the quest for artificial intelligence (AI) (Bengio *et al.*, 2013; Peters *et al.*, 2017; LeCun *et al.*, 2015; Bengio *et al.*, 2007; Schmidhuber, 1992; Lake *et al.*, 2017; van Steenkiste *et al.*, 2019), since they should contain all the information present in the observations in a compact and interpretable structure (Bengio *et al.*, 2013; Kulkarni *et al.*, 2015; Chen *et al.*, 2016) while being independent from the task at hand (Goodfellow *et al.*, 2009; Lenc and Vedaldi, 2015).

Disentangled representations may be useful for (semi-)supervised learning of downstream tasks, transfer and few-shot learning (Bengio *et al.*, 2013; Schölkopf *et al.*, 2012; Locatello *et al.*, 2019a). Further, such representations allow to filter out nuisance factors (Kumar *et al.*, 2017), to perform interventions and to answer counterfactual questions (Pearl, 2009; Spirtes *et al.*, 1993; Peters *et al.*, 2017). First applications of algorithms for learning disentangled representations have been applied to visual concept learning, sequence modeling, curiosity-based exploration or even domain adaptation in reinforcement learning Steenbrugge *et al.* (2018); Laversanne-Finot *et al.* (2018); Nair *et al.* (2018); Higgins *et al.* (2017b, 2018a); Lesort *et al.* (2018); van Steenkiste *et al.* (2019). The research community is in general agreement on the importance of this paradigm and much progress has been made in the past years, particularly on the algorithmic level (e.g. Higgins *et al.*, 2017a; Kim and Mnih, 2018), fundamental understanding (e.g. Higgins *et al.*, 2018b; Suter *et al.*, 2019) and experimental evaluation Locatello *et al.* (2018). However, research has thus far focused on synthetic toy datasets.

The main motivation for using synthetic datasets is that they are cheap, easy to generate and the independent generative factors can be easily controlled. However, real-world recordings exhibit *imperfections* such as chromatic aberrations in cameras and complex surface properties of objects (e.g., reflectance, radiance and irradiance), making transfer learning from synthetic to real data a nontrivial task. Despite the growing importance of the field and the potential societal impact in the medical domain or fair decision making (e.g. Chartsias *et al.*, 2018; Creager *et al.*, 2019; Locatello *et al.*, 2019b), the performance of state-of-the-art disentanglement learning on real-world data is unknown.

To bridge the gap between simulation and the physical world, we built a recording platform which allows to investigate the following research questions: (i) How well do unsupervised state-of-the-art algorithms transfer from rendered images to physical recordings? (ii) How much does this transfer depend on the quality of the simulation? (iii) Can we learn representations on low dimensional recordings and transfer them from the current state-of-the-art of 64×64 images to high quality images? (iv) How much supervision is necessary to encode the necessary inductive biases? (v) Are the confounding

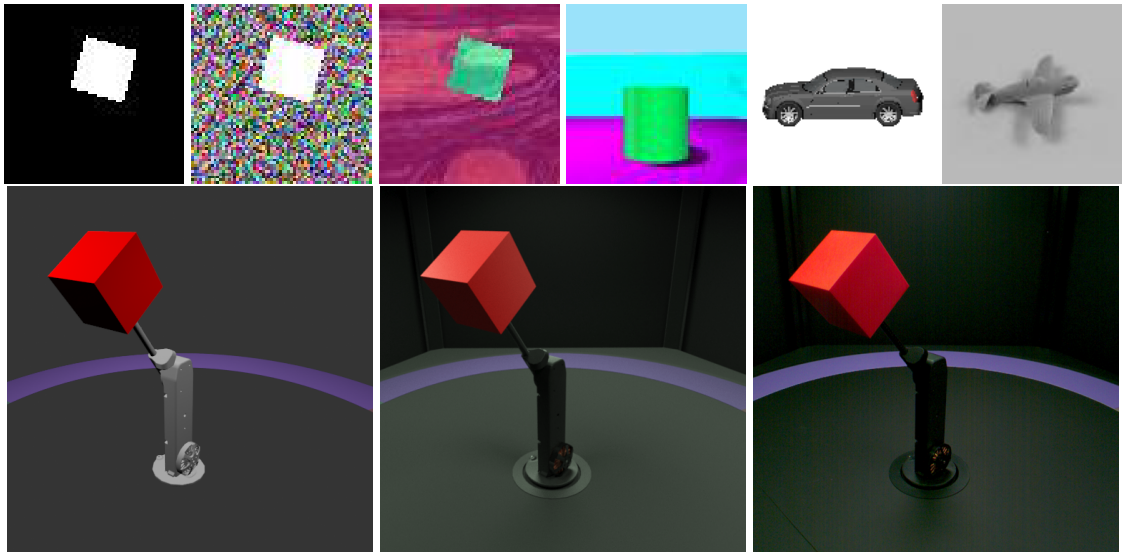


Figure 3.1: **Datasets:** In the top row samples from previously published datasets are shown from left to right: dSprites, Noisy-dSprites, Scream-dSprites, 3dshapes, Cars3D and SmallNORB. In the second row (again left to right) we provide simple simulated data, highly realistically simulated data and real-world data examples of the newly collected dataset.

and distortions of real-world recordings beneficial for learning disentangled representations? (vi) Can we disentangle causal mechanisms (Pearl, 2009; Lake *et al.*, 2015, 2017; Peters *et al.*, 2017) in the data generating process? (vii) Are disentangled representations useful for solving the real-world downstream tasks?

While answering all of the above questions is beyond the scope of this chapter, our key contributions can be summarized as follows.

- We introduce the first *real-world 3D dataset* recorded in a controlled environment, defined by *7 factors of variations*: object color, object shape, object size, camera height, background color and two degrees of freedom of motion of a robotic arm. The dataset is made publicly available[1].
- We provide synthetic images produced by computer graphics with two levels of realism. The robot arm and the objects are printed from a 3D template, ensuring close similarity between the realistic renderings and the real-world recordings.
- The collected datatset of 3D objects consists of over one million images, and each of the two simulated datasets contains the same number of images as well.
- We investigate the role of inductive bias and the transfer of different hyper-parameter settings between the different simulations and the real-world and the requirements on the quality of the simulation for a succesful transfer.

[1] `https://github.com/rr-learning/disentanglement_dataset`

3.2 Background and related work

We assume a set of observations of a (potentially high dimensional) random variable $\boldsymbol{X}$ which is generated by K unobserved causes of variation (generative factors) $\boldsymbol{G} = [G_1, \ldots, G_K]$ (i.e., $\boldsymbol{G} \rightarrow \boldsymbol{X}$) that do not cause each other. These latent factors represent *elementary ingredients* to the causal mechanism generating $\boldsymbol{X}$ Pearl (2009); Peters *et al.* (2017). The elementary ingredients $G_i, i = 1, \ldots, K$ of the causal process work on their own and are changeable without affecting others, reflecting the independent mechanisms assumption Schölkopf *et al.* (2012). However, for some of the factors a hierarchical structure may exist for which this may only hold true when seeing the hierachical structure as a whole as one component. The graphical model corresponding to this framework and adapted from Suter *et al.* (2019) is depicted in Figure 3.2. The hierarchical structure of the factors G^1_{K-1} and G_{K-1} might represent one compositional process e.g., connected joints of a robot arm. The most commonly accepted understanding of *disentanglement* Bengio *et al.* (2013) is that each learned feature in $\boldsymbol{Z}$ should capture one factor of variation in $\boldsymbol{G}$.

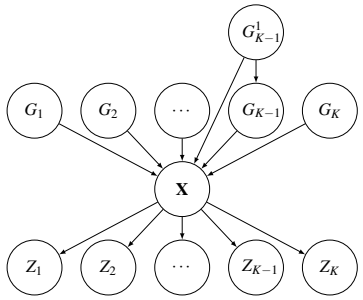


Figure 3.2: Graphical Model, where $\mathbf{G} = (G_1, G_2, \ldots, G_K)$ are the generative factors (color, shape, size, ...) and $\mathbf{X}$ the recorded images. The aim of disentangled representation learning is to learn variables Z_i that capture the independent mechanisms G_i.

Current state-of-the-art disentanglement approaches use the framework of variational auto-encoders (VAEs) Kingma and Welling (2013). The (high-dimensional) observations $\boldsymbol{x}$ are modelled as being generated from some latent features $\boldsymbol{z}$ with chosen prior $p(\boldsymbol{z})$ according to the probabilistic model $p_\theta(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})$. The generative model $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ as well as the proxy posterior $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ can be represented by neural networks, which are optimized by maximizing the variational lower bound (ELBO) of $\log p(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N)$.

$$\mathcal{L}_{VAE} = \sum_{i=1}^{N} \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x}^{(i)})}[\log p_\theta(\boldsymbol{x}^{(i)}|\boldsymbol{z})] - D_{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x}^{(i)})||p(\boldsymbol{z}))$$

Since the above objective does not enforce any structure on the latent space, except for some similarity to the typically chosen isotropic Gaussian prior $p(\boldsymbol{z})$, various proposals for more structure imposing regularization have been made. Using some sort of supervision (e.g. Narayanaswamy *et al.*, 2017; Bouchacourt *et al.*, 2018; Liu *et al.*, 2017; Mathieu *et al.*, 2016; Cheung *et al.*, 2015) or proposing completely unsupervised (e.g. Higgins *et al.*, 2016; Kim and Mnih, 2018; Chen *et al.*, 2018; Kumar *et al.*, 2017; Esmaeili *et al.*, 2018) learning approaches. Higgins *et al.* (2016) proposed the β-VAE penalizing the Kullback-Leibler divergence (KL) term in the VAE objective more strongly, which encourages similarity to the factorized prior distribution. Others used techniques to encourage statistical independence between the different components in $\boldsymbol{Z}$, e.g., FactorVAE

Dataset	Factors of Variation	Resolution	# of Images	3D	Real-World
dSprites	5	64×64	737,280	✗	✗
Noisy dspirtes	5	64×64	737,280	✗	✗
Scream dSprites	5	64×64	737,280	✗	✗
SmallNORB	5	128×128	48,600	✓	✗
Cars3D	3	64×64	17,568	✓	✗
3dshapes	6	64×64	480,000	✓	✗
MPI3D-toy	7	64×64	1,036,800	✓	✗
MPI3D-realistic	7	256×256	1,036,800	✓	✗
MPI3D-real	7	512×512	1,036,800	✓	✓

Table 3.1: **Summary of the properties of different datasets.** The newly contributed datasets in this work are *emphasized*.

Kim and Mnih (2018) or β-TCVAE Chen *et al.* (2018), while DIP-VAE proposed to encourage factorization of the inferred prior $q_\phi(\boldsymbol{z}) = \int q_\phi(\boldsymbol{z}|\boldsymbol{x})p(\boldsymbol{x})\,d\boldsymbol{x}$. For other related work we refer to the detailed descriptions in the empirical study Locatello *et al.* (2018).

3.2.1 Established datasets for the unsupervised learning of disentangled representations

Real-world data is costly to generate and groundtruth is often not available since significant confounding may exist. To bypass this limitation, many recent state-of-the-art disentanglement models Watters *et al.* (2019); Kim and Mnih (2018); Chen *et al.* (2018); Higgins *et al.* (2017a); Chen *et al.* (2016) have heavily relied on synthetic toy datasets, trying to solve a simplified version of the problem in the hope that the conclusions drawn might likewise be valid for real-world settings. A quantitative summary of the most widely used datasets for learning disentangled representations is provided in Table 3.1.

Datasets descriptions. For quantitative analysis, *dSprites*[2] is the most commonly used dataset. This synthetic dataset Higgins *et al.* (2017a) contains binary 2D images of hearts, ellipses and squares in low resolution. In *Color-dSprites* the shapes are colored with a random color, *Noisy-dSprites* considers white-colored shapes on a noisy background and in *Scream-dSprites* the background is replaced with a random patch in a random color shade extracted from the famous The Scream painting Munch (1893). The dSprites shape is embedded into the image by inverting the color of its pixel. The *SmallNORB*[3] dataset

[2] `https://github.com/deepmind/dsprites-dataset`
[3] `https://cs.nyu.edu/~ylclab/data/norb-v1.0-small/`

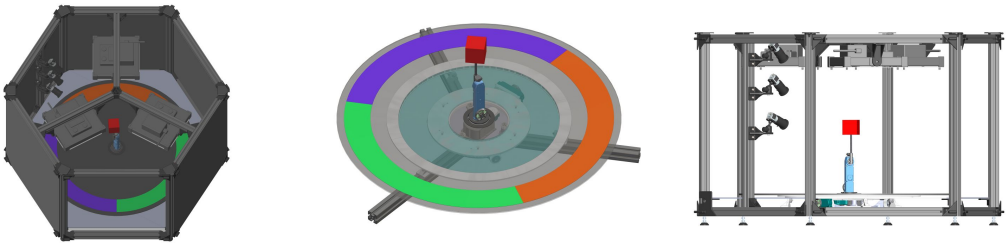


Figure 3.3: **Renderings of the developed robotic platform.** (left) A view from a 30° angle from the top (note that one panel in front and the top panels have been removed such that the interior of the platform is visible. During recordings, the platform is entirely closed. (Center) The robotic arm carrying a red cube (the entire cage is hidden). (Right) Frontal view without the black shielding (note the three cameras at different heights).

contains images of 50 toys belonging to 5 generic categories: four-legged animals, human figures, airplanes, trucks, and cars. The objects were imaged by two cameras under 6 lighting conditions, 9 elevations (30 to 70 degrees every 5 degrees), and 18 azimuths (0 to 340 every 20 degrees) LeCun *et al.* (2004). For *Cars3D*[4],199 CAD models from Fidler *et al.* (2012) were used to generate 64x64 color renderings from 24 rotation angles each offset by 15 degrees Reed *et al.* (2015). Recently, *3dshapes* was made publicly available[5], a dataset of 3D shapes procedurally generated from 6 ground truth independent latent factors. These factors are floor color, wall color, object color, scale, shape and orientation Kim and Mnih (2018).

3.3 Bridging the gap between simulation and the real world: A novel dataset

While other real-world recordings, e.g., *CelebA* Liu *et al.* (2015), exist, they offer only qualitative evaluations i.e., detailed description for the underlying factors of variations does not exist. However, a more controlled dataset is needed to quantitatively investigate the effects of inductive biases, sample complexity and the interplay of simulations and the real-world.

3.3.1 Controlled recording setup

In order to record a controlled dataset of physical 3D objects, we built the mechanical platform illustrated in Figure 3.3. It consists of three cameras mounted at different heights, a robotic manipulator carrying a 3D printed object (which can be swapped) in

[4] `http://www.scottreed.info/files/nips2015-analogy-data.tar.gz`
[5] `https://github.com/deepmind/3dshapes-dataset/`

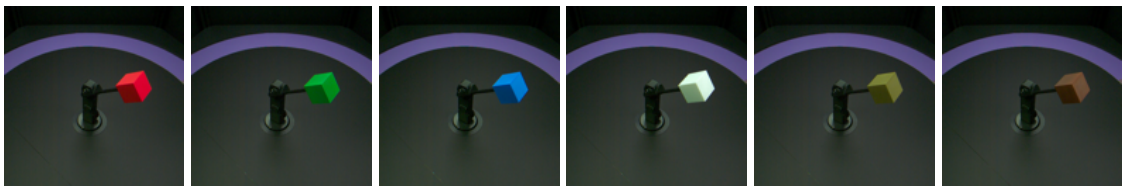


Figure 3.4: We show all the object colors while maintaining the other factors constant.

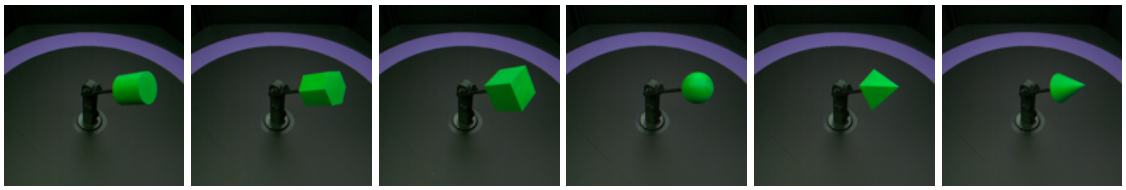


Figure 3.5: We show all object shapes while maintaining all other factors constant.

the center of the platform and a rotating table at the bottom. The platform is shielded with black sheets from all sides to avoid any intrusion of external factors (e.g., light) and the whole environment is relatively uniformly illuminated by three light sources installed within the platform.

3.3.2 Factors of variation

The generative factors of variation $\boldsymbol{G}$ mentioned in Section 3.2 are listed in the following for our recording setup.

Object color. All objects have one of six different colors: red (255, 0, 0), green (0, 255, 0), blue (0, 0, 255), white (255, 255, 255), olive (210,210,80) and brown (153,76,0) (see Figure 3.4).

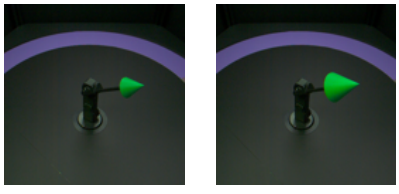


Figure 3.6: We show the two object sizes while maintaining all other factors constant.

Object shape. There are objects of four different shapes in the dataset: a cylinder, a hexagonal prism, a cube, a sphere, a pyramid with square base and a cone. All objects exhibit rotational symmetries about some axes, however the kinematics of the robot are such that these axes never align with the degrees of freedom of the robot. This is important because it ensures that the robot degrees of freedom are observable given the images.

Object size. There are objects of two different sizes in the dataset, categorized as large (roughly 65mm in diameter) and small (roughly 45 mm in diameter).

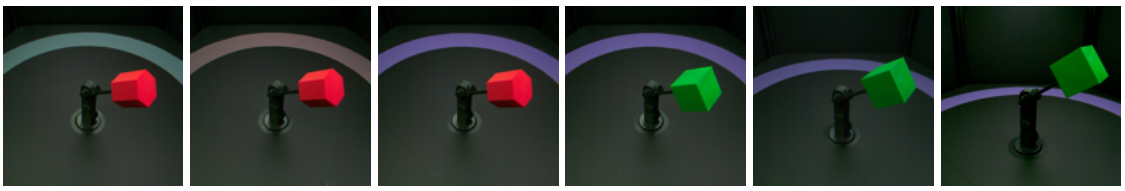


Figure 3.7: **Background Color.** Three images on the left: we vary the backround color. **Camera Height.** Three images on the right: we vary the camera height.

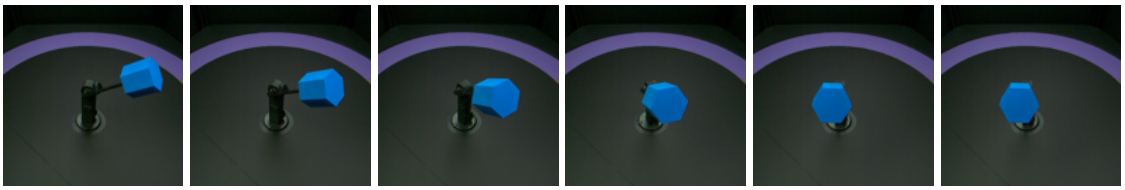


Figure 3.8: Motion along first DOF while maintaining the other factors constant. Note that in total we record 40 steps, here we only show 6 due to space constraints.

Camera height. The dataset is recorded with three cameras mounted at three different heights (see Figure 3.7 on the right), representing another factor of variation in images.

Background color. The rotation table (see Figure 3.7) allows us to change background color. Note that for all images in the dataset we orient the table in such a way that only one background color is visible at a time. The colors are sea green, salmon and purple.

Degrees of freedom of the robotic arm. Each object is mounted on the tip of the manipulator shown in Figure 3.3. This manipulator has two degrees of freedom, a rotation about a vertical axis at the base and a second rotation about a horizontal axis. We move each joint in a range of 180° in 40 equal steps (see Figure 3.8 and Figure 3.9). Note that these two factors of variation are independent, just like all other factors (i.e., we record all possible combinations between the two).

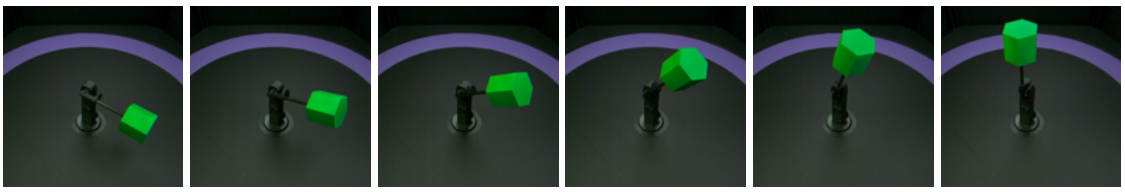


Figure 3.9: Motion along second DOF while maintaining the other factors constant. Note that in total we record 40 steps, here we only show 6 due to space constraints.

3.3.3 Simulated data

In addition to the real-world dataset we recorded two simulated datasets of the same setup, hence all factors of variation are identical across the three datasets. One of the simulated datasets is designed to be as realistic as possible and the synthetic images are visually practically indistinguishable from real images (see Figure 3.1 center). For the second simulated dataset we used a deliberately simplified model (see figure Figure 3.1 left), which allows to investigate transfer from simplified models to the real data.

The synthetic data was generated using Autodesk 3ds Max(2018). Most parts of the scene were imported from SolidWorks CAD files that were originally used to construct the experimental stage including the manipulator and 3D printing of the test objects. The surface shaders are based on Autodesk Physical material with hand-tuned shading parameters, based on full resolution reference images. The camera poses were initialized from the CAD data and then manually fine-tuned using reference images. The realistic synthetic images were obtained using the Autodesk Raytracer (ART) with three rectangular light sources, mimicking the LED panels. The simplified images were rendered with the Quicksilver hardware renderer.

3.4 Experiments

Some fields have been able to narrow the gap between simulation and reality Zhang *et al.* (2019); Bousmalis *et al.* (2016); James *et al.* (2018), which has led to remarkable achievements (e.g., for in-hand manipulation Andrychowicz *et al.* (2018)). In contrast, for disentanglement methods this gap has not been bridged yet, state-of-the-art algorithms seem to have difficulties to transfer learned representations even between toy datasets Locatello *et al.* (2018). The proposed dataset will enable the community to systematically investigate how such transfer of information between simulations with different degrees of realism and real data can be achieved. In the following we present a first experimental study in this directions.

3.4.1 Experimental protocol

Metrics. Various methods to validate a learned representation for disentanglement based on known ground truth generative factors $\boldsymbol{G}$ have been proposed (e.g. Eastwood and Williams, 2018; Ridgeway and Mozer, 2018; Chen *et al.*, 2018; Kim and Mnih, 2018). This has for example been expressed as the mutual information of a single latent dimension Z_i with generative factors $G_1, \ldots, G_K$ Ridgeway and Mozer (2018), where in the ideal case each Z_i has some mutual information with one generative factor G_k but none with all the others. Similarly, Eastwood and Williams (2018) trained predictors (e.g., Lasso or random forests) for a generative factor G_k based on the representation $\boldsymbol{Z}$. In a disentangled model, each dimension Z_i is only useful (i.e., has high feature importance)

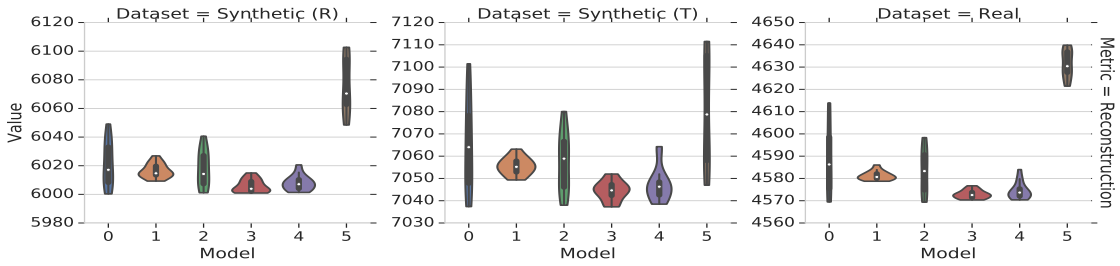


Figure 3.10: **Reconstruction scores of different methods.** Scores achieved by (0=β-VAE, 1=FactorVAE, 2=β-TCVAE, 3=DIP-VAE-I, 4=DIP-VAE-II, 5=AnnealedVAE) on the realistic synthetic dataset, the toy synthetic dataset and the real dataset.

to predict one of those factors. Suter *et al.* (2019) proposed an interventional robustness score. The graphical model of Suter *et al.* (2019) adapted to our setup is illustrated in Figure 3.2. Another form of validation, especially without known generative factors is the visual inspection of *latent traversals* (see e.g., Chen *et al.*, 2018).

We apply all the disentanglement methods (β-VAE, FactorVAE, β-TCVAE, DIP-VAE-I, DIP-VAE-II, AnnealedVAE) which were used in a recent large-scale study Locatello *et al.* (2018) to our three datasets. Due to space constraints, the models are abbreviated with numbers one to five in the plots in the same order. We use disentanglement library [6] and evaluate on the same scores as Locatello *et al.* (2018). In all the experiments, we used images with resolution 64x64. This resolution is used in the recent large-scale evaluations and by state-of-the-art disentanglement learning algorithms Locatello *et al.* (2018). Each of the six methods is trained on each of the three datasets with five different hyperparameter settings (see table B.1 in the appendix for details) and with three different random seeds, leading to a total of 270 models. Each model is trained for 300,000 iterations on Tesla V100 gpus.

Architecture. All the models used the same convolutional encoder and decoder architecture with the fixed latent size of 10. Further details on model and their training hyperparameters are provided in Appendix B.2.

3.4.2 Results

Reconstruction across datasets. Figure 3.10 shows that there is a difference in reconstruction score across datasets: The score is the lowest on real data, followed by the realistic simulated dataset (R) and the simple toy (T) images. This indicates that there is a significant difference in the distribution of the real data compared to the simulated data, and that it is harder to learn a representation of the real data than of the simulated data. However, the relative behaviour of different methods seems to be similar across all three

[6] `https://github.com/google-research/disentanglement_lib`

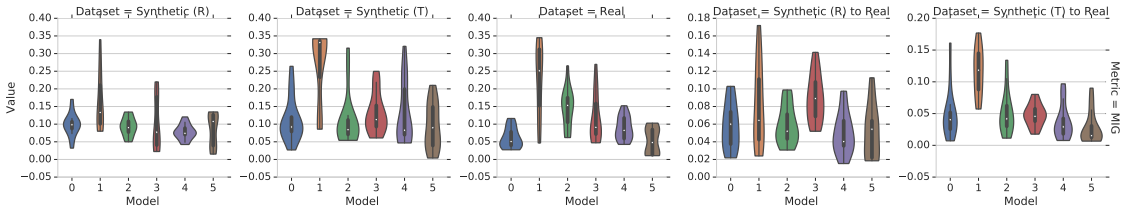


Figure 3.11: **Mutual Information Gap (MIG) scores attained by different methods.** From left to right: (a) trained and evaluated on synthetic realistic, (b) trained and evaluated on synthetic toy, (c) trained and evaluated on real, (d) trained on synthetic realistic and evaluated on real, (e) trained on synthetic toy and evaluated on real. The variance is due to different hyper-parameters and seeds.

datasets, which indicates that despite the differences, the simulated data may be useful for model selection.

Direct transfer of representations. In Figure 3.11 we show the Mutual Information Gap (MIG) scores attained by different methods for different evaluations. The same plots for different metrics look qualitatively similar (see Figure B.5 in the appendix). Given the high variance, it is difficult to make conclusive statements. However, it seems quite clear that all methods perform significantly better when they are trained and evaluated on the same dataset (three plots on the left). Direct transfer of learned representations from simulated to real data (two plots on the right) seems to work rather poorly.

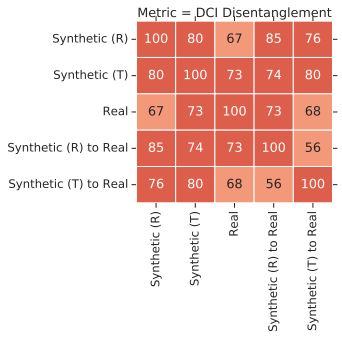


Figure 3.12: Rank-correlation of the DCI disentanglement scores of different models (including hyperparameters) across different data sets.

Transfer of hyperparameters. We have seen that transferring representations directly from simulated to real data seems to work poorly. However, it may be possible to instead transfer information at a higher-level, such as the choice of the method and its hyperparameters as an inductive bias. In order to quantitatively evaluate whether such a transfer is possible, we pick the model (including hyperparameters) which performs best in simulation (according to a metric chosen at random), and we compute the probability of outperforming (according to a metric and seed chosen randomly) a model which was chosen at random. If no transfer is possible, we would expect this probability to be 50%. However, we find that model selection from realistic simulated renderings (R) outperforms random model selection 72% of the time while transferring the model from the simpler synthetic images (T) to real-world data even beats random selection 78% of the time. This finding is confirmed by Figure 3.12, where we show the rank-correlation of the

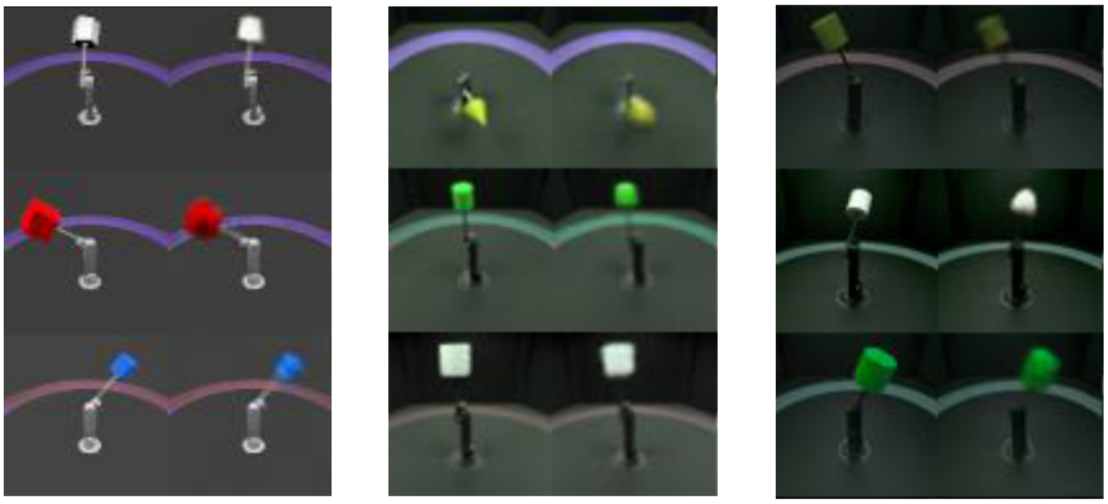


Figure 3.13: **Reconstruction of different datasets samples by Factor VAE. (left)** reconstructed samples corresponding to simple toy dataset, **(center)** realistic dataset **(right)** and real-world dataset. In each figure, left column corresponds to the groundtruth.

performance of models (including hyperparameters) trained on one dataset with the performance of these models trained on another dataset. The performance of a model trained on some dataset seems to be highly correlated with the performance of that model trained on any other dataset. In Figure 3.12 we use the DCI disentanglement metric as a score, however, qualitatively similar results can be observed using most of the disentanglement metrics (see Figure B.8 in the appendix).

Summary. These results indicate that the simulated and the real data distribution have some similarities, and that these similarities can be exploited through model and hyperparameter selection. Surprisingly, it seems that the transfer of models from the synthetic toy dataset may work even better than the transfer from the realistic synthetic dataset.

3.5 Discussion on disentangled representations transfer

Empirically, we observed that some of the best trained models were able to disentangle, though imperfectly, the factors of camera height, background colors, object sizes and the motions along the first and second degrees of freedom. Whereas, they performed poorly in disentangling the factors of some object shapes (e.g., pyramid and cone) and some colors (e.g., olive and brown). This may well be because of less pixel variation in the respective factors. The features of camera height and background color cause the most difference (maximum L2 distance) in image space. Similarly the object positions at different joint configurations (not the consecutive frames) also have big L2 distance which may explain why the models focus more on learning these factors.

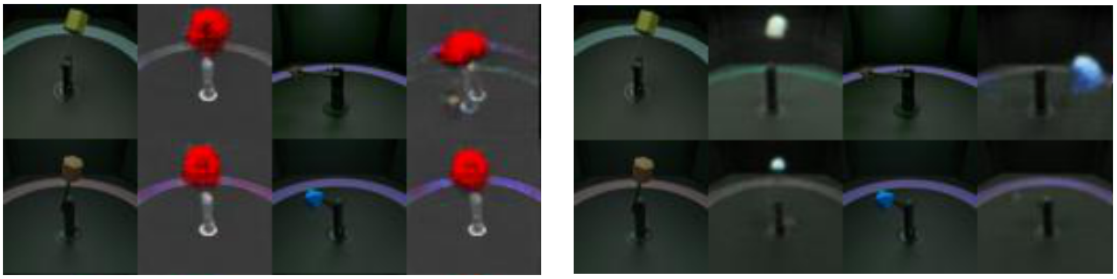


Figure 3.14: **Qualitative analysis of transferring representations for reconstructions. (left)** Image reconstructions corresponding to representations of the Factor VAE model trained on toy dataset and tested on the real-world dataset. **(right)** trained on realistic dataset and tested on real-world dataset.

The image reconstructions become more blurry as we move from the simple simulated dataset to the more complex real-world dataset, which can be seen by the reconstruction scores in Figure 3.10. It has been previously noted that the use of overly simplistic priors like standard normal Gaussians in VAE models Kingma and Welling (2013) can lead to over-regularization Tomczak and Welling (2017). To achieve disentanglement in VAE models, Higgins *et al.* (2017a) put higher weights (i.e., $\beta > 1$) on the KL divergence which decreases the reconstruction quality. With the increased complexity in the dataset, the decrease in reconstruction quality becomes even more pronounced, as illustrated in Figure 3.13, depicting reconstructed samples for all three datasets. See Appendix B.3 for further analysis of qualitative results. In Figure 3.14 we show cross-dataset reconstructions i.e., transfer of representations from simple simulated and realistic simulated datasets to the real-world dataset respectively. The models completely fail in transferring the representations from the simple toy simulated to the real-world data. On the other hand, in the realistic simulated to real-world transfer (Figure 3.14(right)) the models sometimes reconstruct the correct background strip color, manipulator pose and the camera height factor. However, the object properties seem to differ a lot. This shows that in the case of complex environments, the models put more focus on learning the environment to get the better reconstruction accuracy than to learn the important but relatively small changing factors. See Appendix B.3 for more results. This behavior for VAE models has also been confirmed by Ha and Schmidhuber (2018).

3.6 Conclusion

Despite the intended applications of disentangled representation learning algorithms to real data in fields such as robotics, healthcare and fair decision making (e.g. Chartsias *et al.*, 2018; Creager *et al.*, 2019; Higgins *et al.*, 2017b), state-of-the-art approaches have only been systematically evaluated on synthetic toy datasets. Our work effectively com-

plements related efforts (e.g. Locatello *et al.*, 2018) to address current challenges of representation learning, offering the possibility of investigating the role of inductive biases, sample complexity, transfer learning and the use of labels using real-world images.

A key aspect of our datasets is that we provide rendered images of increasing complexity for the *same* setup used to capture the real-world recordings. The different recordings offer the possibility of investigating the question if disentangled representations can be transferred from simulation to the real world and how the transferability depends on the degree of *realism* of the simulation. Beyond the evaluation of representation learning algorithms, the proposed dataset can likewise be used for other tasks such as 3D reconstruction and scene rendering Eslami *et al.* (2018) or learning compositional visual concepts Higgins *et al.* (2017c). Furthermore, we are planning to use the novel experimental setup for recording objects with more complicated shapes and textures under more difficult conditions, such as dependence among different factors.

Chapter 4

Dynamic Inference with Neural Interpreters

"The whole is greater than the sum of its parts."

— Aristotle

Modern neural network architectures can leverage large amounts of data to generalize well within the training distribution. However, they are less capable of systematic generalization to data drawn from unseen but related distributions, a feat that is hypothesized to require compositional reasoning and reuse of knowledge. In this work, we present Neural Interpreters, an architecture that factorizes inference in a self-attention network as a system of modules, which we call *functions*. Inputs to the model are routed through a sequence of functions in a way that is end-to-end learned. The proposed architecture can flexibly compose computation along width and depth, and lends itself well to capacity extension after training. To demonstrate the versatility of Neural Interpreters, we evaluate it in two distinct settings: image classification and visual abstract reasoning on Raven Progressive Matrices. In the former, we show that Neural Interpreters perform on par with the vision transformer using fewer parameters, while being transferrable to a new task in a sample efficient manner. In the latter, we find that Neural Interpreters are competitive with respect to the state-of-the-art in terms of systematic generalization.

4.1 Introduction

Rule-based programming is the basis of computer science, and builds the foundation of symbolic-AI based expert systems that attempt to emulate human decision-making

Based on (Rahaman *et al.*, 2021). Nasim Rahaman*, Muhammad Waleed Gondal*, Shruti Joshi, Peter Gehler, Yoshua Bengio, Francesco Locatello, Bernhard Schölkopf. "Dynamic Inference with Neural Interpreters." In: Advances in Neural Information Processing Systems (NeurIPS), 2021.

to solve real-world problems. The process of inference entails channeling information through a chain of computational units (e.g., logical primitives) and culminates in a conclusion that answers a given query. Such systems have the advantage that they permit efficient reuse of computational units and enable iterative reasoning over multiple cycles of computation. As a simple example, consider the relation `parent_child(A, B)`, which can be used to construct a new relation `sibling(U, V) = parent_child(A, U) AND parent_child(A, V)`, which in turn can be used to build another relation `cousin(U, V) = parent_child(X, U) AND parent_child(Y, V) AND sibling(X, Y)`, and so on. However, such systems are known to suffer from the knowledge acquisition problem, which is the inability to leverage unstructured data to derive new computational units and improve existing ones (Kendal and Creen, 2007).

In stark contrast to the symbolic paradigm, modern machine learning models excel at absorbing large amounts of unstructured data and yield strong performance in many challenging domains, ranging from large-scale image classification to language modelling. However, they are relatively rigid in how they share and reuse computation in order to process information: convolutional neural networks, for instance, process the content of an image at every location to arrive at the class label of a given image. In doing so, they only reuse computational units (here, convolutional filters) *laterally* at a constant depth i.e., amongst information being processed simultaneously. In the same spirit, recurrent neural networks only reuse computational units (here, RNN cells) *vertically*, i.e., in depth. Such rigidity in how computation is reused is believed to be one of the reasons current deep neural networks are less capable of systematically generalizing to problems not encountered during training (Bahdanau *et al.*, 2019; Lake and Baroni, 2018; Loula *et al.*, 2018).

In the present work, we draw inspiration from typed programming languages to develop a self-attention based neural architecture that relaxes this rigidity in computation reuse. The resulting class of models, which we call Neural Interpreters, *learns* to flexibly use and compose computational units directly from data without additional supervision. Neural Interpreters factorize a self-attention network (Vaswani *et al.*, 2017) as a system of computational units that we call *functions*. The input to the network is set-valued and processed in the network by a dynamically inferred sequence of functions – potentially by the same function multiple times, enabling vertical sharing of computational units. This aligns with earlier ideas on independent mechanisms (Peters *et al.*, 2017; Parascandolo *et al.*, 2018; Goyal *et al.*, 2019, 2020; Goyal and Bengio, 2020; Schölkopf *et al.*, 2021; Rahaman *et al.*, 2020; Goyal *et al.*, 2021): a set of mechanisms can be combined and reused in many different ways depending on context (the current input or task), thus factorizing knowledge in independent pieces which can lead to better systematic generalization.

Neural Interpreters have two key benefits over vanilla self-attention networks.

- The modular inductive biases facilitate generalization beyond the training distribution and adaptation to new tasks in a sample-efficient manner.

- Second, but consistent with the notion of factorizing knowledge into approximately independent and composable pieces,

The proposed parameterization is (by construction) not only agnostic to the cardinality of the input set, but also to the number of functions. The latter implies that given a new task, additional functions can be non-disruptively added and fine-tuned. In other words, knowledge acquired from the prior training tasks can be effectively repurposed for new tasks. Our primary contributions are as follows.

- We introduce the Neural Interpreter, an attention-based architecture that can be applied on arbitrary set-valued inputs or representations.
- We quantitatively evaluate the proposed architecture in two distinct problem settings: multi-task image classification and abstract reasoning. In the former setting, we show that Neural Interpreters are capable of sample-efficient adaptation to new tasks and can exploit additional capacity added after pre-training. In the latter setting, we demonstrate that Neural Interpreters are capable of out-of-distribution generalization. In particular, we find that Neural Interpreters outperform the Vision Transformer baseline (Cordonnier *et al.*, 2019; Dosovitskiy *et al.*, 2020), which in turn is competitive in-distribution with the prior state-of-the-art. In both settings, we find that Neural Interpreters develop the ability to gracefully trade-off performance with compute at inference time (Zilberstein, 1996). In addition, we include results on a toy problem, where we explicitly probe the ability of Neural Interpreters to learn recomposable computational primitives.
- We ablate over the architectural parameters of Neural Interpreters and qualitatively visualize what the model learns. We find patterns in how the input is routed through the network and that a wide range of hyperparameters yield promising results.

4.2 Related work

Modularity. Despite recent successes of deep neural networks, their ability to recombine and reuse meaningful units for systematic compositional generalization is shown to be limited (Lake and Baroni, 2018; Loula *et al.*, 2018; Keysers *et al.*, 2019). To mitigate this issue, work has been done to modularize deep architectures to enable the reuse and recombination of learned modules (Andreas *et al.*, 2016; Chang *et al.*, 2018; Hudson and Manning, 2018; Rosenbaum *et al.*, 2017; Shazeer *et al.*, 2017; Fernando *et al.*, 2017; Kirsch *et al.*, 2018), and connections have been drawn to the principle of Independent Causal Mechanisms (Schölkopf *et al.*, 2012; Peters *et al.*, 2017; Parascandolo *et al.*, 2018; Goyal *et al.*, 2019; Besserve *et al.*, 2020; Rahaman *et al.*, 2020; Goyal *et al.*, 2020; Lamb *et al.*, 2021; Goyal *et al.*, 2021). Modular architectures vary in how the modules are learned and laid out during inference. For instance, Neural Module Networks

(Andreas *et al.*, 2016) and Neural Event Semantics (Buch *et al.*, 2021) dynamically compose the modules using a natural language parser, whereas neuro-symbolic architectures (Van Steenkiste *et al.*, 2018; Goyal *et al.*, 2019; Mao *et al.*, 2019; Yi *et al.*, 2019; Goyal and Bengio, 2020; Locatello *et al.*, 2020; Goyal *et al.*, 2020; Wu *et al.*, 2020; Goyal *et al.*, 2021) typically require entity-centric representations (Van Steenkiste *et al.*, 2018; Yi *et al.*, 2019; Locatello *et al.*, 2020) but use neural components like attention for compositional reasoning.

Unlike some of these methods, Neural Interpreters (NI) neither require domain knowledge nor object-centric representations. To obtain a strong compositional inductive bias, NIs aim at factorizing the computational graph in terms of functional units which are dynamically recombined in order to solve a particular task. In comparison with SCOFF (Goyal *et al.*, 2020) and NPS (Goyal *et al.*, 2021) which also separate functions, arguments and values and enable a similar dynamic and composed application of functions, NIs introduce the notion of interpreter (which enables parameter sharing between functions and extending the set of functions easily) and sophisticated signature and typing mechanisms described in the previous section.

Another perspective on modularity comes from the notion of *Independent Causal Mechanisms* (Schölkopf *et al.*, 2012; Peters *et al.*, 2017) in causality, which encourages the modules to be independent (Besserve *et al.*, 2020; Parascandolo *et al.*, 2018; Goyal *et al.*, 2019; Rahaman *et al.*, 2020; Lamb *et al.*, 2021). More specifically, modules compete against each other to process certain parts of the input, promoting specializations across time (Goyal *et al.*, 2019), space (Besserve *et al.*, 2020), physical mechanisms (Parascandolo *et al.*, 2018), and time and space (Rahaman *et al.*, 2020; Lamb *et al.*, 2021). Inspired from this idea, functions in NIs also compete against each other to process the input tokens. However, this competition occurs for multiple computational steps (function iterations), thus encouraging specializations across space and compute.

Dynamic routing of information. The systematic generalization of modular architectures is shown to depend on the layout of modules and learning module parameters and their composition in an end-to-end fashion can degrade its generalization performance (Bahdanau *et al.*, 2019; Rosenbaum *et al.*, 2019). Despite these challenges, there exist different frameworks to facilitate the joint learning of module layout and parameters (Sabour *et al.*, 2017; Rosenbaum *et al.*, 2017; Rahaman *et al.*, 2020; Goyal *et al.*, 2019; Fernando *et al.*, 2017; Shazeer *et al.*, 2017; Buch *et al.*, 2021). For example, (Rosenbaum *et al.*, 2017) uses collaborative multi-agent reinforcement learned routing, (Shazeer *et al.*, 2017) uses a trainable gating network. In this work, we use kernel modulated dot product attention (Rahaman *et al.*, 2020) between function signatures and the input types to assign inputs to functions at each step.

Transformers. Modularity in Transformers (Vaswani *et al.*, 2017) is relatively less studied. Repulsive attention (An *et al.*, 2020) is based on repelling transformer heads

to encourage specialization, whereas TIM (Lamb *et al.*, 2021) draws on the principle of independent mechanisms by dividing computation over parallel transformers that interact via competitive top-k attention (Goyal *et al.*, 2019). In contrast, functions in NIs iteratively acquire set elements with kernel dot-product attention, which, if need be, can uniformly activate all the functions for a given token, encouraging efficient use of computation. Closer to Neural Interpreters, Switch Transformers (STs) (Fedus *et al.*, 2021) employ a mixture of experts framework where experts specialize across tokens, guided by a routing scheme which activates only one expert per token. However, STs focus more on dynamically *selecting* experts, whereas Neural Interpreters focus on dynamically *composing* available experts. In contrast, functions in NIs iteratively compete for tokens across multiple computational steps. Moreover, the routing in NIs is not controlled by *top-k* rule, but by kernel modulated dot product attention (Rahaman *et al.*, 2020) which, if need be, can uniformly activate all the experts for a give token, encouraging efficient reuse of the parameters.

The starting point of this work is the vision transformer (ViT) (Cordonnier *et al.*, 2019; Dosovitskiy *et al.*, 2020), which applies a standard Transformer directly to image patches. We use the same architectural scaffolding as ViTs, but implement additional inductive biases that facilitate fast adaptation and systematic generalization. In fact, NIs can be seen as a modular and parameter-efficient scaffolding of ViTs where ViTs are a special case. Similar to ViTs, and unlike above mentioned works, we also use a learnable *[class]* token whose output state serves as the image representation. In this work, we analyze how modularity in ViTs facilitates sample-efficient transfer learning and systematic generalization while maintaining high accuracy.

4.3 Neural Interpreters

In this section, we describe in detail the components of a Neural Interpreter; see Figure 4.1 for an overview. This architecture can be used as a drop-in replacement for a self-attention network, e.g., Transformers (Vaswani *et al.*, 2017). In line with prior work (Cordonnier *et al.*, 2019; Dosovitskiy *et al.*, 2020; Touvron *et al.*, 2021b), we focus on applications to visual data.

Input and output. The input to the Neural Interpreter is any set with vector-valued elements $\{\mathbf{x}_i\}_i, \mathbf{x}_i \in \mathbb{R}^{d_{\text{in}}}$ and the output is another set of vectors $\{\mathbf{y}_j\}_j, \mathbf{y}_j \in \mathbb{R}^{d_{\text{out}}}$ with the same cardinality as the input set. We assume in this work that the input set contains vector embeddings of image patches (Cordonnier *et al.*, 2019) or of entire images (Barrett *et al.*, 2018). The input set additionally includes one or more learned vectors, called CLS Tokens (Devlin *et al.*, 2019), for which the corresponding outputs interface with their respective classifiers (Logeswaran *et al.*, 2020).

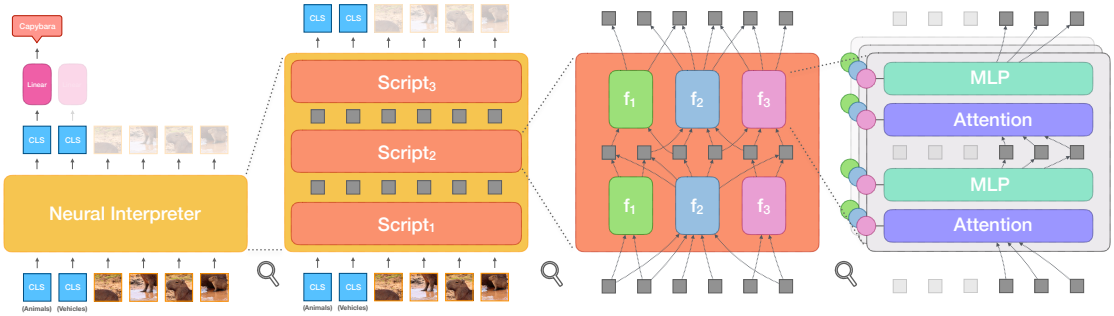


Figure 4.1: **Leftmost:** Overview of the architecture, shown here with image patches as inputs and two CLS tokens with corresponding classification heads. **Center Left:** The Neural Interpreter, shown here as a stack of three *scripts*. **Center Right:** A script, shown here as a collection of three *functions* applied over two *function iterations*. **Rightmost:** A stack of two *Lines of Code* (LOCs), spread over three parallel computational streams (one per function) and conditioned by the respective function codes (colored circles). Residual connections are present but not shown in this figure.

Scripts. A Neural Interpreter can be expressed as a stack of n_s *scripts*, where a script maps one set of vectors to another with the same number of elements:

$$\{\mathbf{y}_j\}_j = \text{NeuralInterpreter}(\{\mathbf{x}_i\}_i) = \left[\text{Script}_{n_s} \circ \ldots \circ (n_s \text{ times}) \circ \ldots \circ \text{Script}_1\right](\{\mathbf{x}_i\}_i) \tag{4.1}$$

A script has four components: a type inference module, a type matching mechanism, a set of functions and an interpreter, as explained below. The parameters of these components are not shared between scripts.
Role: Scripts function as independent building blocks that can be dropped in any set-to-set architecture, and Neural Interpreters with a single script can perform well in practice.

Functions. *Functions* make the computational units in Neural Interpreters; they can be represented as a tuple of vector valued parameters, i.e., $f_u = (\mathbf{s}_u, \mathbf{c}_u)$ where u indexes functions. Here, $\mathbf{s}_u$ is referred to as the signature of the function f_u (with a meaning similar to that in programming languages), and it is a normalized vector of d_{type} dimensions. The signature vector specifies to the type matching mechanism (see below) what inputs are to be routed to f_u. We refer to $\mathbf{c}_u$ as the code vector of f_u, as it instructs an interpreter module (shared between functions, see below) how to process the inputs to f_u.
Role: Functions are vector-valued *instructions* to other components in the script. They implement the computational units that can be reused in the computational graph.

Type matching and inference. The type matching mechanism (Figure 4.2) enables the learned routing of set elements through functions, and it proceeds in three steps. First,

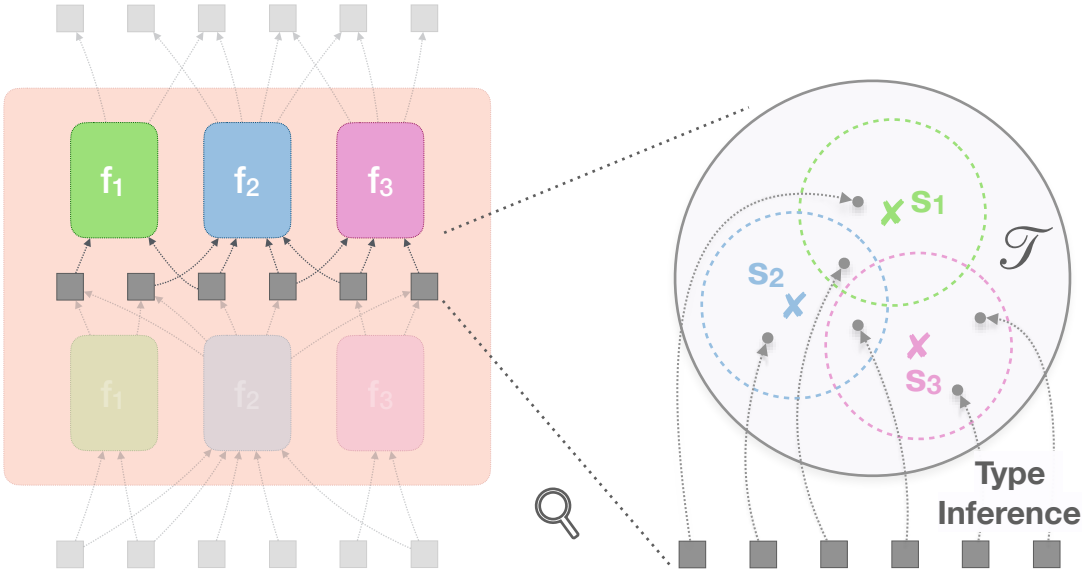


Figure 4.2: Illustration of the type matching mechanism. Functions only access set elements whose types lie in the vicinity of their signatures.

given a set element $\mathbf{x}_i$, it is processed by an MLP module that outputs its *type vector* $\mathbf{t}_i$. This module is called the *type inference* module, and the resulting type vector is an element of the same topological space as function signatures, i.e., a d_{type}-dimensional hypersphere $\mathcal{T}$.

Next, given a function f_u and its signature vector $\mathbf{s}_u \in \mathcal{T}$, the compatibility C_{ui} between the function f_u and a set element $\mathbf{x}_i$ is determined by the cosine similarity between $\mathbf{s}_u$ and $\mathbf{t}_i$ in $\mathcal{T}$ (larger $\implies$ more compatible). Finally, if this compatibility is larger than a threshold (τ), f_u is permitted access to $\mathbf{x}_i$. Formally, let $\{\mathbf{x}_i\}_i$ be a set of intermediate representation vectors (indexed by i). With a learnable parameter σ and a hyper-parameter τ, we have:

$$\mathbf{t}_i = \text{TypeInference}(\mathbf{x}_i) \in \mathcal{T}; \qquad d_{\mathcal{T}}(\mathbf{s}_u, \mathbf{t}_i) = (1 - \mathbf{s}_u \cdot \mathbf{t}_i) \tag{4.2}$$

$$C_{ui} = \frac{\tilde{C}_{ui}}{\varepsilon + \sum_u \tilde{C}_{ui}} \quad \text{where} \quad \tilde{C}_{ui} = \exp\left[-\frac{d_{\mathcal{T}}(\mathbf{s}_u, \mathbf{t}_i)}{\sigma}\right] \text{ if } d_{\mathcal{T}}(\mathbf{s}_u, \mathbf{t}_i) > \tau, \text{ else } 0. \tag{4.3}$$

τ is called the truncation parameter of the kernel and ε is a small scalar for numerical stability. The compatibility matrix $C_{ui} \in [0, 1]$ will serve as a modulation mask (Rahaman *et al.*, 2020) for the self-attention mechanism in the interpreter (see below).

Role: The type matching mechanism is responsible for routing information through functions. The truncation parameter controls the amount of sparsity in routing.

ModLin layers and ModMLP. The components described below make use of linear layers conditioned by some code $\mathbf{c}$. Consider a linear layer with weight matrix $\mathbf{W} \in \mathbb{R}^{d_{\text{out}}} \times \mathbb{R}^{d_{\text{in}}}$ and a bias vector $\mathbf{b} \in \mathbb{R}^{d_{\text{out}}}$. Let $\mathbf{x} \in \mathbb{R}^{d_{\text{in}}}$ denote the input vector to the layer, and $\mathbf{c} \in \mathbb{R}^{d_{\text{cond}}}$ a condition vector, and $\mathbf{W}_c \in \mathbb{R}^{d_{\text{in}}} \times \mathbb{R}^{d_{\text{cond.}}}$ a learnable matrix. The output $\mathbf{y} \in \mathbb{R}^{d_{\text{out}}}$ is given as:

$$\mathbf{y} = \text{ModLin}(\mathbf{x}; \mathbf{c}) = \mathbf{W}(\mathbf{x} \otimes \text{LayerNorm}(W_c \mathbf{c})) + \mathbf{b} \tag{4.4}$$

where $\otimes$ denotes element-wise product and we call the resulting layer a modulated linear layer, or a ModLin layer (Anokhin *et al.*, 2020). Further, one may stack (say) L such ModLin layers (sharing the same condition or code vector $\mathbf{c}$) interspersed with an activation function (we use GELUs (Hendrycks and Gimpel, 2020)) to obtain a ModMLP:

$$\mathbf{y} = \text{ModMLP}(\mathbf{x}; \mathbf{c}) = (\text{ModLin}_L(\cdot\,; \mathbf{c}) \circ \text{Activation} \circ \ldots \circ \text{ModLin}_1(\cdot\,; \mathbf{c}))(\mathbf{x}) \tag{4.5}$$

Role: ModLin layers and the ModMLP can be interpreted as *programmable* neural modules, where the *program* is specified by the condition or code vector $\mathbf{c}$.

ModAttn. ModAttn is a conditional variant of the kernel modulated dot product attention (KMDPA) (Rahaman *et al.*, 2020), where the key, query and value vectors are obtained from ModLin layers conditioned by a vector. In our case, this vector is the code vector $\mathbf{c}_u$ of function f_u, and the corresponding key, query and value vectors are computed as follows (with $\{\mathbf{x}_i\}_i$ as input and h indexing attention heads):

$$\mathbf{k}_{uhi} = \text{ModLin}^h_{\text{key}}(\mathbf{x}_i, \mathbf{c}_u) \qquad \mathbf{q}_{uhi} = \text{ModLin}^h_{\text{query}}(\mathbf{x}_i, \mathbf{c}_u) \qquad \mathbf{v}_{uhi} = \text{ModLin}^h_{\text{value}}(\mathbf{x}_i, \mathbf{c}_u) \tag{4.6}$$

Note that further below (e.g., in Equation (4.10)), we will encounter $\mathbf{x}_{ui}$, where the extra u in the subscript denotes that the set element at index i is specific to the function u; in this case, $\mathbf{x}_i$ is substituted with $\mathbf{x}_{ui}$ in Equation (4.6).

Next, given the keys, queries and the function-variable compatibility matrix C_{ui}, the modulated self-attention weights W_{uhij} are given by:

$$W_{uhij} = \frac{\tilde{W}_{uhij}}{\varepsilon + \tilde{W}_{uhij}} \quad \text{where} \quad \tilde{W}_{uhij} = C_{ui} C_{uj} \left[\text{softmax}_j \left(\frac{\mathbf{q}_{uhi} \cdot \mathbf{k}_{uhj}}{\sqrt{d_{\text{key}}}} \right) \right] \tag{4.7}$$

Here, the quantity W_{uhij} denotes the attention weights in function f_u between elements $\mathbf{x}_i$ and $\mathbf{x}_j$ at head h and the softmax operation normalizes along j; intuitively, information about $\mathbf{x}_i$ and $\mathbf{x}_j$ is mixed by f_u at head h only if $W_{uhij} \neq 0$. Now, on the one hand, this can be the case only if both C_{ui} and C_{uj} are non-zero, i.e., f_u is granted access to both variables $\mathbf{x}_i$ and $\mathbf{x}_j$ by the typing mechanism. But on the other hand, f_u does not necessarily mix $\mathbf{x}_i$ and $\mathbf{x}_j$ even if both C_{ui} and C_{uj} are non-zero, for the self-attention

weights (square brackets) may still be close to zero depending on the context (i.e., the content of $\mathbf{x}_i$ and $\mathbf{x}_j$). Next, the values are linearly mixed using the computed attention weights, which is then processed by a final ModLin layer to yield the output $\mathbf{y}_{ui}$:

$$\mathbf{y}_{ui} = \text{ModLin}(\tilde{\mathbf{y}}_{ui;h}; \mathbf{c}_u) \quad \text{where} \quad \tilde{\mathbf{y}}_{uhi} = \textstyle\sum_j W_{uhij} \mathbf{v}_{uhj} \tag{4.8}$$

Here, $\tilde{\mathbf{y}}_{ui;h}$ means the head-axis is folded into channels.
Role: ModAttn enables interaction between the elements of its input set in multiple parallel streams, one for each function. The query, key, value, and output projectors of each stream are conditioned on the corresponding code vectors, and the interaction between elements in each stream is weighted by their compatibility with the said function.

Line of code (LOC). An LOC layer is a ModAttn layer followed by a ModMLP layer (Figure 4.1, rightmost), where both layers share the same condition vector $\mathbf{c}_u$, and there are weighted residual connections between the layers. Assuming inputs $\{\mathbf{x}_{ui}\}_{u,i}$ to the LOC, we have:

$$\tilde{\mathbf{a}}_{ui} = \text{ModAttn}(\{\text{LayerNorm}(\mathbf{x}_{uj})\}_j; \mathbf{c}_u, \{C_{uj}\}_j) \qquad \mathbf{a}_{ui} = \mathbf{x}_{ui} + C_{ui}\tilde{\mathbf{a}}_{ui} \tag{4.9}$$

$$\tilde{\mathbf{b}}_{ui} = \text{ModMLP}(\text{LayerNorm}(\mathbf{a}_{ui}); \mathbf{c}_u) \qquad \mathbf{y}_{ui} = \mathbf{a}_{ui} + C_{ui}\tilde{\mathbf{b}}_{ui} \tag{4.10}$$

This parameterization ensures that $\mathbf{y}_{ui} = \mathbf{x}_{ui}$ if $C_{ui} = 0$. In words, if a function (f_u) is not granted access to a variable ($\mathbf{x}_i$) by the typing mechanism, it acts as an identity function for this variable. Further, note that we allow the input set to be indexed only by i; in this case, we assume $\mathbf{x}_{ui} = \mathbf{x}_i$ for all u.
Role: A LOC can be thought of as multiple instances of a layer in the original transformer architecture (comprising a self-attention and a MLP module with residual connections), applied in parallel streams, one per function. Computations therein are conditioned on the respective code and signature vectors.

Interpreter. The interpreter layer is a stack of n_l LOCs sharing the same function codes $\mathbf{c}_u$ and function-variable compatibilities C_{ui}. Assuming the input to the interpreter is a set $\{\mathbf{x}_i\}_i$, we have:

$$\mathbf{y}_i = \mathbf{x}_i + \textstyle\sum_u C_{ui} \left(\text{LOC}_{n_l} \circ ... (n_l \text{ times}) ... \circ \text{LOC}_1\right)(\{\mathbf{x}_j\}_j; \mathbf{c}_u, \{C_{uj}\}_j) \tag{4.11}$$

More specifically, the interpreter broadcasts a given set element to multiple parallel computational streams, one for each function. After the streams have processed their copy of the input element, the results are aggregated by a weighted sum over the streams, where the weights correspond to the compatibility of the input element with the respective function. Equation (4.11) can be justified by making two observations. First, if a variable $\mathbf{x}_i$ is not matched with any function by the typing mechanism, it is left unmodified by the interpreter; i.e., if $C_{ui} = 0$ for all u, then $\mathbf{y}_i = \mathbf{x}_i$. This allows signals to be propagated

through the interpreter without interference from existing functions, if so determined by the type inference module. Second, the additive aggregation over the function index u implies that the overall parameterization of Neural Interpreters does not depend on the number of functions. This allows one to add a new function f_v simply by including its signature and code $(\mathbf{s}_v, \mathbf{c}_v)$ as learnable parameters and finetuning these on (say) a new problem.
Role: The interpreter serves as a general-purpose instruction executor (one that is shared between functions). Given a set of inputs and an instruction (here, the function code), it executes said instruction to compute the output.

Function iterations in script. Having expressed the overall model as a stack of multiple scripts, we are now equipped to describe the computational graph of a single script. A script can be expressed as a recurrent application of the type matching mechanism and the interpreter, where we refer to the composition of the latter two as a Function Iteration (FnIter):

$$\{\mathbf{y}_i\}_i = \text{FnIter}(\{\mathbf{x}_j\}_j) = \text{Interpreter}(\{\mathbf{x}_j\}_j, \{\mathbf{c}_u\}_u, \{C_{uj}\}_{u,j}) \tag{4.12}$$

$$\text{where } C_{uj} = \text{TypeMatching}(\mathbf{s}_u, \mathbf{x}_j) \tag{4.13}$$

Here, the TypeMatching component encapsulates both type inference and kernel matching, as detailed in Equation (4.3).

A script (cf. Equation (4.1)) can now be expressed as a recurrent application of FnIter:

$$\{\mathbf{y}_j\}_j = (\text{FnIter} \circ \ldots \circ (n_i \text{ times}) \circ \ldots \circ \text{FnIter})(\{\mathbf{x}_i\}_i) \tag{4.14}$$

Role: Inside a script, function iterations enable sharing of computational units in depth. Increasing the number of function iterations can increase depth without increasing the number of parameters.

Preventing function signatures and variable types from collapsing on each other. One might obtain a scenario where the signatures of all functions and the types of all possible set elements collapse to a single point in type-space. This causes all set elements to be routed to all functions, thereby undermining the inductive bias of modularity. One effective way of preventing this degeneracy is to keep the function signatures fixed at initialization (i.e. a high-entropy distribution). This effectively encourages the (learnable) type-inference module to produce diverse types, in order to use all available functions.

In summary, we observe that any input element $\mathbf{x}_i$ may trace a large number of *computational paths* as it progresses through the model, where a computational path is a partially ordered set (poset) of functions (refer to Figure 4.6 for a visualization of such computational paths). In particular, owing to the fact that we recurrently apply function iterations in scripts, this poset may contain repeated functions, enabling weight sharing in depth. Further, the use of an interpreter that is shared between functions but explicitly

conditioned (or *programmed*) by their code vector allows us to remain independent of the number of functions and retain the ability to add more functions after the model has been trained. Future work may investigate manipulating the code vectors themselves, thereby enabling higher-order functions (i.e., functions that manipulate other functions).

4.4 Experiments

In this section, we empirically evaluate Neural Interpreters in a variety of problem settings. The questions we seek to answer are the following. **(a)** Can Neural Interpreters learn reusable computational units, given a number of training tasks that can in principle be solved with a fixed set of primitives? (Section 4.4.1) **(b)** Do Neural Interpreters modularize information in a way that helps fast adaptation (Bengio *et al.*, 2019)? (Section 4.4.2) **(c)** Can Neural Interpreters gracefully handle a larger number of functions (computational units) than they were trained with? (Section 4.4.2) **(d)** Do the inductive biases help with systematic generalization required in abstract reasoning problems? (Section 4.4.3).

4.4.1 Learning fuzzy boolean expressions

In this section, we construct a toy problem to investigate whether neural interpreters can learn reusable functions when trained on a large number of tasks that share the same underlying building blocks.

Task definition. Consider a set of N scalars $\{x_1, ..., x_N\}$, where $x_i \in [0,1]$ is a real number between 0 and 1 (inclusive). We now define the following operations on the elements of this set:

$$\texttt{and}(x_i, x_j) = x_i x_j; \quad \texttt{not}(x_i) = \bar{x}_i = 1 - x_i; \quad \texttt{or}(x_i, x_j) = x_i \oplus x_j = \overline{\bar{x}_i \bar{x}_j} \tag{4.15}$$

Note that the above operations map from $[0,1]^2$ to $[0,1]$; if $x_i, x_j \in \{0,1\}$, they reduce to their Boolean namesakes. By combining these primitives, it is possible to construct and sample from a family of 2^{2N} *fuzzy* Boolean functions mapping from $[0,1]^N$ to $[0,1]$ as shown in Figure 4.3. See more details on the dataset construction in Appendix C.1.

The problem is now set-up as follows. We sample 30 random fuzzy Boolean functions of $N = 5$ variables, $\{f_i\}_{i=1}^{30}$ where $f_i : [0,1]^5 \to [0,1]$, of which we use 20 for the pre-training phase and reserve 10 for the later adaptation phase. For each function, we sample 163840 points from a uniform distribution over $[0,1]^5$, of which we use 80% for training, and the remaining for validation. This yields two multi-task regression datasets, one for pre-training and the other for adaptation. A sample from the pre-training dataset comprises a 5D input vector $\mathbf{x} \in [0,1]^5$ and the scalar regression targets $f_1(\mathbf{x}), ..., f_{20}(\mathbf{x})$.

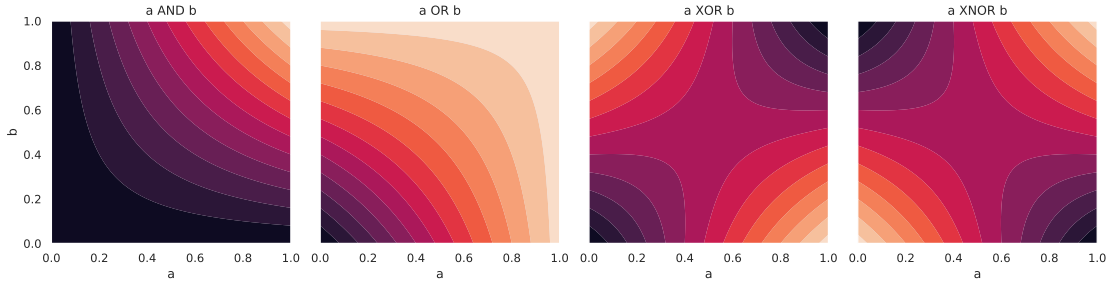


Figure 4.3: Visualization of fuzzy relaxations of binary operations mapping $a \in [0,1]$ and $b \in [0,1]$ to a value in $[0,1]$. From left to right: `and`, `or`, `xor` and `xnor`.

Method. We pre-train a Neural Interpreter to regress all 20 functions simultaneously. To do this, we feed it a set of 25 elements as input: the first 5 are the components of $\mathbf{x}$ (with learnable positional embeddings (Radford *et al.*, 2019) added in) and the remaining 20 are learned CLS tokens (vectors), one for each function. We attach a (shared) regression head to the output elements corresponding to the CLS tokens, and train their outputs to regress the respective function. Having pre-trained the model for 20 epochs, we finetune it for an additional 3 epochs on the 10 reserved functions $\{f_{21}, ..., f_{30}\}$. For the latter, we always instantiate and train 10 new CLS tokens, corresponding to the new functions; however, we investigate three settings in which three different sets of parameters remain frozen. In the first setting, all parameters remain frozen, implying that the CLS tokens are the only trainable parameters. In the second setting, we unfreeze the parameters of the type matching mechanism (function signatures and parameters of the type inference MLP, in addition the CLS tokens). In the third setting, we unfreeze all parameters and finetune the entire model. Additional details in Appendix C.1.

Hypothesis. By finetuning just the type-matching parameters, we only permit adaptation in how information is routed through the network. In other words, we only allow the computational units to *rewire* themselves in order to adapt to the new task at hand, while preserving their functionality. Now if the computational primitives that are learned during pre-training are recomposable, one would expect the performance of Neural Interpreters having finetuned just the type-matching parameters to approach that obtained by finetuning all the parameters, the latter including ones that determine the functionality of the computational primitives.

Results. Section 4.4.1 compares the coefficients of determination[1] (R^2) obtained in each of the investigated finetuning settings. We find that relative to finetuning just the CLS tokens, the performance difference between finetuning all parameters and just the

[1] $R^2 = 1$ implies perfect fit; a model that regresses to the mean has $R^2 = 0$.

Parameter Set	$\mathbf{R^2}$
All Parameters (Pretraining)	0.9983 ± 0.0005
Finetuning CLS Tokens	0.9202 ± 0.0198
+ Type Inference Parameters	0.9857 ± 0.0034
+ Remaining Parameters	0.9953 ± 0.0013

Table 4.1: Mean Coefficient of Determination (R^2) and StdDev over 10 tasks after training various sets of parameters. **Gist:** By just allowing the functions to *rewire* themselves by training only the type-matching parameters, one approaches the performance of tuning all remaining parameters, including those of the functions themselves. This suggests that the functions have learned recomposable primitives.

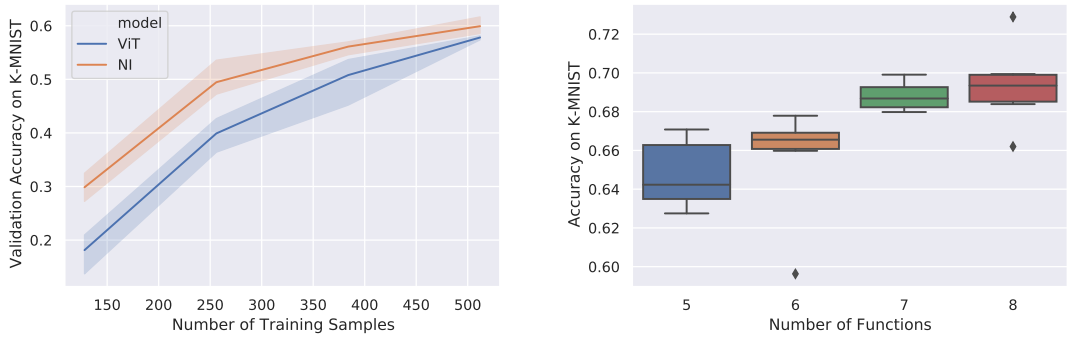
type matching parameters is small. This is in line with expectation, suggesting that Neural Interpreters are indeed capable of learning recomposable computational primitives.

4.4.2 Multi-task image classification

In this section, we evaluate Neural Interpreters in a multi-task image classification setting. Our goals are (a) to determine whether the inductive bias helps with fast adaptation, (b) to investigate whether the interpreter can indeed function as a general instruction executor, as intuited in Section 4.3, and (c) to demonstrate that the proposed architecture produces modules that can function autonomously (without additional training objectives that encourage this behaviour). Additional results in Appendix C.2 analyze the effects of varying hyper-parameters.

Task definition. We consider three related datasets sharing the same label semantics, namely SVHN (Netzer *et al.*, 2011), MNISTM (Ganin *et al.*, 2016) and MNIST (LeCun *et al.*, 2010). The images therein are upsampled to shape 32×32 (if required), and the resulting image is augmented with RandAugment (Cubuk *et al.*, 2019). Subsequently, the augmented images are split (Cordonnier *et al.*, 2019) to 64 patches of shape 4×4. The combined training set has 193257 samples, whereas the validation set has 46032 samples. In addition, we also use unaugmented samples from the K-MNIST dataset (Clanuwat *et al.*, 2018) of Hiragana characters to probe fast-adaptation to new data.

Method. We train Neural Interpreters for 100 epochs on the combined *digits* dataset described above. The input set contains 67 vector valued elements – the first 64 corresponding to linear embeddings of the 4×4 patches, and the remaining 3 to learnable CLS tokens (vectors), one for each dataset. For each CLS token, a linear classification head is attached to the corresponding output; given an input sample from a certain dataset, the respective classification head is trained to predict the correct class.



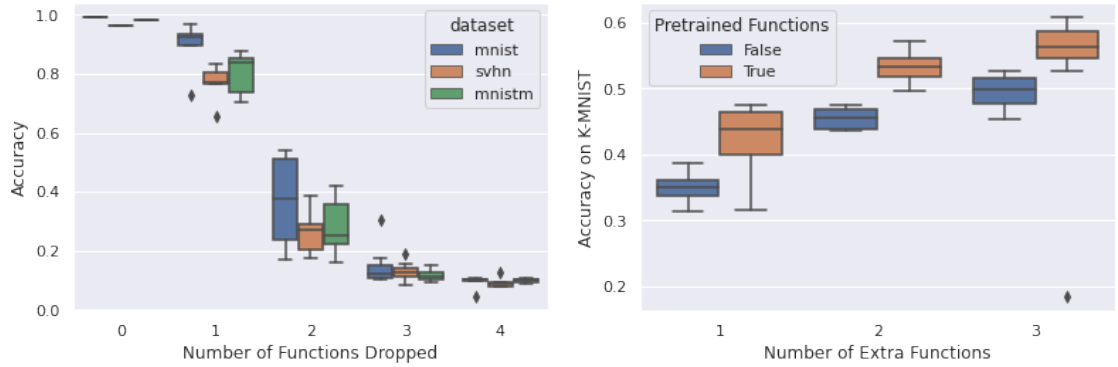
(a) Fast adaptation (b) Capacity extension

Figure 4.4: ***Figure 4.4a:*** Performance of Neural Interpreters (NI) compared to that of a Vision Transformer (ViT) that performs equally well on the validation set (y-axis), plotted against the number of training samples presented (x-axis). We see Neural Interpreters can adapt faster in the low-data regime. ***Figure 4.4b:*** Validation performance (y-axis) of a Neural Interpreter trained with 5 functions but finetuned with various number of functions (x-axis). We see that performance increases with increasing functions, showing that the model does not *overfit* to the number of functions it was trained with.

We chose to share the parameters in the classification heads and the CLS tokens, since the alternative did not yield a significant difference in performance. To inject position information into the model, we use a variant of the relative positional encoding scheme described in (Cordonnier *et al.*, 2019), but applied only to the first 64 input elements (corresponding to the image patches). Having pre-trained on the digits dataset, we finetune the model on K-MNIST with varying numbers of samples for 10 epochs. For additional details, please refer to Appendix C.2.

Baseline. Vision Transformers (ViT) (Cordonnier *et al.*, 2019; Dosovitskiy *et al.*, 2020) make the natural baseline for Neural Interpreters, given the fact that the former is a special case of the latter. The set-up with CLS tokens and classification heads is identical to that of Neural Interpreters, as is the training protocol. We use a light-weight 8-layer deep model with approximately 1.8M parameters, but ensure that the considered Neural Interpreter model is even lighter, with roughly 0.6M parameters. We train it under identical conditions as Neural Interpreters, and use the hyper-parameter configuration of the DeiT-Tiny model (Touvron *et al.*, 2021b), and ensure that the Neural Interpreter models we experiment with have yet fewer parameters.

Hypotheses. **(1)** It has been suggested that a model that appropriately modularizes the learning problem should excel at fast-adaptation to a new but related data distribution (Bengio *et al.*, 2019). If Neural Interpreters obtain a useful modularization of the prob-



(a) Dropped functions

(b) Increasing functions

Figure 4.5: ***Figure 4.5a:*** Accuracy on the digits dataset as a function of the number of dropped functions. We find that the performance degrades gracefully as functions are dropped, suggesting that learned functions are autonomous, i.e. they can operate in the absence of other functions. ***Figure 4.5b*** Performance on K-MNIST after adaptation. In one case (orange), functions pretrained from the digits dataset are kept intact. In another case (blue), function codes and signatures are randomly sampled before adaptation. We find that the former performs better. **Gist:** Knowledge acquired during pretraining is indeed reused during adaptation. This is apparent when comparing with a baseline where this knowledge is destroyed, leading to decreased performance after adaptation.

lem, we should expect it to be able to adapt faster to the K-MNIST dataset than a non-modular baseline like the ViT. This is because some of the visual primitives present in the digit datasets (e.g., motifs, strokes or shapes) might reoccur in the K-MNIST dataset, which Neural Interpreters should be able to leverage. **(2)** Recall that in Section 4.3, we positioned the interpreter as a general purpose instruction executor, capable of being programmed by a code vector. If this is indeed the case, and if the interpreter does not overfit to the functions it was trained with, we should expect the capacity (and therefore performance) to increase as we add and train new functions (i.e., tuples of code and signatures), while keeping all other parameters fixed. **(3)** If functions (modules) are indeed autonomous, we should expect that as some of them are removed at test time, the others are still able to maintain performance. Consequently, the accuracy of the Neural Interpreter (trained with all functions) should only degrade gracefully as functions are (randomly) removed at test time.

Results. Figure 4.4a compares the fast-adaptation performance of a Vision Transformer to that of a Neural Interpreter. While both models achieve almost identical performance on the validation set for all datasets (the difference being less than 0.2% in favor of Vision Transformers), we find that Neural Interpreters significantly outperform Vision Transformers at fast-adaptation in the low-data regime, and the performance gap is only

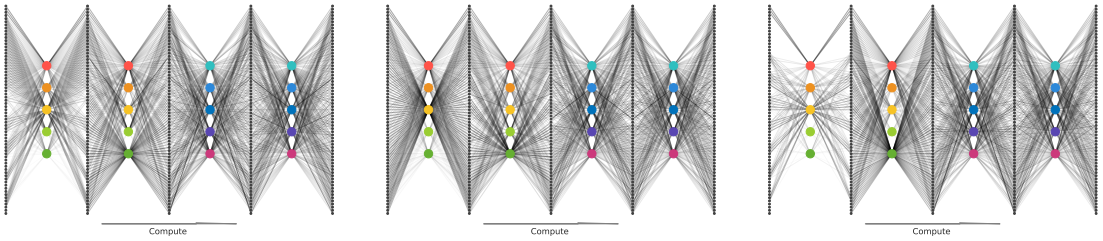


Figure 4.6: Visualization of computational paths taken by input set elements corresponding to three different samples as they progress through a Neural Interpreter. Colored dots identify functions (same color implies shared parameters), and the weight of the lines denote their compatibility with set elements. **Gist:** There are variations (but also similarities) between samples in how their constituent set elements are routed through the network.

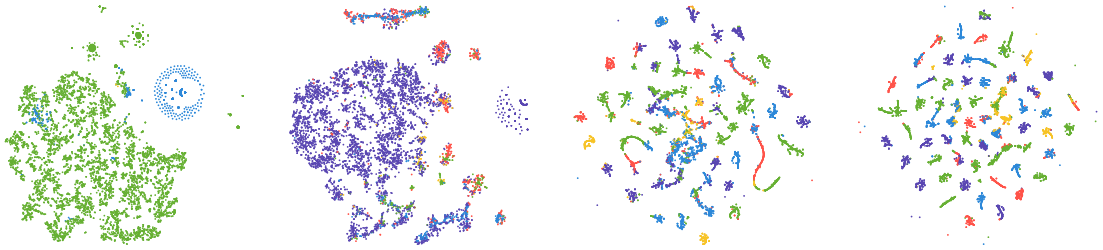


Figure 4.7: t-SNE embeddings of the inferred types of set elements as they progress through a Neural Interpreter with two scripts with two function iterations each. The color identifies the closest function in type space, and the progression from left to right is over the function iterations. **Gist:** Types are more clustered in the later function iterations, suggesting that the input set elements gradually *develop* a type as they progress through the network.

gradually closed as more data becomes available. This observation supports hypothesis **(1)**. Further, in Figure 4.5b, we establish a baseline where the functions pretrained on the digits dataset are destroyed prior to adaptation. We find that keeping the functions acquired from pretraining intact leads to significantly improved adaptation performance, suggesting that knowledge acquired during pretraining is indeed being reused during adaptation. This lends further support to hypothesis **(1)**.

Figure 4.4b shows the validation performance of a Neural Interpreter that was pretrained with 5 functions on the digits dataset, but tested on the K-MNIST dataset with varying numbers of functions, having finetuned just the function signatures and codes. We find that the performance improves as new functions are added at adaptation-time, supporting hypothesis **(2)**. Figure 4.5a shows the effect of randomly removing functions on the model accuracy. We observe that the performance on all datasets degrades gracefully as more functions are removed, thereby supporting hypothesis **(3)**.

In addition, Figure 4.6 visualizes the computational path taken by input set elements as they progress through the network, verifying that there is diversity in how samples are routed through the network, and shows that the routing mechanism discriminates between samples. Figure 4.7 shows that the input set elements gradually *develop* a type as they progress through the model. Additional figures in Appendix C.2 analyze the effect of varying the number of scripts, number of function iterations, number of functions, kernel truncation parameter, dimension of type space $\mathcal{T}$ and freezing the function codes / signatures on the validation performance. The key finding is that, on the one hand, a wide range of hyper-parameter settings leads to performant models; on the other hand, there are patterns in what hyper-parameters perform consistently well.

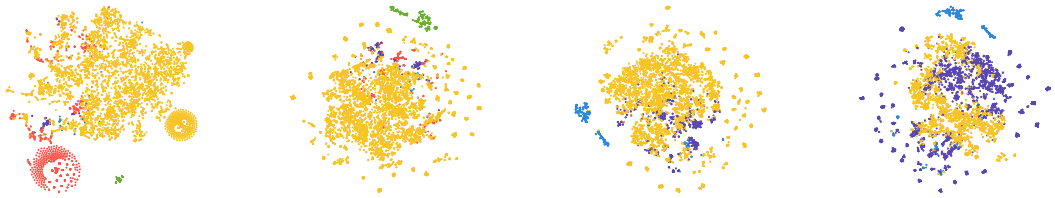


Figure 4.8: Same plot as Figure 4.7, but with routing fixed at initialization. **Gist:** Inferred types of set elements exhibit less structure and diversity at initialization, especially at later function iterations. This suggests that the learning process indeed induces non-trivial patterns in how information is routed through the network.

Diversity in routing mechanism We further investigate whether the learned routing in neural interpreters is meaningfully diverse i.e. whether certain samples get routed through certain functions? To answer it, we visualize the t-SNE embeddings of the variable types in Figure 4.7. The color-codes represent the close affinities between variables and certain functions in type space. We compare it against the case where the routing is fixed at initialization Figure 4.8. It can be seen that in the randomly initialized routing the type-function assignments (given by the colors assigned to a dot) exhibit less structure and diversity, especially at the later function iterations. This suggests that the learning process in neural interpreters indeed induces non-trivial patterns in how information is routed between modules.

4.4.3 Abstract reasoning

In this section, our goal is to (a) use visual abstract reasoning tasks to evaluate whether Neural Interpreters are capable of systematic (compositional) generalization, and (b) characterize how Neural Interpreters maintain performance when the amount of compute is reduced at test time.

Task definition. Progressively Generated Matrices (PGM) (Barrett *et al.*, 2018) is a procedurally generated series of datasets based on Raven Progressive Matrices (RPMs) (Raven and Court, 1998), a popular IQ test designed for humans. The datasets, each comprising around 1.2M samples, aim to assess abstract and analogical reasoning capabilities of machine learning models beyond surface statistical regularities (Jo and Bengio, 2017; Rahaman *et al.*, 2019). The tasks in PGMs mimic the popular human IQ tests, Raven Progressive Matrices (RPMs) (Raven and Court, 1998). Each sample in a dataset (Figure 4.9) comprises 8 context panels arranged in an incomplete 3×3 matrix, and 8 candidate answer panels (a panel being an 80×80 image).

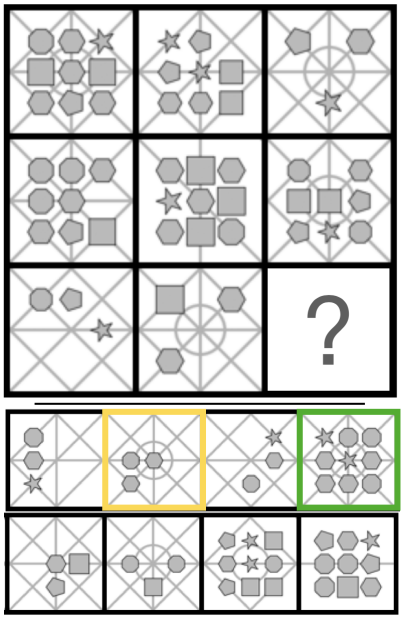


Figure 4.9: A PGM task. **Top:** Context panels. **Bottom:** Candidate panels. WReN and ViT predict the wrong answer (yellow), whereas NI predicts the correct one (green).

Given access to the context panels, the model must predict which of the candidate images is the most appropriate choice to complete the matrix. The content of the panels in a matrix are related by one or more *triples*, where a triple comprises a logical rule ($\in$ progression, XOR, OR, AND, consistent union) applied to an attribute ($\in$ size, type, color, position, number) of an object of a type ($\in$ shapes, lines). There are 8 datasets in the series, of which 7 measure systematic generalization in different ways, i.e., the test datasets contain samples where the panels are related in ways that are not encountered during training. In this work, we consider 6 datasets that probe the compositional reasoning ability of the model, namely: Interpolation, Extrapolation, Held-out (H.O.) triples, H.O. pairs of triples, H.O. Attribute Pairs, and Neutral. We omit H.O. line-type and H.O. shape-type datasets, since they stress the convolutional feature extraction instead of compositional reasoning component of the model. Please refer to Appendix C.3 and (Barrett *et al.*, 2018) for additional details.

Method. Each panel (context and choice) is embedded with a shallow convolutional neural network to obtain an embedding vector. The input to the model is a set of 10 elements, comprising the embeddings of 8 context panels, that of a single candidate panel, and a CLS token (learnable vector). The model output corresponding to the CLS token is fed as input to the prediction head, which is trained to output a score measuring the compatibility of the candidate panel to the context panels. The final prediction is obtained by applying a softmax over the scores of all candidate panels. We note that this set-up resembles the one proposed in (Barrett *et al.*, 2018) and refer to Appendix C.3 for details.

Baselines. We train Vision Transformers (Cordonnier *et al.*, 2019; Dosovitskiy *et al.*, 2020) and Neural Interpreters with the same architectural scaffolding (described above). Additional baselines include Wild Relational Networks (WReN) (Barrett *et al.*, 2018), Multiplex Graph Networks (MXGNet) (Wang *et al.*, 2020) and WReNs atop disentangled representations (VAE-WReN) (Steenbrugge *et al.*, 2018). Regarding the latter, we note that disentangled representations only improves on the feature extraction component of the WReN model; as such, it is entirely complementary to our contribution, which is the abstract reasoning component. Future work may explore replacing the convolutional embeddings with disentangled representations.

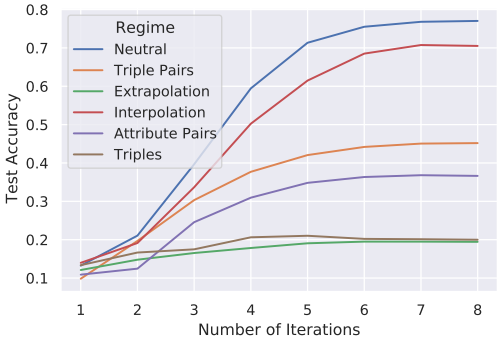


Figure 4.10: Test performance of a Neural Interpreter as a function of the number of function iterations (i.e. the amount of inference-time compute). **Gist:** Neural Interpreters are capable of trading-off performance with runtime compute.

Hypothesis. Recall that the test sets of (all but one) PGM datasets contain samples where the panels are not related in a way that is encountered during training. A model that is able to infer and reason about the underlying compositional structures is therefore more likely to be successful. If the inductive biases do indeed enable Neural Interpreters to factorize knowledge in to recomposable primitives, we might expect them to be able to dynamically recombine these primitives in order to generalize to problems it has not encountered during training and thereby excel at this task.

Results. Table 4.2 tabulates the validation (in-distribution) and test (out-of-distribution) accuracies obtained by all models for the various considered datasets. We make two observations: (1) Neural Interpreters are competitive in terms of test accuracy, outperforming both Vision Transformers and the prior state-of-the art in 4 of 6 datasets. (2) Vision Transformers are competitive in-distribution: they outperform the state of the art (excluding Neural Interpreters) in 5 of 6 datasets in terms of validation accuracy. However, they are outperformed in terms of test accuracy by either Neural Interpreters or other baselines in all datasets. In addition, Figure 4.10 shows how the test-set performance varies with the number of function iterations, for a model that was trained with a fixed number of function iterations (8). We note that the number of function iterations is proportional to the amount of computation used by the model. In all cases, the model maintains at least 70% of its original performance while using only half the amount of compute; in most cases, at least 80% of the original performance is maintained. This suggests that Neural Interpreters can function as anytime predictors (Zilberstein, 1996), which are models that

can trade-off performance with compute. This also suggests that the inductive bias of recurrent computation (as implemented in Equation (4.14)) indeed manifests as a useful property of the model, as in NIs can function as an anytime predictor.

Regime	Neutral		Interp.		Attribute P.		Triple P.		Triples		Extra.	
Model	**Val.**	**Test**	**Val.**	**Test**	**Val.**	**Test**	**Val.**	**Test**	**Val.**	**Test**	**Val.**	**Test**
WReN	63.0	62.6	79.0	64.4	46.7	27.2	63.9	41.9	63.4	19.0	69.3	17.2
V-WReN	64.8	64.2	-	-	**70.1**	**36.8**	64.6	43.6	59.5	**24.6**	-	-
MXGNet	67.1	66.7	74.2	65.4	68.3	33.6	67.1	43.3	63.7	19.9	69.1	18.9
ViT	73.3	72.7	**89.9**	67.7	69.4	34.1	67.6	44.1	73.8	15.9	**92.2**	16.4
NI (ours)	**77.3**	**77.0**	87.9	**70.5**	69.5	36.6	**68.6**	**45.2**	**79.9**	20.0	91.8	**19.4**

Table 4.2: Performance (prediction accuracy) of all models on different generalization regimes of PGM datasets. Note that the *Val.* performance measures in-distribution generalization, and the *Test* performance measures out-of-distribution generalization on the corresponding datasets (except for Neutral). Here, *V-WReN* refers to VAE-WReN, *Extra.* refers to Extrapolation, *Interp.* to Interpolation, *Attribute P.* to Attribute Pairs and *Triple P.* to Triple Pairs.

4.5 Configuring Neural Interpreters

Neural Interpeters introduce a number of components that are not present in Vision Transformers, and accordingly, it introduces additional hyperparameters. While we found a large range of hyperparameters can work well in practice, there are patterns that warrant discussion. In this section, we provide a detailed discussion of these hyperparameters, and how these are set in this work. We also remark that there might be other hyperparameter settings that work well for different settings, and the insights in this section should merely function as a guide.

Partitioning depth between LOCs, Function iterations and Script

There are three distinct ways of increasing the depth of Neural Interpreters.

1. **Increasing the number of Function iterations.** Increasing n_i, the number of function iterations, is a natural way of increasing the depth of Neural Interpreters *without* increasing the number of parameters. Large n_i encourages the recursive and iterative reuse of computation (Dehghani *et al.*, 2019), but might result in a scarcity of parameters.

2. **Increasing the number of Scripts.** Increasing n_s, the number of scripts, is a way of increasing the depth of Neural Interpreters *while* increasing the number of parameters. Larger n_s tends to result in models that are easier to train, potentially due

to a larger number of good solutions in the parameter space (Choromanska *et al.*, 2015) (owing to the larger number of parameters). However, if n_s is increased at the price of decreasing n_i, one might forego some inductive bias towards iterative reuse of computation.

3. **Increasing the number of LOCs.** Increasing n_l, the number of LOCs, also increases the depth *while* increasing the number of parameters. However, unlike increasing n_s, increasing n_l results in a deeper block of layers being recursively applied. Increasing n_l might come at the price of decreasing n_i, in which case some recursive inductive bias is foregone; or it might come at the price of decreasing n_s, which might result in models that are less consistent.

Recommendation. If training is less stable or in-distribution performance is important, one should consider increasing the number of scripts n_s. If the training is stable but out-of-distribution generalization or fast-adaptation performance is important for the application, one should consider increasing the number of function iterations n_i. If there is additional budget for hyper-parameter search, one could consider tuning the number of LOCs (starting with $n_l \in \{1,2\}$).

Increasing the number of Functions

Increasing n_f, the number of functions, is a parameter efficient way of increasing the width of the network in a model-parallelizable way. This is especially apparent from Equation (4.10), where the index over functions (u) can be effectively folded in to the batch-axis. Further, we found benefits in increasing the number of functions (also in terms of in-distribution performance), suggesting that the distribution of parameters between interpreter and the codes (as described in Equation (4.4)) is scalable.

Recommendation. The number of functions can be safely increased to match available resource capacity.

Kernel truncation parameter and dimension of type space

These hyperparameters (inherited from (Rahaman *et al.*, 2020)) have to do with routing of information through the network. The truncation parameter $\tau \in [0,2)$ controls the hardness of the routing – if τ is small, functions are only granted access to set elements whose types lie in the immediate vicinity of their signatures. For larger τ, functions may be granted access to set elements whose types are less similar to their signatures in type-space, albeit the said elements are down-weighted by the kernel. The type space dimension d_{type} controls the amount of flexibility afforded to the routing mechanism. Intuitively, larger d_{type} implies that there are more ways to how the signature and type

vectors can be positioned in the type space $\mathcal{T}$ (a hypersphere of dimension d_{type}) relative to each other.

Recommendation. These hyperparameters may vary with the problem at hand. If sparsity is desired, one should consider lower values for τ. If training is less stable, larger values of τ might mitigate the issue. We find $\tau \in [1.2, 1.7]$ to be a reasonable range for hyperparameter sweeps. As for d_{type}, we find all values between 20 and 50 to work well in our experiments.

Learning Function signatures and code

When pre-training the model, one decision that must be made is whether or not the function signatures and codes should be trained. Note that freezing these parameters at the pre-training stage does not necessarily constrain the model in a significant way – if the function signatures are fixed, the type-inference MLP can adapt (Equation (4.3)); likewise, if function codes are frozen, the weight matrices W_c (Equation (4.10)) can adapt. Note that this applies in the pre-training phase, where the type inference MLP and the interpreter parameters are allowed to adapt.

Recommendation. While we did not find a large difference, runs with frozen function codes were slightly less consistent than the ones with learned function codes. At the same time, runs with frozen function signatures tended to perform at least as well as the ones that learned function signatures, if not slightly better.[2]

Choice of an optimizer and scheduler

Like for most self-attention based models (including the transformer (Vaswani *et al.*, 2017)), the choice of an optimizer and learning rate schedule plays an important role. A common practice is to use Adam with a linear learning rate warm-up and cosine annealing (once per optimization step). However, learning rate warm-up is known to be a heuristic to control the variance of Adam learning rate in the early stages of training, a problem that Rectified Adam (Liu *et al.*, 2019) (RAdam) solves in a more principled way while eliminating a sensitive hyperparameter (the number of warm-up steps). Further, for certain adaptation tasks where the loss-landscape can potentially be challenging, we found Shampoo (Gupta *et al.*, 2018) to work particularly well.

Recommendation. For pre-training Neural Interpreters, we can recommend the RAdam optimizer with a cosine annealing schedule (without warm-up). We anneal the learning rate by roughly two orders of magnitude over 80-90% of the training steps, and keep the

[2] This is less surprising in light of the fact that the type-inference MLP has a larger number of parameters that can be adapted during training.

learning rate at the minimum for the remainder of the steps. While we found RAdam to also work well for most finetuning experiments, Shampoo (Gupta *et al.*, 2018) with appropriately tuned learning rate can serve as a reasonable alternative in the event that RAdam does not perform as expected.

4.6 Conclusion

We have presented Neural Interpreters, a self-attention based architecture capable of learning a system of recomposable computational primitives. Neural Interpreters relax the rigidity in how computation is reused in current neural architectures, and our experiments show that the modular inductive biases it incorporates facilitate systematic generalization and sample-efficient adaptation to new tasks.

There are multiple exciting avenues of future research. One line of work could leverage the capacity extension capability of Neural Interpreters in a continual learning setting, thereby enabling *continuous integration* of knowledge in a neural model. Another promising direction entails using Neural Interpreters as a backbone for learning world models, where systematic generalization and out-of-distribution robustness are of paramount importance.

Chapter 5

Learning Transferable Meta-Representations of Functions

"Learning is a treasure that will follow its owner everywhere."

— Chinese Proverb

Meta-learning algorithms adapt quickly to new tasks that are drawn from the same task distribution as the training tasks. The mechanism leading to fast adaptation is the conditioning of a downstream predictive model on the inferred representation of the task's underlying data generative process, or *function*. This *meta-representation*, which is computed from a few observed examples of the underlying function, is learned jointly with the predictive model. In this work, we study the implications of this joint training on the transferability of the meta-representations. Our goal is to learn meta-representations that are robust to noise in the data and facilitate solving a wide range of downstream tasks that share the same underlying functions. To this end, we propose a decoupled encoder-decoder approach to supervised meta-learning, where the encoder is trained with a contrastive objective to find a good representation of the underlying function. In particular, our training scheme is driven by the self-supervision signal indicating whether two sets of examples stem from the same function. Our experiments on a number of synthetic and real-world datasets show that the representations we obtain outperform strong baselines in terms of downstream performance and noise robustness, even when these baselines are trained in an end-to-end manner.

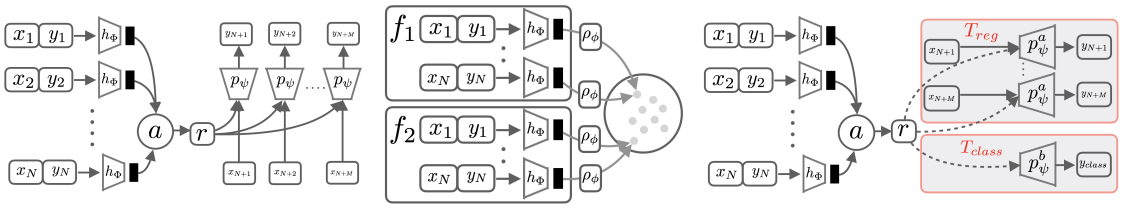
Based on (Gondal *et al.*, 2021). Muhammad Waleed Gondal, Shruti Joshi, Nasim Rahaman, Stefan Bauer, Manuel Wüthrich, Bernhard Schölkopf. "Function Contrastive Learning of Transferable Meta-representations." In: International Conference on Machine Learning (ICML), 2021.

5.1 Introduction

Many supervised learning problems are concerned with approximating a data-generating function $f : \mathcal{X} \to \mathcal{Y}$ given a finite set of N samples, $\{x_i, y_i = f(x_i)\}_{i=1}^{N}$. Expressive models, such as deep neural networks, are known to excel at this function approximation task, but they often heavily rely on the number of samples N being large. This poses further challenges: in many domains of interest, sourcing enough data is a challenging endeavour; further, the process of training such models can be prohibitively slow for many applications. This is exacerbated by the fact that in the typical setting, each new data-generating function encountered requires that the model be retrained. In other words, the model is not shared between data-generating functions, even when training a model to approximate one function can potentially be beneficial for approximating another function.

To overcome these challenges, a variety of meta-learning methods have been proposed (Vinyals *et al.*, 2016; Snell *et al.*, 2017; Garnelo *et al.*, 2018a; Ravi and Larochelle, 2016; Finn *et al.*, 2017). In the present work, we are interested in a class of models that use encoder-decoder architectures such as Conditional Neural Processes (CNPs) (Garnelo *et al.*, 2018a) and Generative Query Networks (GQNs) (Eslami *et al.*, 2018). In the first stage, an encoder is used to infer a fixed-dimensional representation of a given function f from just a few input-output examples $O^f = \{(x_i, y_i)\}_i$, *the context dataset*. We call it the *meta-representation* of the function $r = h_\phi(O^f)$, where h is an encoder parameterized by ϕ. In the second stage, the meta-representation is then used to condition a predictive model in order to solve a downstream prediction task related to that function. For instance, the task may consist of predicting the function value y at unseen locations x or classifying images after observing only a few pixels (in that case, x is the pixel location and y is the pixel value). This two-stage process has multiple benefits. First, the extraction of prior knowledge about f directly from the training data, in the form of meta-representation, reduces the need for specifying inductive biases (model architectures, training details, etc.) particular to f. Thus, it allows learning to be shared between functions such that a single model can be trained on a distribution over functions. Second, the computation of meta-representations is efficient and can be done online. Third, the computation of meta-representations provides flexibility to solve a variety of downstream tasks concerning a specific function.

However, CNPs optimize encoder jointly with the decoder on the downstream prediction task, i.e., prediction of function values y at unseen locations x, as illustrated in Figure 5.1a. This ties the meta-representation's quality to the combined encoder and decoder performances on this particular task and thereby makes it susceptible to supervision collapse, i.e. the representations lose any information which is irrelevant for solving the training task, but may be necessary for the transfer to new tasks (Doersch *et al.*, 2020). Moreover, many real-world tasks are noisy, and the prediction task might entail reconstructing high dimensional data, such as images in GQNs (Eslami *et al.*, 2018). The corresponding objective function can cause the model to waste its capacity on re-



(a) CNP (b) FCRL (encoder training) (c) FCRL (transfer)

Figure 5.1: **Comparison of CNP and FCRL.** The difference in the training of CNP (Garnelo *et al.*, 2018a) and FCRL for learning meta-representations r of functions. ***Figure 5.1a:*** CNP learns the aggregated representation r of the context set by maximizing the conditional likelihood of the target data. ***Figure 5.1b:*** Training of FCRL encoder h_ϕ by contrasting context sets of different functions. Note that the target inputs x_{N+1} etc., are not used at this stage. ***Figure 5.1c:*** Using the pretrained FCRL encoder h_ϕ, we train separate decoders p^*_ψ for each downstream task, shown in grey boxes. The dotted arrows indicate the transfer of inferred meta-representations to the tasks.

constructing unimportant features such as static backgrounds and noise, while ignoring visually small but important details in its learned representation (Anand *et al.*, 2019; Kipf *et al.*, 2019). This is crucial for many real-world applications; for instance, in order to manipulate a small object in a complex scene, the model's ability to infer the object's shape, form and size carries more importance than inferring its color, texture or reconstructing the static background.

In this work, we study the generalization of a function's meta-representations in terms of their transferability to downstream tasks and their robustness to noise. We empirically show that the joint optimization of meta-representations and a prediction task is detrimental to the transferability of meta-representations and makes them vulnerable to noise. To address this issue, we propose a decoupled encoder-decoder training scheme, wherein the encoder is exclusively trained by a novel contrastive learning framework which we call FCRL (Function Contrastive Representation Learning). Instead of relying on reconstructions, it learns by contrasting sets of input-output pairs sampled from different functions. The key idea is that two sets of samples from the same function should have similar latent representations, while representations of sets of samples from different functions should remain easily distinguishable. FCRL retains the useful properties of meta-representations such as shared learning and sample efficiency while improving its transferability to downstream tasks and robustness to noise. Unlike contemporary meta-learning algorithms, meta-representations in FCRL are explicitly optimized over a distribution of functions rather than tasks.

To evaluate the effectiveness of the proposed method, we conduct comprehensive experiments on diverse downstream problems, including classification, regression, parameter identification, scene understanding, scene reconstruction and reinforcement learning.

We consider different datasets, ranging from simple 1D and 2D regression to challenging simulated and real-world scenes. In particular, we find that a downstream predictor trained with our (pre-trained) encoder compares favorably to related methods on these tasks, including ones where the predictor is trained jointly with the encoder.

5.2 Related work

Meta-Learning. Supervised meta-learning can be broadly classified into two main categories. The first category considers the learning algorithm to be an optimizer and meta-learning is about optimizing that optimizer, for e.g., gradient-based methods (Ravi and Larochelle, 2016; Finn *et al.*, 2017; Li *et al.*, 2017; Lee *et al.*, 2019) and metric-learning based methods (Vinyals *et al.*, 2016; Snell *et al.*, 2017; Sung *et al.*, 2018; Allen *et al.*, 2019; Qiao *et al.*, 2019). The second category is the family of Neural Processes (NP) (Garnelo *et al.*, 2018a,b; Kim *et al.*, 2019; Eslami *et al.*, 2018) which draw inspirations from Gaussian Processes (GPs). These methods use data-specific priors in order to adapt to a new task at test time while using only a simple encoder-decoder architecture. However, they approximate the distribution over tasks in terms of their predictive distributions which does not incentivize NP to fully utilize the information in the data-specific priors. Our method draws inspiration from this simple, yet elegant framework. However, our proposed method extracts the maximum information from the context which is shown to be useful for solving not just one task, but multiple downstream tasks.

Self-Supervised learning. Self-supervised learning methods aim to learn the meaningful representations of the data by performing some pretext learning tasks (Zhang *et al.*, 2017; Doersch *et al.*, 2015). These methods have recently received huge attention (Oord *et al.*, 2018; Tian *et al.*, 2019; Hjelm *et al.*, 2018; Bachman *et al.*, 2019; Chen *et al.*, 2020; He *et al.*, 2019) mainly owing their success to noise contrastive learning objectives (Gutmann and Hyvärinen, 2010). At the same time, different explanations have recently come out to explain the success of such methods for e.g., from both empirical perspective (Tian *et al.*, 2020; Tschannen *et al.*, 2019) and theoretical perspective (Wang and Isola, 2020; Arora *et al.*, 2019). The goal of these methods has mostly been to extract useful, low-dimensional representation of the data while using downstream performance as a proxy to evaluate the quality of the representation. For example, CPC (Oord *et al.*, 2018) proposes an auto-regressive model to obtain a representation of a sample at time t that is then matched with that at time $t+k$, making it specialized for sequence-valued inputs. On the other hand, (Tian *et al.*, 2019; Chen *et al.*, 2020; He *et al.*, 2019) match the representation of a sample with the representation of its randomly augmented view. In this work, we take inspiration from these methods and propose a self-supervised learning method which meta-learn the representation of the functions. However, instead of requiring randomly augmented views or sequential ordered data points, our self-supervised loss uses partially observed, unordered views, sampled from the underlying functions.

In a similar spirit of enriching NPs (Garnelo *et al.*, 2018b) with better approximation capability, (Ton *et al.*, 2019) proposed to replace the conditional expectations $\mathbb{E}(y|x)$ in NPs with more expressive conditional densities $p(y|x)$ estimated via NCE (Gutmann and Hyvärinen, 2010). In contrast to FCRL, it directly estimates the conditional distribution $p(y|x)$ and uses a binary classifier for NCE. However, this estimation is done in a small data regime where the standard conditional density estimation does not work well. Therefore, their method is practically limited to low dimensional problems. Recently, (Zhang *et al.*, 2020; Srinivas *et al.*, 2020) has shown the benefits of using self-supervised representations, learned without reconstruction, for reinforcement learning tasks. In this work, we explore the utility of such representations for the reinforcement learning tasks defined on scenes (functions).

5.3 Preliminaries

5.3.1 Problem setting

Consider a distribution over data-generating functions $p(f)$. Let f be a sample from this distribution $f \sim p(f)$, where $f : \mathcal{X} \to \mathcal{Y}$ with $\mathcal{X} = \mathbb{R}^d$ and $\mathcal{Y} \subseteq \mathbb{R}^{d'}$:

$$y = f(x, \xi); \quad \xi \sim \mathcal{Z} \tag{5.1}$$

where ξ is sampled from some noise distribution $\mathcal{Z}$. Let $O^f = \{(x_i, y_i)\}_{i=1}^N$ be a set of few observed examples of a function f, referred to as the context set, and consider a set of downstream tasks $\mathcal{T}$. Here, each task $T \in \mathcal{T}$ can be defined as a mapping defined over f. In the case of few shot regression (see Section 5.5.1), T maps from f to a predictive model $p_\psi(y|x)$. In the case of parameter identification, T maps from f to some scalar or vector valued parameter of f.

Our goal is therefore to learn an encoder which maps a context set O^f to a representation of f that can interchangeably be used for multiple downstream tasks $\mathcal{T}$ defined on the same function (without requiring retraining).

5.3.2 Background

In this section, we briefly discuss a class of meta-learning methods that are particularly relevant to our encoder-decoder setting, namely conditional neural processes (CNPs) and generative query networks (GQNs) (Garnelo *et al.*, 2018a,b; Eslami *et al.*, 2018).

Conditional Neural Processes (CNPs). The key proposal in CNPs (applied to few-shot learning) is to express a distribution over predictor functions given a context set. They learn the meta-representations r by jointly training the encoder and decoder, as illustrated in Figure 5.1a. To this end, they first encode the context O^f into individual

representations $r_i = h_\Phi(x_i, y_i)\ \forall i \in [N]$, where h_Φ is a neural network. The representations are then aggregated via a mean-pooling operation into a fixed size vector $r = {}^1\!/_N(r_1 + r_2 + ... + r_N)$. The idea is that r captures all the relevant information about f from the context set O^f; accordingly, the predictive distribution is approximated by maximizing the conditional likelihood of the target distribution $p(y|x, O^f)$, where $y = f(x)$.

Generative Query Networks (GQN). GQN (Eslami *et al.*, 2018) can be seen as an extension of NPs (Garnelo *et al.*, 2018b) for learning 3D scenes representations. The context dataset O^f in GQN consists of tuples of camera viewpoints in 3D space ($\mathcal{X}$) and the images taken from those viewpoints ($\mathcal{Y}$). Like NPs, GQNs learn to infer the latent representation of the scene (regarded as a function here) by conditioning on the aggregated context and maximizing the likelihood of generating the correct image corresponding to a queried viewpoint.

5.4 Function-Contrastive Representation Learning (FCRL)

We take the perspective here that the sets of context points O^f provide a partial observation of an underlying function f. Our goal is to find an encoder $g_{(\phi,\Phi)}$ which maps such partial observations to low-dimensional representations of the underlying function. The key idea is that a good encoder $g_{(\phi,\Phi)}$ should map different context sets (i.e. partial observations) of the same function to be close in the latent space, such that they can easily be identified among context sets of different functions. This motivates the contrastive-learning objective which we will detail in the following.

Encoder structure. Since the inputs to the encoder $g_{(\phi,\Phi)}$ are sets, it needs to be permutation invariant with respect to input order and able to process inputs of varying sizes. We enforce this permutation invariance in $g_{(\phi,\Phi)}$ via sum-decomposition, proposed by (Zaheer *et al.*, 2017). We first average-pool the point-wise transformations of O^f to get the encoded representations

$$r^f = \frac{1}{|O^f|} \sum_{(x,y)\in O^f} h_\Phi(x,y) \tag{5.2}$$

where $h_\Phi(\cdot)$ is the encoder network. We then obtain a nonlinear projection of this encoded representation $g_{(\phi,\Phi)}(O^f) = \rho_\phi(r^f)$. Note that the function ρ_ϕ can be any nonlinear function. We use an MLP with one hidden layer which also acts as the projection head for learning the representation. Similar to (Chen *et al.*, 2020), we found that it is beneficial to define the contrastive objective on these projected representations $\rho_\phi(r^f)$,

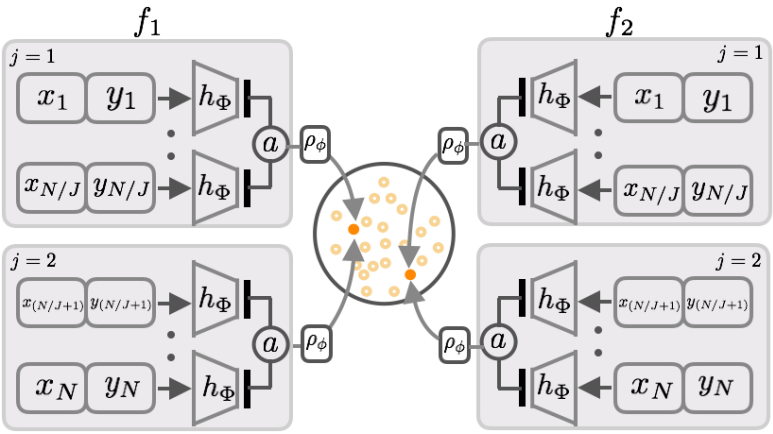


Figure 5.2: **Inner-workings of the FCRL objective function.** We split the context set of each function into J disjoint views, and align the aggregated representations of those views. Shown here is the example of two functions, with two views each i.e., $J = 2$.

rather than directly on the encoded representations r^f. More details can be found in our ablation study in Section 5.6.

Encoder training. At training time, we are provided with partial observations $O^{1:K}$ from K functions. Each observation is a set of N i.i.d. (independent and identically distributed) samples $O^k = \{(x_i^k, y_i^k)\}_{i=1}^N$. To encourage that different observations of the same functions are mapped to similar representations, we will now formulate a contrastive learning objective, as illustrated in Figure 5.2. To apply contrastive learning, we create different views of the same function k by splitting each sample set O^k into J subsets of size N/J. Defining $t_j := \{(j-1)N/J+1, ..., jN/J\}$, we obtain a split of O^k into J disjoint subsets of equal size:

$$O^k = \cup_{j=1}^J O_{t_j}^k \text{ with } O_{t_i}^k \cap O_{t_j}^k = \emptyset \quad \text{if} \quad i \neq j \tag{5.3}$$

where each subset $O_{t_j}^k$ is a partial view of the underlying function k. Here, J is a hyperparameter and can vary in the range of $[2, N]$ and must divide N, i.e. $N \mod J = 0$.

Its value is empirically chosen based on the data domain: for 1D and 2D regression problems (Section 5.5.1), the number of examples per view (N/J) is relatively large as a single context point does not provide much information about the underlying function; whereas in scenes (Section 5.5.3), a few (or even one) images per view (partial observation) may provide enough information. We expand on the appropriate choice of J in our experiments and the respective ablations. For brevity of notation, we define $v_j^k := g_{(\phi,\Phi)}(O_{t_j}^k)$. We now formulate the contrastive learning objective as follows:

$$\mathcal{L}=\sum_{k=1}^{K}\sum_{1\leq i<j\leq J}\left[\log\frac{\exp\left(\mathrm{sim}\left(v_j^k,v_i^k\right)/\tau\right)}{\sum_{m=1}^{K}\exp\left(\mathrm{sim}\left(v_j^k,v_i^m\right)/\tau\right)}\right] \tag{5.4}$$

where $\mathrm{sim}(a,b) := \frac{a^\top b}{\|a\|\|b\|}$ is the cosine similarity measure. Intuitively, the objective function in Equation ((5.4)) encourages that the similarity measure $\mathrm{sim}(v^p_{(\cdot)}, v^q_{(\cdot)})$ acts as a discriminatory function, yielding a large value if $v^p_{(\cdot)}$ and $v^q_{(\cdot)}$ are representations of sets of samples drawn from the same function, i.e. if $p = q$ (*positives*), and a small value otherwise, i.e. if $p \neq q$ (*negatives*). The second summation in Equation ((5.4)) over $1 \leq i \leq j \leq J$ is over available pairs of positives, and τ is a temperature parameter which scales the scores returned by the similarity measure. Similar to SimCLR (Chen *et al.*, 2020), we find that temperature adjustment is important for learning good representations and treat it as a hyperparameter (ablation study in Section 5.6).

We note that the learning objective effectively balances two goals. The first is that of avoiding overfitting (i.e., regularization). It encourages that any two independent samples from the same distribution get mapped to similar points. This is akin to the method of "*symmetrization by a ghost sample*" which is a standard trick in proving learning theory bounds (Vapnik, 1995). Essentially, if two means on different samples are close, then they will also be close to their expectation, i.e., they will not overfit to the data. This is an example of the more general phenomenon of concentration of measure, applicable not just to means but also to other functions that aggregate samples. For a simple argument, let $\mathbb{E}_{O^i}$ denote the expectation w.r.t. drawing the sample O^i, and g be the mapping function applied to two independent samples O^1, O^2 from the same distribution. Then we have the following.

$$\begin{aligned}
\left|g(O^1)-\mathbb{E}_{O^1}[g(O^1)]\right| &= \left|g(O^1)-\mathbb{E}_{O^2}[g(O^2)]\right| && (5.5)\\
&= \left|\mathbb{E}_{O^2}[g(O^1)-g(O^2)]\right| && (5.6)\\
&\leq \mathbb{E}_{O^2}\left[\left|g(O^1)-g(O^2)\right|\right] && (5.7)
\end{aligned}$$

The second equality uses independence of the samples O^1 and O^2, and the last step uses Jensen's inequality. This shows that if in expectation the embeddings of two samples are close (r.h.s.), then each embedding is close to its expectation. In spirit, this is close also to the idea of regularization by enforcing stability (i.e., weak dependence on sampling points) (Bousquet and Elisseeff, 2002). The second goal is to preserve contrastive information, ensuring that samples from different distributions get mapped to different points. Both goals are intricately linked in our setting, where the aggregation function is being learnt, since the second component is necessary to prevent the system from trivially meeting the first goal by, say, mapping everything to zero.

Estimating density ratios. The contrastive objective in Equation ((5.4)), in essence, tries to solve a classification problem i.e. to identify whether the given observation O^i comes from the function f^i or not. The supervision signal is provided by taking another observation $\hat{O}$ from the same function f^i as an anchor (a target label), thus making it a self-supervised method. This self-supervised, view-classification task, for a function f^i, leads to the estimation of density ratios between the joint distribution of observations $p(O^1, O^2|i)$ and their product of marginals $p(O^1|i)p(O^2|i)$. This joint distribution in turn corresponds to the joint distribution of the input-output pairs of the function $p(x,y|i)$. This way of learning a function's distribution is different from the typical regression objectives, which learn about a given function f^i by trying to approximate the predictive distribution $p(y|x)$. By assuming the universal function approximation capability of $g_{(\phi,\Phi)}$, and the availability of infinitely many functions $f^k \sim p(f)$ with fixed number of context points N each, the model posterior learned by the optimal classifier corresponding to Equation ((5.4)) would be equal to the true posterior given by Bayes rule.

$$f^k \sim P(f) \quad \forall \ k \in \{1,..,K\} \tag{5.8}$$

$$O^k \sim P(O|f^k) \quad \forall k \in \{1,..,K\} \tag{5.9}$$

$$i \sim \mathcal{U}(K) \tag{5.10}$$

$$\hat{f} = f^i \tag{5.11}$$

$$\hat{O} \sim P(O|\hat{f}) \tag{5.12}$$

$$\begin{aligned} p(i|O^{1:K},\hat{O}) &= \frac{p(O^{1:K},\hat{O}|i)p(i)}{\sum_i p(O^{1:K},\hat{O}|i)p(i)} \\ &= \frac{p(O^i,\hat{O}|i)p(i)\prod_{k\neq i} p(O^k|i)p(\hat{O}|i)}{\sum_j p(O^j,\hat{O}|j)p(j)\prod_{k\neq j} p(O^k|j)p(\hat{O}|j)} \\ &= \frac{\frac{p(O^i,\hat{O}|i)}{p(O_i)p(\hat{O})}p(i)}{\sum_j \frac{p(O^j,\hat{O}|j)}{p(O_j)p(\hat{O})}p(j)} \end{aligned} \tag{5.13}$$

The posterior probability for a function f^i is proportional to the class-conditional probability density function $p(O^i,\hat{O}|i)$, which shows the probability of observing the pair $(O^i,\hat{O})$ from function f^i. The optimal classifier would then be proportional to the density ratio given below

$$\exp(sim_{(\phi,\Phi)}(\hat{O},O^i)) \propto \frac{p(O^i,\hat{O})}{p(O_i)p(\hat{O})} \tag{5.14}$$

Similar analysis has been shown by the (Oord *et al.*, 2018) for showing the mutual information perspective associated with self-supervised contrastive objective (infoNCE). The joint distribution over the pair of observations correspond to the distribution of the

underlying function f^i. Thus, given some observation of a function, an optimal classifier would attempt at estimating the true density of the underlying function.

Application to downstream tasks. Once representation learning using FCRL is concluded, h_Φ is fixed and can now be used for few-shot downstream prediction tasks $\mathcal{T}$ defined on the underlying data-generating functions. To solve a particular downstream task, one may optimize a parametric decoder $p_\psi(\cdot|r)$ conditioned on the learned representation r. Specifically, the decoder maps the representations learned in the previous step to the variable of interest in the given task. Depending on the nature of the downstream task, the conditional distributions and the associated objectives can be defined in different ways.

5.5 Experiments

To illustrate the benefits of learning function representations without an explicit decoder, we consider four different experimental settings. In all experiments that follow, we first learn the encoder, and then keep it fixed. Subsequently, we optimize decoders for the specific downstream problems at hand, while keeping the meta-representations from the encoder detached from the computational graph.

Baselines. We compare the downstream predictive performance of FCRL based representations with the representations learned by the closest task-oriented, meta-learning baselines. For a fair comparison, all the baselines and FCRL have the same encoding architecture. For instance, for 1D and 2D regression functions, we consider CNPs and NPs as baselines. We share with these methods an identical way of mapping the context set to its representation, but unlike us, they optimize directly for the performance of the decoder $p(y|x)$ jointly with the said representation. For scene datasets, we use GQN (a variant of NPs) as the baseline, one that explicitly learns to reconstruct scenes using a limited number of context samples, comprised of pairs of camera viewpoints and their corresponding images.

5.5.1 1D functions

In the first set of experiments, we consider two different distributions of functions i.e., a distribution over 1D sinusoidal waves, proposed by (Finn *et al.*, 2017), and a relatively harder distribution where images are modelled as 2D functions (Garnelo *et al.*, 2018a,b; Gordon *et al.*, 2020; Kim *et al.*, 2019). The representation learning stage for both datasets is similar, however the downstream tasks differ.

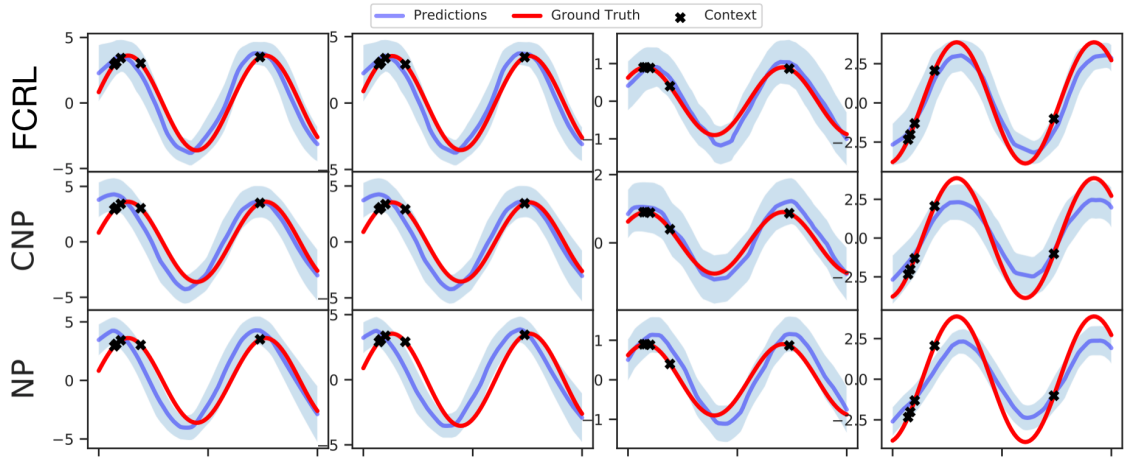


Figure 5.3: **Results on 5-shot sinusoid regression.** Each column corresponds to a different sinusoid function where only 5 context points are given. The predictions of the decoder trained on FCRL based encoder are closer to the groundtruth.

1D sinusoidal functions. We consider a dataset of $20,000$ training, 1000 validation and 1000 test sinusoidal functions. The amplitude and the phase of the functions are sampled uniformly from $[0.1, 0.5]$ and $[0, \pi]$ respectively. For each function f, the x-coordinates are uniformly sampled from $[-5.0, 5.0]$ and then f is applied to obtain the y-coordinates.

Modeling images as 2D functions. In this setting, each image is regarded as a function mapping from 2D pixel coordinates (comprising function input x_i) to the pixel intensities at the corresponding pixel coordinate (comprising function output y_i). We consider images of MNIST digits (LeCun *et al.*, 1998), where $x_i \in [0,1]^2$ are the normalized pixel coordinates and $y_i \in [0,1]$ is a grayscale pixel intensity. The training and validation datasets consists of $60,000$ MNIST training and $10,000$ test samples, respectively.

Representation learning stage

We first describe the representation learning stage for both datasets, and then provide results on their respective downstream tasks. For training the encoder $g_{(\phi,\Phi)}$, we have a dataset $O = \{O^k\}_{k=1}^K$ at our disposition, where each k corresponds to a function f_k which has been sampled as described above. Each individual sample $O^k = \{(x_i^k, y_i^k)\}_{i=1}^N$ from the dataset is itself a set, comprising N input-output pairs from that particular function f_k, i.e. $y_i^k = f_k(x_i^k)$. For sinusoidal functions, we fix the maximum number of context points to 20 and the number of examples N is chosen randomly in $[2, 20]$ for each k. For MNIST digits as 2D functions, we allow a maximum of 200 samples per context set, and N is sampled randomly from $[2, 200]$ for each k. The encoder $g_{(\phi,\Phi)}$ is then trained

	Few-shot Regression (FSR)	
Models	**5-shot**	**20-shot**
NP	0.310 ± 0.05	0.218 ± 0.02
CNP	0.265 ± 0.03	0.149 ± 0.02
FCRL	$\mathbf{0.172 \pm 0.04}$	$\mathbf{0.100 \pm 0.02}$

Table 5.1: Mean squared error (MSE) for all the target points in 5 and 20 shot regression on test sinusoid functions. The reported values are the mean and standard deviation of three independent runs. FCRL performs slightly better than CNP and NP on both tasks.

by splitting each context set O^k into J disjoint views. We set $J = 2$ for the sinusoidal functions and $J = 10$ for the 2D functions. An ablation study for the choice of J is presented in Section 5.6.

Downstream tasks on 1D sinusoids

After training the encoder $g_{(\phi,\Phi)}$, we discard the projection head ρ_ϕ and use the trained encoder h_Φ to extract the representations. For 1D sinusoids, we define two downstream tasks on the learned representation: $\mathcal{T}_{1D} = \{T_{fsr}, T_{fspi}\}$, where T_{fsr} and T_{fspi} are few-shot regression and few-shot parameter identification tasks, respectively. The decoders for the downstream tasks are trained as follows.

Few-shot regression (FSR). FSR for 1D functions is a well-studied problem in meta-learning (Garnelo *et al.*, 2018a; Finn *et al.*, 2017; Kim *et al.*, 2019; Xu *et al.*, 2019). For each sampled function f_k, we are given a context set $O^k = \{(x_i^k, y_i^k = f_k(x_i^k))\}_{i=1}^N$ of size N, which can be utilized to infer the meta-representation r^k of f_k via the pre-trained encoder h_Φ. We are then provided with M additional samples from f_k (not seen by the encoder h_Φ). The goal for a downstream decoder is to predict y_i^k, given x_i^k and the meta-representation r^k. In other words, the downstream decoder with parameters ψ models the distribution $p_\psi(y_i^k | x_i^k, r^k)$. Where $D^k = \{(x_i^k, y_i^k)\}_{i=1}^{N+M}$, the decoder is therefore trained to solve the following objective.

$$\max_{\psi} \mathbb{E}_{f_k \sim p(f)} \left[\mathop{\mathbb{E}}_{(x_i^k, y_i^k) \sim D^k} [\log p_\psi(y_i^k | x_i^k, r^k)] \right] \tag{5.15}$$

Here, the value of M is sampled randomly from the interval $[0, 20 - N]$. The decoder p_ψ is an MLP with two hidden layers and it is trained with the same training functions as the encoder h_Φ. In addition to the Gaussian mean of y_i^k, it also outputs the variance in order to quantify the uncertainty in the point estimates. The qualitative results on test functions for 5-shot regression are shown in Figure 5.3 while for 20-shot are shown in Figure 5.4. Both figures demonstrate that our model is able to quickly adapt with as few as 5 context

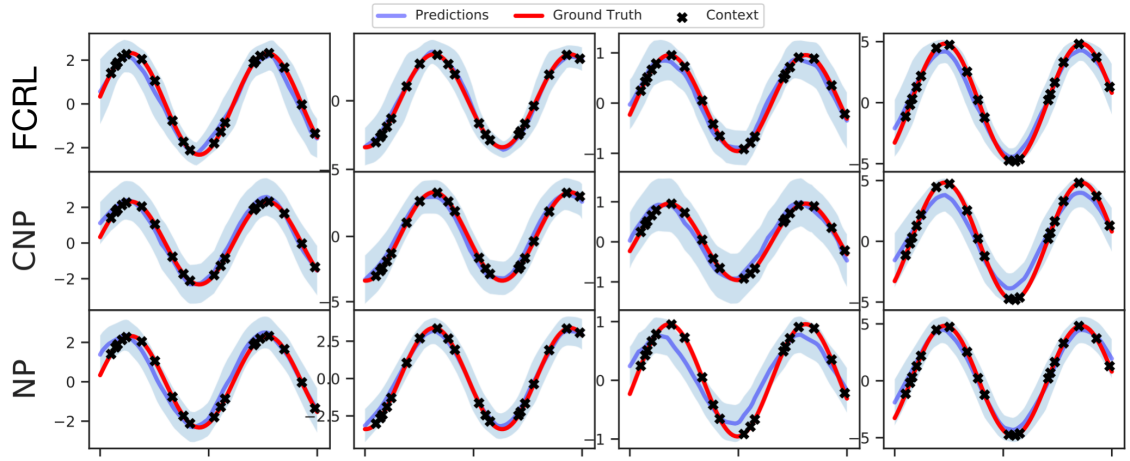


Figure 5.4: **Results on 20-shot sinusoid regression.** Each column corresponds to a different sinusoid function where only 20 context points are given. The predictions of the decoder trained on FCRL based encoder are comparable to CNP and better than NP.

points. The curves generated by the decoder using FCRL learned representations are closer to the groundtruth. The difference is evident in 5-shot experiments which supports the quantitative results in Table 5.1. In Table 5.1, we compare our method with CNP and NP quantitatively and show that the predictions of our method are closer to the groundtruth, even though the encoder and decoder in both CNP and NP are explicitly trained to directly maximize the log likelihood to fit the sinusoid.

Few-shot parameter identification (FSPI). The goal here is to predict the amplitude (y^k_{amp}) and phase (y^k_{phase}) of the sampled sine wave f_k, given a context set $O^k = \{(x^k_i, y^k_i = f_k(x^k_i))\}^N_{i=1}$ of N samples. Having encoded the context set O^k to meta-representation r^k via the pre-trained encoder h_Φ (following Equation ((5.2))), we train a linear decoder p_ψ on top of the said representation by maximizing the likelihood of the sine wave parameters. The predictive distribution is $p_\psi(y^k_{amp}, y^k_{phase}|r^k)$ and the objective is as follow.

$$\max_{\psi} \mathbb{E}_{f_k \sim p(f)}[\log p_\psi(y^k_{amp}, y^k_{phase}|r^k)] \tag{5.16}$$

Similar to FSR, we use the same training functions to train p_ψ as we did to train the encoder h_Φ. In Table 5.2, we report the mean squared error for three independent runs, averaged across all the test tasks for 5-shots and 20-shots FSR and FSPI. In both prediction tasks, the decoders trained on FCRL representations outperform CNP and NP. More details on the experiment are given in Appendix D.1.

	Few-shot Parameter Identification	
Models	**5-shot**	**20-shot**
NP	0.0087 ± 0.0007	0.0037 ± 0.0005
CNP	0.0096 ± 0.0007	0.0049 ± 0.0011
FCRL	$\mathbf{0.0078 \pm 0.0004}$	$\mathbf{0.0032 \pm 0.0002}$

Table 5.2: Mean squared error (MSE) for all the target points in 5 and 20 shot parameter identification tasks on test sinusoid functions. The reported values are the mean and standard deviation of three independent runs. FCRL performs slightly better than CNP and NP on both tasks.

5.5.2 2D functions

Similar to the tasks above, after training the model $g_{(\phi,\Phi)}$, we discard the projection head ρ_ϕ and use the trained encoder h_Φ to extract the representations. For MNIST digits as functions, we formulate two downstream prediction tasks on the learned representations: $\mathcal{T}_{2D} = \{T_{fsic}, T_{fscc}\}$, where T_{fsci} corresponds to few-shot image completion and T_{fscc} corresponds to few-shot content classification task. The decoders for the downstream tasks are trained as following.

Few-shot content classification (FSCC). To evaluate how much semantic information is captured by the meta-representations, we propose the task of few-shot content classification (FSCC). The goal here is to predict the class of each MNIST image given a context set $O^k = \{(x_i^k, y_i^k)\}_{i=1}^N$ comprising a few randomly sampled pixel coordinates x_i^k and the corresponding grayscale intensities y_i^k. The lack of explicit spatial structure in the context points makes it a challenging problem. We use the pre-trained encoder h_Φ to encode O^k to its representation r^k, and train a linear decoder on top to classify the class label $y_{one\_hot}^k$ corresponding to the MNIST image from which O^k is sampled. The decoder p_ψ therefore solves the following classification problem:

$$\max_{\psi} \mathbb{E}_{f_k \sim p(f)} [\log p_\psi(y_{one\_hot}^k | r^k)] \tag{5.17}$$

We train the decoder with the same functions (images) that were used for training the encoder h_Φ, and subsequently evaluate them on unseen functions from the validation set. Figure 5.5 shows the performance of decoders applied to representations obtained from FCRL, CNP and NP for varying size of the context set O^k. We find that FCRL significantly outperforms the baselines at any given number of context points, suggesting that the encoder h_Φ is able to efficiently extract semantic information in an unsupervised manner. We also observe that it is able to generalize to larger number of context points than encountered during training i.e., 200.

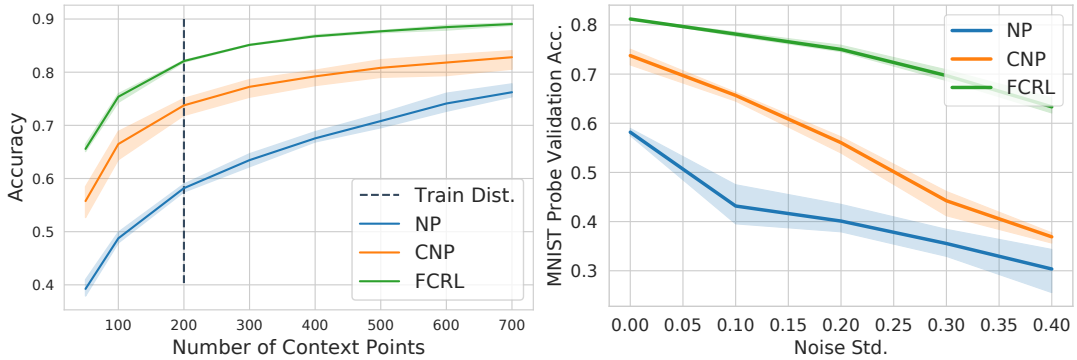


Figure 5.5: **Left:** Quantitative evaluation of the models in terms of digit classification from the fixed number of context points (varying along the x-axis). The error bands show the standard deviation over three runs. FCRL achieves substantially higher accuracy than both baselines for all evaluated numbers of context points. **Right:** Quantitative comparison for robustness to noise on MNIST content classification downstream task via linear probing. It can be seen that the FCRL based representations are much more robust to noise than with CNPs and NPs.

Few-shot image completion (FSIC). This setting is identical to that of Few-Shot Regression (FSR) described in Section 5.5.1, except M is sampled randomly from $[0, 200 - N]$ and the decoder is a two-layer MLP with two input units (to account for the fact that the input x_i^k is now two dimensional). Qualitative results of FSIC on test images are shown in Figure 5.6 for 50-shot completion and in Figure 5.7 for 200-shot image completion task. It can be seen that the decoder trained on FCRL representations is able to predict the pixel intensities reasonably well, even when the number of context points is as low as 50, or approximately 6% of the image. We compare its performance against CNP, which uses the same parameterization of both the encoder and the decoder. We however note a crucial distinction; in FCRL, the meta-representation (resulting from the encoder) is not optimized for the image completion task. In particular, no gradient flows from the decoder to the encoder, and the former is trained independently of the latter. On the contrary, CNP jointly optimizes both encoder and decoder parameters to solve the image completion task (i.e. to predict the pixel values). Despite the fact, it appears that the quality of reconstructions from the FCRL decoder matches that from the CNP decoder. We note that the gap between CNP and FCRL is reduced if the training scheme aligns with the downstream task. In FSIC, the downstream task is to obtain a generative model which is exactly what CNPs are trained for, therefore CNPs tend to perform on par with FCRL as shown in Table 5.3.

Robustness to noise corruptions on 2D functions

In our experiments so far, we have considered the functions to be deterministic. However in real-world settings, data-generating functions are corrupted with noise. We consider

	Few-shot Image Completion (FSIC)	
Models	**50-shot**	**200-shot**
NP	0.0531 ± 0.0002	0.0424 ± 0.0002
CNP	0.0477 ± 0.0006	0.0347 ± 0.0011
FCRL	0.0481 ± 0.0001	0.0355 ± 0.0001

Table 5.3: MSE for all the target points in 50 and 200 shot image completion task on MNIST as 2D functions. The reported values are the mean and standard deviation of three independent runs.

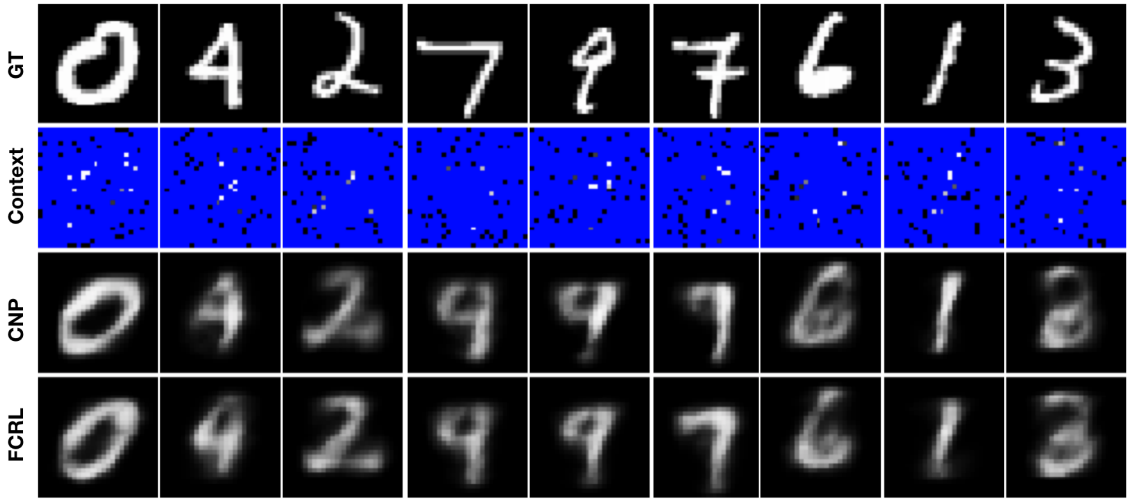


Figure 5.6: Qualitative comparison of CNP and FCRL on 50-shot MNIST image completion. Here, each digit corresponds to one function. The context is shown in the second row where target pixels are colored blue. Predictions made by a decoder trained on FCRL based encoder are slightly better than the CNP in terms of guessing the correct form of digits, demonstrating better high-level understanding of the task.

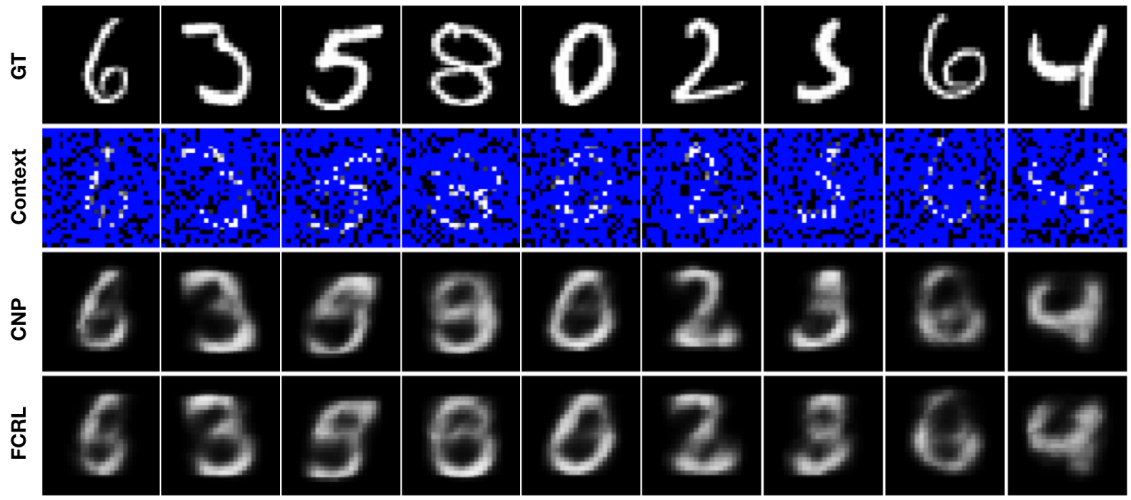


Figure 5.7: Qualitative comparison of CNP and FCRL on 200-shot MNIST image completion. Here, each digit corresponds to one function. The context is shown in the second row where target pixels are colored blue. Predictions made by a decoder trained on FCRL based encoder are comparable to CNP.

the form of the function as given in Equation ((5.1)) and vary the standard deviation σ of the added noise. We now investigate the robustness of FCRL and the baselines as σ is varied. To this end, we train all the models on the noisy data and evaluate the quality of the learned representation on the Few-Shot Content Classification downstream task, as defined above. We find that the representations learned by FCRL to be significantly more robust to increasing noise strength (σ) than the baselines, as illustrated in Figure 5.5(right). One possible explanation for the susceptibility of CNP and NP to noise is the fact that the representations are learned by reconstructing the outputs, where signal to noise ratio is low. On the other hand, FCRL learns by contrasting the set of examples, extracting only the invariant features and discarding non-correlated noise in the input. Similar results on scene understanding datasets are presented in Section 5.5.3.

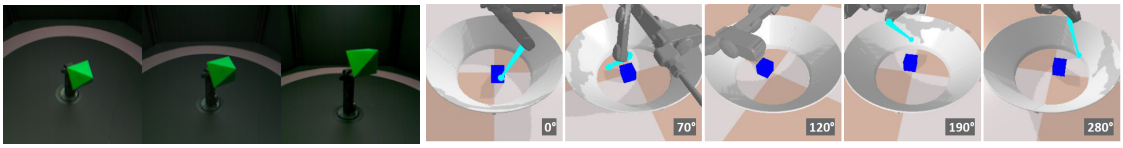
5.5.3 Representing scenes as functions

Like Eslami *et al.* (2018), we represent scenes as deterministic functions which map camera viewpoints to images. Precisely, each such scene is represented by a function f_k, and we consider context sets $O^k = \{(x_i^k, y_i^k)\}_{i=1}^N$, where x_i is the 3D camera viewpoint and y_i is the corresponding image taken from that viewpoint. Given this set of viewpoint-image pairs, we apply the proposed method on O^k to obtain a representation of the scene, r^k. The usefulness of this representation is then evaluated for three downstream tasks: (1) scene understanding, (2) scene reconstruction and (3) reinforcement learning (RL). For the first task, our goal is to determine whether the representation r^k contains enough information to infer the underlying factors of variation (Bengio *et al.*, 2013) of a given

scene. For the second task, we analyze whether the learned representation r^k can be used to reconstruct the scene from an unseen viewpoint. This objective is what GQNs (Eslami *et al.*, 2018) are originally trained for thus a competitive baseline for this task.

MPI3D dataset. To this end, we use MPI3D (Gondal *et al.*, 2019a), a real-world robotics dataset containing pairs of images from three camera viewpoints and the corresponding factors of variation (including the position, orientation, size and color of an object in the scene). The dataset comes in three different formats, varying in the levels of realism i.e. real-world, simulated-realistic and simulated-toy. Each dataset contains $1,036,800$ images of a robotic manipulator each, encompassing seven different factors of variations i.e., object colors (6 values), object shapes (6 values), object sizes (2 values), camera heights (3 values), background colors (3 values), rotation along first degree of freedom ((40 values)) and second degree of freedom ((40 values)). Thus, each image represents a unique combination of all the factors. We refer to Figure 5.8(a) for example datapoints of the dataset.

In this work, we consider the real-world version of the dataset. The multi-view setting is formulated by considering the images of a scene captured by three different cameras, placed at different heights. This effectively gives us $345,600$ scenes with three views each. We split the dataset into training and validation chunks, where the training dataset contains $310,000$ scenes and the validation dataset contains the rest $35,600$ scenes, approximately 10% of the dataset.



(a) (b)

Figure 5.8: **Overview of scenes datasets.** Datasets for Scenes Representation Learning **(a)** MPI3D (Gondal *et al.*, 2019a) has three camera viewpoints, with images of a robotics arm manipulating an object. **(b)** RLScenes has 36 possible camera viewpoints for capturing an arena consisting of a robot finger and an object.

RLScenes dataset. Finally, for the last task, our objective is to determine whether the learned representation r^k contains enough useful information to guide an RL agent towards maximizing its reward. To this end, we create RLScenes, a multi-view robotics dataset based on an open-source physics simulation engine. The RLScenes dataset is generated in simulation using (Joshi *et al.*, 2020) for a single 3-DOF robotic manipulator in a 3D environment. The dataset consists of $40,288$ scenes, each scene parametrized by: object colors (one of 4), robot tip colours (one of 3), robot positions (uniformly sampled from the range of feasible joint values), and object positions (uniformly sampled within

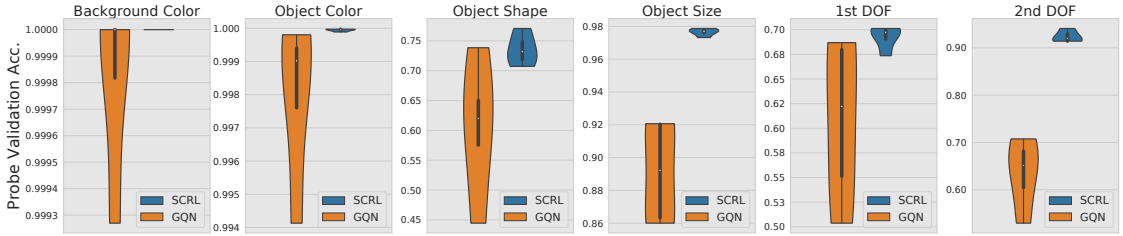


Figure 5.9: **Scene understanding comparison.** Quantitative Comparison of FCRL and GQN on MPI3D downstream classification tasks. The classifiers trained with FCRL's representation outperform classifiers based on GQN's representations on all the tasks.

an arena bounded by a high boundary as seen in Figure 5.8). Each scene consists of 36 views, corresponding to the uniformly distributed camera viewpoints along a ring of fixed radius and fixed height, defined above the environment. As can be seen in Figure 5.8, the robot finger might not be visible completely in all the views, or the object might be occluded in some view. The 36 views help by capturing a 360 deg holistic perspective of the environment. First a configuration of the above scene parameters is selected and displayed in the scene, then the camera is revolved along the ring to capture its multiple views. For learning the scene representations via both FCRL and GQN, we split the dataset into 35000 training and 5288 validation points. Having trained the encoder on RLScenes, we feed the representation r^k of the scene as input to a control policy rewarded for solving the considered RL task.

Scenes representation learning

We use the same setting for learning the representations on both datasets. We fix the maximum number of context sets (J) to 3 in MPI3D dataset and 20 in RLScenes. The number of tuples drawn, N, is then chosen randomly in $[2,3]$ and $[2,20]$ respectively. For learning the scene representations for both MPI3D dataset and RL Scenes, we used similar base encoder architecture. More specifically, we adapted the "pool" architecture provided in GQN (Eslami *et al.*, 2018), as it has been regarded to exhibit better view-invariance, factorization and compositional characteristics as per the comprehensive study done in (Eslami *et al.*, 2018). We further augmented this architecture with batch-normalization. The architecture used is shown in Figure D.1. For more details on scenes representation learning for both MPI3D and RLScenes, we refer readers to Appendix D.3.

Even though no explicit structural constraint is imposed by FCRL for learning scenes representations. The algorithm implicitly figures out the commonality between the factors in different scenes. We visualize these latent clusterings in the Figure 5.10. We plot the 2D t-SNE embeddings of the 128 dimensional representations inferred by the model. Thereafter, to visualize the structure corresponding to each factor, we only vary one factor and fix the rest of them except for first degree of freedom and second degree of freedom factors. A clear structure can be seen in the learned representations.

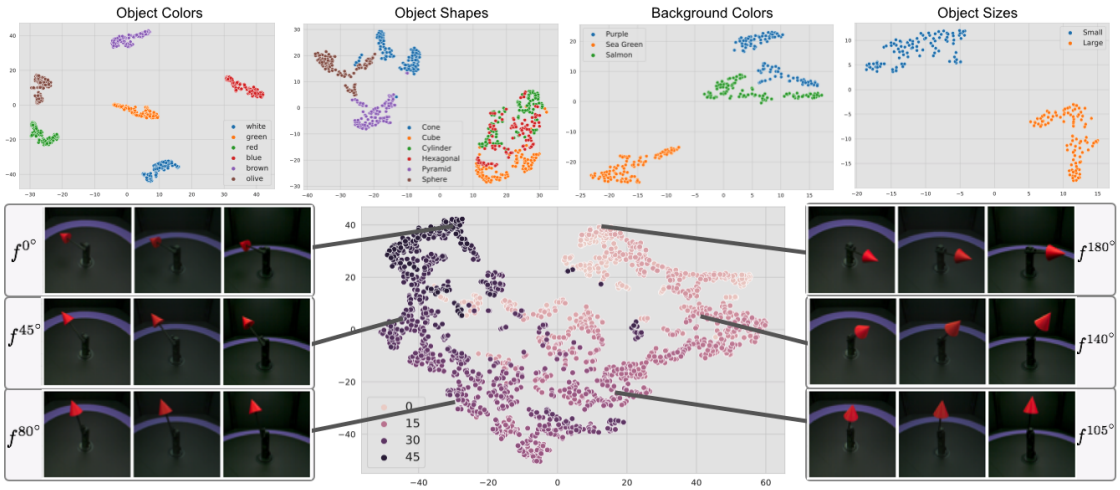


Figure 5.10: t-SNE projections of the meta-representations learned by FCRL on MPI3D dataset. **Top row:** Each plot shows the latent structure corresponding to the factors mentioned. **Bottom row:** Shows the latent structure corresponding to the factor of robot arm rotation along 1^{st} DOF. Each embedding corresponds to the aggregated representation of three views of the same scene, denoted as $f^{angle^{\circ}}$. It can be seen that by learning to distinguish between functions, FCRL captures the semantic underlying structure of the functions' distribution. *Note:* Each plot is generated by varying one factor and keeping the rest fixed, except for the factors of the first and second degree of freedom.

Downstream tasks on scenes

After training the encoder $g_{(\phi,\Phi)}$, we discard the projection head ρ_ϕ and use the trained encoder h_Φ to extract the representations r^k and train decoders for downstream problems.

Scene understanding on MPI3D dataset. In MPI3D, each scene is identified by 6 factors of variations. This allows us to define a set of 6 tasks $\mathcal{T}_{mpi3D} = \{T_v^k\}_{v=1}^6$, where the task T_v^k maps the scene to a discretized factor of variation y_v^k. For each task, we train a linear decoder using the objective in Equation ((5.17)), using a single image to infer the representation r^k. Figure 5.9 shows the linear probes validation performance for six independently trained models on both GQN learned representations and FCRL learned representations and we see that the representations learned by FCRL consistently outperform GQN for identifying all the factors of variations in scenes.

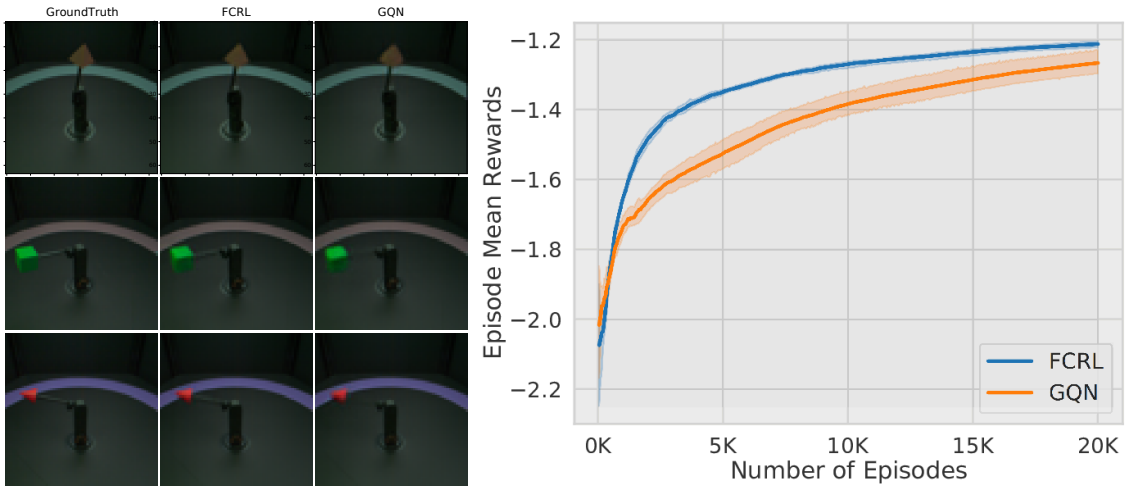
Scene reconstruction on MPI3D dataset. Similar to the FSIC task in Section 5.5.2, we train a separate decoder to reconstruct the scenes corresponding to an unseen (query) viewpoint x_q^k. Conditioning on the inferred representation r^k and the query viewpoint x_q^k, the decoder reconstructs the corresponding view of the scene y_q^k. The decoder architecture is the same as the one used in GQN. The qualitative comparison in Figure 5.11a shows that the decoder trained with FCRL representation is capable of preserving the

information required to reconstruct the subtle details in the scene.

Reinforcement learning on RLScenes dataset. In RLScenes, the goal for the agent (a robotic finger) is to locate the object in the arena, reach it, and stay close to it for the remainder of the episode. We use the Soft-Actor-Critic (SAC) algorithm (Haarnoja *et al.*, 2018) to learn an MLP policy for all the joints of the robot, where the policy takes as input the representation r^k (inferred from a single image) and outputs an action. As the baseline, we use a RL policy trained with GQN representation as input. Figure 5.11b shows the mean rewards and standard deviations over five runs achieved by both FCRL and GQN-based policies. We find that the FCRL agent clearly outperforms the baseline GQN-agent, both in terms of the final performance and sample-efficiency. In particular, the FCRL agent obtains convergence level control performance with approximately 2 times fewer interactions with the environment.

Robustness to noise in learning scenes representations

Similar to the noise robustness experiments for 2D datasets in Section 5.5.2. We consider noise robustness for images datasets. We likewise consider the form of the function as given in Equation ((5.1)) and vary the standard deviation σ of the added noise.



(a) Reconstructed MPI3D scenes. (b) RL agent performance on scenes representations.

Figure 5.11: ***Figure 5.11a:*** Qualitative Comparison of FCRL and GQN on scene reconstruction task. The decoder trained with FCRL's representation performs on par with GQN in terms of reconstructing a scene from unseen viewpoints. ***Figure 5.11b*** Comparison between GQN and FCRL on learning a data-efficient control policy for an object reaching downstream task. FCRL based policy clearly outperforms GQN based policy.

It can be seen that with the increased level of noise the features in the image start to diminish, shown in the Figure 5.12. GQN approach the learning problem by reconstruct-

ing these noisy images where the signal to noise ratio is very low. On the other hand,

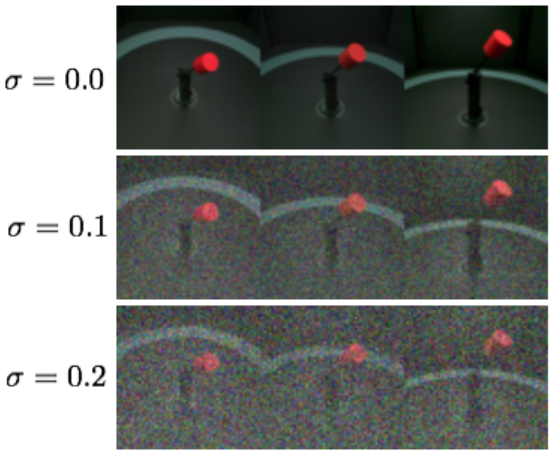


Figure 5.12: **Noise overlay on MPI3D**. For the scene understanding downstream task, we considered varying level of additive Gaussian noise on MPI3D dataset.

FCRL learns to contrast the scene representations with other scenes without requiring any reconstruction in the pixel space. This helps it in extracting invariant features in the views of a scene, getting rid of any non-correlated noise in the input. In addition to the analysis on MNIST 2D regression task in Figure 5.5, we test the performance of these representations learning algorithm on MPI3D factors identification task in Figure 5.13. It can be seen that FCRL representations can recover the information about the data factors, even in the extreme case where the noise level is very high (standard deviation of 0.2). On the other hand, GQN performs very poorly such that the downstream probes achieve the random accuracy.

5.6 Ablating FCRL hyperparameters

FCRL consists of several hyperparameters. In the following, we study the importance of each hyperparameter and provide details on the optimal configuration for each task type in different datasets domains.

Role of number of observations (J). The number of observations J corresponds to the number of partial observations that we have of a functions f^k. Ideally, we only need two such observations to learn the representations via contrastive objective. However, it has been shown that having more positive pairs result in learning better representations (Chen *et al.*, 2020; Tian *et al.*, 2019).

It should be noted that in our setting, the number of observed context sets N do not necessarily correspond to the number of observations J. This is because for some datasets,

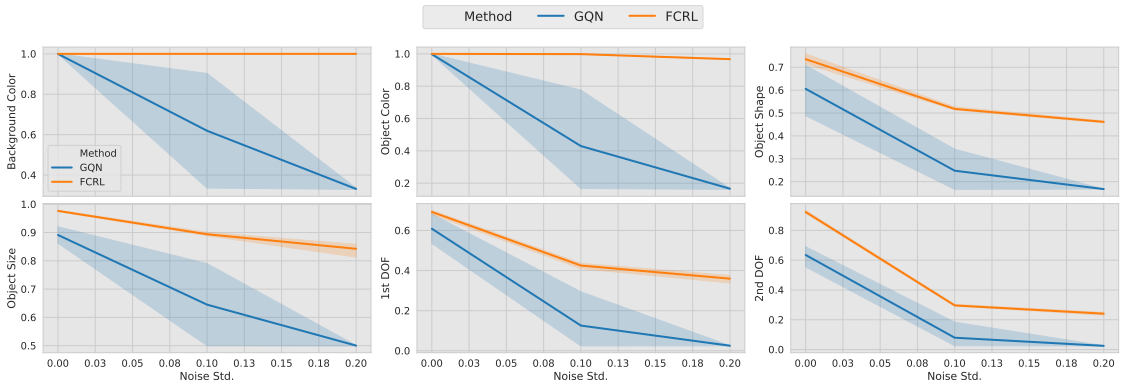


Figure 5.13: Quantitative comparison of FCRL and GQN for noise robustness on MPI3D downstream tasks. X-axis in each plot corresponds to the varying level of Gaussian noise (as depicted in Figure 5.12). The representations are extracted from one view only, and the accuracy bands show the standard deviation of five independent runs. It can be seen that the linear classifiers trained on FCRL representations are able to predict the attributes of objects even in extremely noisy case, significantly outperforming the linear classifiers trained on GQN representation.

we aggregate the context points to get one partial observation (see Figure 5.2). This is important for the simple 1D and 2D regression functions where a single context point does not provide much information, hence the individual partial observations need more than one context point. In our setting, we treat J as a hyperparameter whose optimal value varies depending on the function. For instance, the MPI3D scene dataset has only 3 views per scene, therefore J can not be greater than 3 and we keep it fixed. For 1D and 2D regression functions, we observe that for a fixed number of context points N, the optimal number of observations J varies. For understanding the role of J better in these experiments, we perform an ablation study on MNIST2D dataset with FSCC (few-shot content classification) as the downstream task Figure 5.14. For each hyper-parameter configuration we train three models, initialized with different random seeds. The maximum number of context points is fixed to 200 while the J varies between 2 and 40. It can be seen that the smaller values of J do not result in better FSCC score, however, the accuracy seems to plateau after $J = 10$. Therefore, we fix it to 10 for MNIST2D. For RLScene dataset, we fixed the maximum number of context points to be 20 and found the optimal number of partial observations to be $J = 4$. Note that the FSCC accuracy seems to be more influenced by the hyperparameters of critic and temperature, shown concurrently in the Figure 5.14. We discuss their roles in the next section.

Role of critics and temperature (τ). We regard the discriminative scoring functions, including the projection heads, as critics. The simplest critic function does not contain any projection layer, regarded as *dot product* critic, where the contrastive objective is

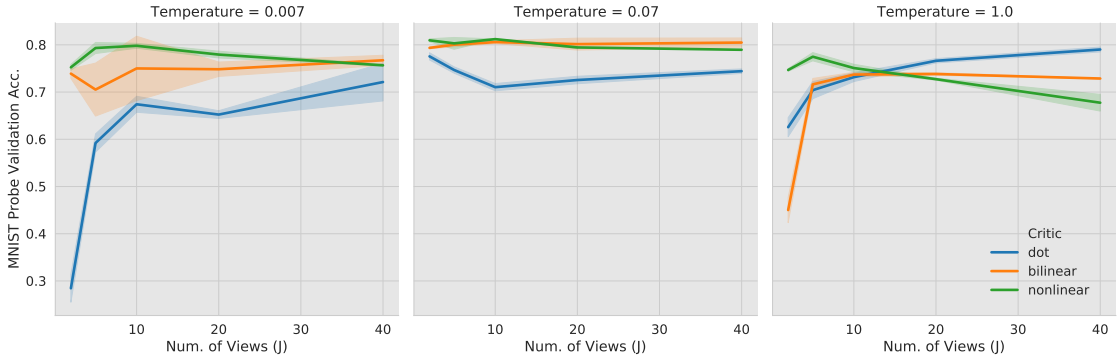


Figure 5.14: **Ablating number of views J.** An ablation study for understanding the role of the number of partial observations (J), the critics and the temperature parameter (τ) for learning FCRL based representation. The graph shows the accuracy achieved by the linear classifier (Few-shot content classification task) on MNIST digits validation dataset. Note that each image corresponds to a 2D functions, and the accuracy bands show the standard deviation of three independent runs.

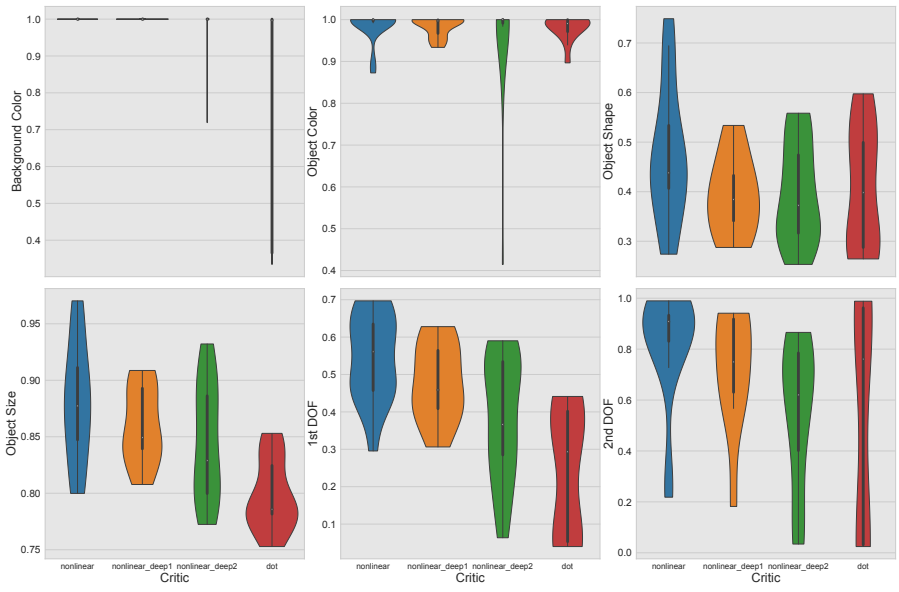


Figure 5.15: **Ablating Critics.** An ablation study to understand the role of four different critics in learning meta-representations of MPI3D scenes via FCRL. Shown here is the accuracy achieved by six different linear classifiers, trained on the representations, for identifying the six factors of variation. Non-linear critic consistently performs better than the other critics.

defined directly on the representations returned by the base encoder h_Φ. However, recently the role of critics in learning better representations has been explored in depth (Oord *et al.*, 2018; Chen *et al.*, 2020). Building on these findings, we evaluate the role played by different critics in learning the functions representations. Figure 5.14 shows the ablation for three different critics on MNIST2D validation dataset. It can be seen that the performance of critics is also highly linked with the temperature parameter τ. For an optimal temperature value τ, the non-linear critic performs consistently better.

Such hyperparameter grid search (done for MNIST2D) is very expensive for the ablation studies on the bigger datasets, such as, the MPI3D and RLScenes datasets. Therefore, we define the range for the t and perform a random sweep of 80 models with randomly selected hyper-parameter values for critic and temperature on MPI3D dataset. We did not find any pattern for the effect of temperature τ on the downstream tasks, however the pattern emerged for the class of critics. Figure 5.15 shows the ablation for critics on MPI3D dataset. It can be seen that the non-linear critic performs better in extracting features which are useful for the downstream classification tasks. Because of this trend across two different datasets, we therefore performed all our experiments with non-linear critic. The project head in nonlinear critics is defined as an MLP with one hidden layer and batch normalization in between.

5.7 Conclusion

In this work, we proposed a novel self-supervised representations learning algorithm for few-shot learning problems. We deviate from the commonly-used, task-specific training routines in meta-learning frameworks and propose to learn the representations of the relevant functions independently of the prediction task. Experiments on various datasets and the related set of downstream few-shot prediction tasks show the effectiveness of our method. The flexibility to reuse the same representation for different task distributions defined over functions brings us one step closer towards learning a generic meta-learning framework. Using a shared generic representation of the data-generating process, we plan to adapt the proposed framework in order to tackle multiple challenging few-shot problems such as object detection, segmentation, and visual question answering.

Chapter 6

Conclusion

The main goal of this dissertation was to examine how incorporating various inductive biases, specifically disentanglement, modularity, and meta-learning can enhance knowledge transfer in deep learning models. Through the introduction of novel algorithms and datasets, we systematically studied the role of these biases in developing more adaptive and generalizable models. In the following, we summarize the contributions, limitations, and future prospects of each inductive bias, grounded in the broader context of enhancing the generalization and knowledge transfer abilities of deep learning models.

The thesis commenced with the investigation of learning disentangled representations in sequential videos and static images. Learning well-disentangled representations of data has been identified as crucial for transfer and few-shot learning Bengio *et al.* (2013); Schölkopf *et al.* (2021); Peters *et al.* (2017). In this context, to learn video dynamics, we developed *Disentangled State Space Models* that demonstrated how separating invariant system dynamics from domain-specific details can significantly enhance a model's generalization across diverse environments. For images, we introduced the first real-world disentanglement dataset, *MPI3D*, and its two simulated versions with varying levels of realism. Benchmarking on these datasets provided the first glimpse of the performance gap on real and synthetic datasets by contemporary, state-of-the-art disentanglement algorithms that were previously tested only on synthetic datasets. The benchmark also revealed that models trained on synthetic data have poor transferability to real-world scenarios, underscoring the need to develop improved models that can effectively utilize simulated data for practical applications. Moreover, besides evaluating representation learning algorithms, the dataset introduced in this work can be applied to a range of other tasks, including learning of compositional visual concepts, 3D reconstruction, scene rendering etc.

Next, we focused on modularity in deep neural networks and introduced a novel yet flexible architecture, *Neural Interpreters*, which factorizes the computational graph into functional units dynamically composed at inference time. Through extensive evaluations on three diverse problem settings i.e., learning fuzzy boolean expressions, multi-task image classification, and abstract reasoning, we demonstrated that Neural Interpreters outperform vanilla self-attention networks like vision transformers Dosovitskiy *et al.* (2020) in terms of systematic generalization and sample-efficient transfer to new tasks. The flex-

ibility of Neural Interpreters in combining and reusing computational primitives draws inspiration from the concept of learning reusable mechanisms Schölkopf *et al.* (2021) and mirrors the cognitive adaptability seen in human learning, enabling the application of learned concepts to new tasks with notable efficiency.

Finally, we investigated the goal of learning generic meta-learning algorithms. More specifically, we introduced the first approach to marry meta-learning with self-supervised learning in order to learn sample-efficient yet transferable *meta-representations* of functions. The resulting methodology, *Function Contrastive Representation Learning* (FCRL), provided a novel perspective on learning stochastic functions that benefit from both the sample efficiency of meta-learning frameworks and the transferability of contrastively learned representations, while maintaining high performance on a set of downstream predictive tasks.

Limitations and future work. Even though the MPI3D dataset aims to narrow the performance gap between synthetic and real-world datasets for disentanglement algorithms, its representation of reality is somewhat limited. The dataset was captured under controlled laboratory conditions to ensure the independence amongst different factors. However, a real-world dataset, captured in the wild, would serve as an ultimate benchmark for disentanglement algorithms. Capturing such data, with known factors of variation and without confounders poses significant challenges, thus quantifying the degree of disentanglement and its direct impact on real-world downstream task performance remains an open question. To address this issue partially, we released an updated version of the dataset, MPI3D-complex[1], featuring complex 3D models of items like coffee cups and tennis balls. While CelebA Liu *et al.* (2015) provides a closer proxy to a real-world setting, it lacks information on independent variation factors. Efforts should extend beyond image data to capture datasets in other modalities, enabling reliable use of disentanglement algorithms in fields like robotics, healthcare, and fair decision-making.

In developing Disentangled State Space Models (DSSM), we made several implicit assumptions about the generative model for dynamical systems, such as known world dynamics and the presence of only a single varying dynamical component across environments. This setting is restrictive, given the complexity of real-world dynamics, such as varying gravity, elasticity, and surface friction. Future models should allow for the disentanglement of multiple factors, enhancing domain adaptability and generalization across diverse environments.

Neural Interpreters inherently rely on the attention mechanism, which may not always capture the most efficient pathways for knowledge reuse across tasks. This limitation can be mitigated by the meta-learning approaches, that can guide functions for selecting tokens based on content. In addition, the computational complexity of Neural Interpreters scales quadratically with the size of the input. Some of these above issues have been addressed in Weiss *et al.* (2022). The architecture of Neural Interpreters naturally al-

[1] `https://github.com/rr-learning/disentanglement_dataset`

lows the extension of capacity that can be meaningfully exploited for continual learning framework in future, enabling continuous integration of knowledge without catastrophic forgetting. Further research could also explore integrating modularity with disentangled and object-centric representations, potentially enhancing symbolic reasoning and systematic generalization capabilities. For FCRL, a natural extension would involve applying meta-representations trained on image class functions to tasks like classification, segmentation, and depth estimation.

The insights from the extensive investigations on disentangled representations learning, modularity, and meta-learning within this dissertation underscores the multifaceted approach required to advance the field of deep learning towards more human-like generalization capability. Each inductive bias contributes uniquely to this overarching goal, yet the integration of all presents a promising frontier for future research. Moreover, as these inductive biases complement each other, we did not investigate the joint implementation of these inductive biases in a single system. Such a model will surely benefit from the structured and compositional structure induced by disentanglement and modularity, meanwhile it will have the capability to adapt to new tasks in sample efficient manner.

Furthermore, this work does not provide a comprehensive overview of all inductive biases that can improve generalization, such as the roles of deep learning architectures and objective functions. We refer readers to Schölkopf *et al.* (2021); Goyal and Bengio (2022) for further exploration and broader understanding.

Appendix A

Appendix to Chapter 2

A.0.1 Optimization details for training DSSM

Following the insights from (Bowman *et al.*, 2015), we tried different settings for KL annealing in the model. In DSSM, we have three KL terms in our model which have different roles. To make it relatively easier for the model to learn the time-invariant components we do not penalize terms of time-invariant components i.e. $KL(q(S_0|\vec{X})||p_0(S))$ and $KL(q(E|\vec{X})||p_0(E))$ as much as $KL(q(\beta_i|S_i,X_i)||p_0(\beta))$ during training. This makes it relatively easier for the model to learn the time-invariant components. Similarly to (Fraccaro *et al.*, 2017), we also found that down-weighing the reconstruction term helps in faster convergence. In particular we applied scaling coefficients of $[0.1, 0.2, 0.3]$ for terms $\mathbb{E}_{q(\vec{S}|\vec{X})}[\log\, p(\vec{X}|\vec{S})]$, $KL(q(E|\vec{X})||p_0(E))$, and $KL(q(S_0|\vec{X})||p_0(S))$ respectively and 1.0 for $KL(q(\beta_i|S_i,X_i)||p_0(\beta))]$. We use ADAM as the optimizer with 0.0008 as the initial learning rate, and weight decay of 0.6 applied every 20 epochs.

A.0.2 Evidence lower bound derivation for DSSM objective

The conditional likelihood of DSSM objective $\mathcal{L}$ is as follows.

$$\mathbb{E}_{q(\vec{S}|\vec{X})}[\log\, p(\vec{X}|\vec{S})] = \mathbb{E}_{q(\vec{S}|\vec{X})}[\log \prod_{i=1}^{T} p(X_i|S_i)] = \mathbb{E}_{q(\vec{S}|\vec{X})}[\sum_{i=1}^{T} \log\, p(X_i|S_i)]$$
$$= \sum_{i=1}^{T} \mathbb{E}_{q(S_i|\vec{X})}[\log\, p(X_i|S_i)]$$

where the conditional independence follows from the state space model formulation. Following is the derivation of evidence lower bound for the DSSM variational objective in Equation (2.8).

$$
\begin{aligned}
&\mathrm{KL}\left(q(\vec{S}|\vec{X})\|p_0(\vec{S})\right) \\
&= -\int q(\vec{S}|\vec{X}) \log\left(\frac{p_0(\vec{S})}{q(\vec{S}|\vec{X})}\right) d\vec{S} \\
&= -\int q(S_0|\vec{X}) q(E|\vec{X}) q(\vec{\beta}|\vec{X},S_0,E) q(\vec{S}_0|S_0,E,\vec{\beta},\vec{X}) \\
&\quad \log\left(\frac{p_0(S_0)p_0(E)p_0(\vec{\beta}|S_0,E)p_0(\vec{S}_0|S_0,E,\vec{\beta})}{q(S_0|\vec{X})q(E|\vec{X})q(\vec{\beta}|\vec{X},S_0,E)q(\vec{S}_0|S_0,E,\vec{\beta},\vec{X})}\right) d\vec{S}_0 d\vec{\beta} dE dS_0 \\
&\quad \textit{(Factorizing the distributions based on the model's structure and the assumption of conditional dependencies.)} \\
&\quad \left(\text{Note } \vec{S}_0 \text{ is vector } \vec{S} \text{ without } S_0\right) \\
&= -\int q(E|\vec{X}) \log\left(\frac{p_0(E)}{q(E|\vec{X})}\right) dE \\
&\quad -\int q(S_0|\vec{X}) \log\left(\frac{p_0(S_0)}{q(S_0|\vec{X})}\right) dS_0 \\
&\quad -\int q(S_0|\vec{X}) q(E|\vec{X}) \int q(\vec{\beta}|\vec{X},S_0,E) \log\left(\frac{p_0(\vec{\beta}|S_0,E)}{q(\vec{\beta}|\vec{X},S_0,E)}\right) d\vec{\beta} dE dS_0 \\
&\quad -\int q(S_0|\vec{X}) q(E|\vec{X}) q(\vec{\beta}|\vec{X},S_0,E) \int q(\vec{S}_0|S_0,E,\vec{\beta},\vec{X}) \log\left(\frac{p_0(\vec{S}_0|S_0,E,\vec{\beta})}{q(\vec{S}_0|S_0,E,\vec{\beta},\vec{X})}\right) d\vec{S}_0 d\vec{\beta} dE dS_0 \\
&\quad \text{(Expanding each term, dropping the corresponding term based on conditional dependencies.)} \\
&= -\mathrm{KL}\left(q(E|\vec{X})\|p_0(E)\right) - \mathrm{KL}\left(q(S_0|\vec{X})\|p_0(S_0)\right) \\
&\quad -\mathrm{KL}\left(q(\vec{\beta}|\vec{X},S_0,E)\|p_0(\vec{\beta}|S_0,E)\right) \\
&\quad -\mathrm{KL}\left(q(\vec{S}_0|S_0,E,\vec{\beta},\vec{X})\|p_0(\vec{S}_0|S_0,E,\vec{\beta})\right) \\
&\textit{(where we dropped the integral sums for which the corresponding term does not depend on)} \\
&= -\mathrm{KL}(q(E|\vec{X})||p_0(E)) \\
&-\mathrm{KL}(q(S_0|\vec{X})||p_0(S_0)) \\
&-\mathrm{KL}(q(\vec{\beta}|\vec{X},E,S_0)||p_0(\vec{\beta}|E,S_0)) \\
&\textit{(where the last term vanishes since } \vec{S}_0|S_0,E,\vec{\beta} \textit{ is deterministic)} \\
&= -(\mathrm{KL}(q(E|\vec{X})||p_0(E)) \\
&+\mathrm{KL}(q(S_0|\vec{X})||p_0(S_0)) \\
&+\sum_{i=1}^{T} \mathop{\mathbb{E}}_{q(\beta_i,E,S_{i-1}|\vec{x})} \mathrm{KL}(q(\beta_i|X_i,E,S_{i-1})||p_0(\beta))) \\
&\textit{(where we have } p_0(\beta_i|E,s_i) = p_0(\beta) \textit{ by design)}
\end{aligned}
$$

A.0.3 Architecture details of state update model

In our experiments, the parameteric models for initial state encoding ϕ_S^{enc} and environment modelling ϕ_E^{enc} are implemented as bi-directional LSTM followed by a multilayer perceptron to convert LSTM-based sequence embedding into S_0 and E. However, f is implemented as an LSTM cell (elaborated by following equations, taken from (Graves, 2013).

$$
\begin{aligned}
r_i &= \sigma(W_{Dr}E + W_{Sr}S_{i-1} + W_{cr}c_{i-1} + b_r) \\
l_i &= \sigma(W_{Dl}E + W_{Sl}S_{i-1} + W_{cl}c_{i-1} + b_l) \\
c_i &= l_i \circ c_{i-1} + r_i \circ \tanh(W_{Dc}E + W_{Sc}S_{i-1} + b_c) \\
o_i &= \sigma(W_{Do}D + W_{So}S_{i-1} + W_{co}c_i + b_o) \\
S_i^{-} &= o_i \circ \tanh(c_i)
\end{aligned}
$$

A.0.4 Evaluation tool for videos generation.

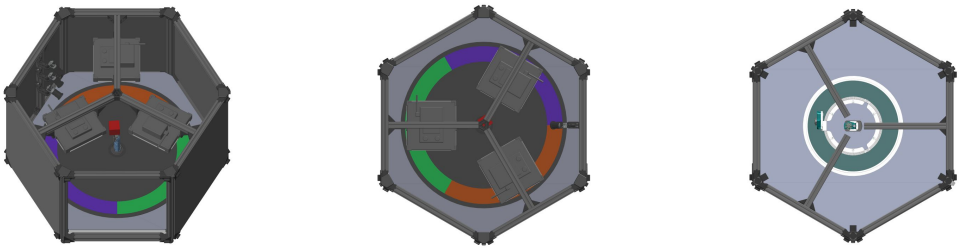
We use OpenCV's (Itseez, 2015) inbuilt functions to detect the pixel level positions of the ball in the images.

```
import cv2
import imutils
def find_positions(image):
    ret, binary_mask = cv2.threshold(image, 0.01, 1, cv2.THRESH_BINARY)
    binary_mask = cv2.erode(binary_mask, None, iterations=1)
    binary_mask = cv2.dilate(binary_mask, None, iterations=1)
    fake_frame = cv2.convertScaleAbs(binary_mask.copy())
    cnts = cv2.findContours(fake_frame,
                            cv2.RETR_EXTERNAL,
                            cv2.CHAIN_APPROX_SIMPLE)
    cnts = imutils.grab_contours(cnts)
    c = max(cnts, key=cv2.contourArea)
    ((x, y), radius) = cv2.minEnclosingCircle(c)
    return x, y
```

Appendix B

Appendix to Chapter 3

B.1 Platform for capturing MPI3D dataset



(a) Side view of platform. (b) Top view of platform. (c) Bottom view of platform.

Figure B.1: **Different rendered views of the recording setup. (a):** View from 30° angle from the top, showing the manipulator with a red cube in the middle. **(b):** View from the top showing the three light centered light panels. **(c):** View from the bottom, showing the rotating table (turquoise ring) and the manipulator in the center.

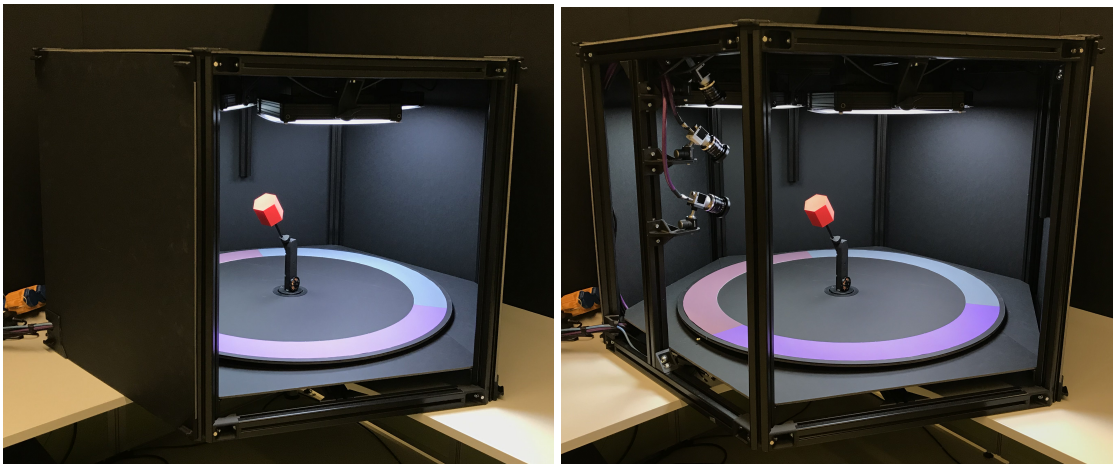


Figure B.2: The mechanical platform for recording the real-world dataset.

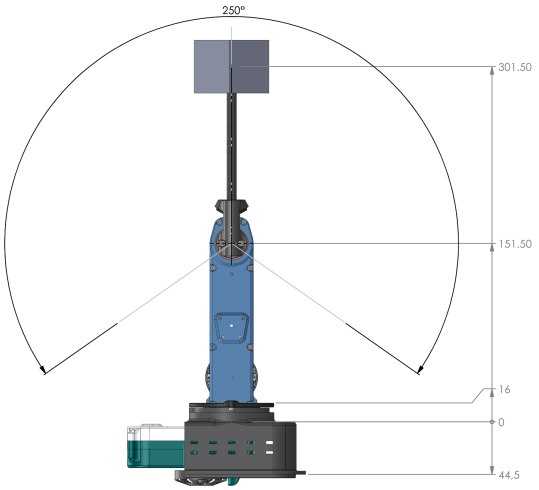


Figure B.3: A technical drawing of the manipulator, dimensions are given in mm.

B.2 Details of the experimental protocol

Metrics: Various methods to validate a learned representation for disentanglement based on known ground truth generative factors $\boldsymbol{G}$ have been proposed (e.g. Eastwood and Williams, 2018; Ridgeway and Mozer, 2018; Chen *et al.*, 2018; Kim and Mnih, 2018). This has for example been expressed as the mutual information of a single latent dimension Z_i with generative factors $G_1, \ldots, G_K$ Ridgeway and Mozer (2018), where in the ideal case each Z_i has some mutual information with one generative factor G_k but none with all the others. Similarly, Eastwood and Williams (2018) trained predictors (e.g., Lasso or random forests) for a generative factor G_k based on the representation $\boldsymbol{Z}$. In a disentangled model, each dimension Z_i is only useful (i.e., has high feature importance) to predict one of those factors. Suter *et al.* (2019) proposed an interventional robustness score. The graphical model of Suter *et al.* (2019) adapted to our setup is illustrated in Figure 3.2. Another form of validation, especially without known generative factors is the visual inspection of "latent traversals" (see e.g. Chen *et al.*, 2018).

Architecture. All the models used the same convolutional encoder and decoder architecture with the fixed latent size of 10.

Training hyperparameters. The training hyperparameters were kept fixed for each of the considered methods. Like (Locatello *et al.*, 2018), we keep the hyperparameters for all the models consistent across all datasets. In Table B.3 we provide the values of training hyperparameters that were the same across the methods and datasets. In Table B.1, we provide model-specific hyperparameters. Note that configuration of the models is the same as in (Locatello *et al.*, 2018).

Model	Parameter	Values
β-VAE	β	[1, 2, 4, 6, 8]
AnnealedVAE	c_{max}	[5, 10, 25, 50, 75]
	iteration threshold	100000
	γ	1000
FactorVAE	γ	[10, 20, 30, 40, 50]
DIP-VAE-I	λ_{od}	[1, 2, 5, 10, 20]
	λ_d	$10\lambda_{od}$
DIP-VAE-II	λ_{od}	[1, 2, 5, 10, 20]
	λ_d	λ_{od}
β-TCVAE	β	[1, 2, 4, 6, 8]

Table B.1: Hyperparameters corresponding to different methods.

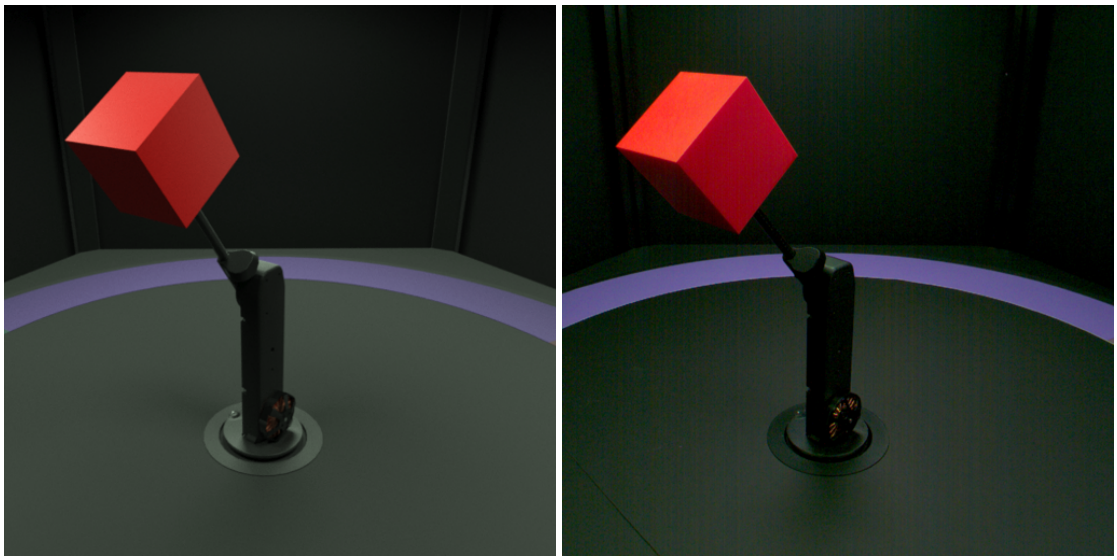
Encoder	**Decoder**
Input: $64 \times 64 \times$ number of channels	Input: 10
4×4 conv, 32 ReLU, stride 2	FC, 256 ReLU
4×4 conv, 32 ReLU, stride 2	FC, $4 \times 4 \times 64$ ReLU
4×4 conv, 64 ReLU, stride 2	4×4 upconv, 64 ReLU, stride 2
4×4 conv, 64 ReLU, stride 2	4×4 upconv, 32 ReLU, stride 2
FC 256, F2 2×10	4×4 upconv, 32 ReLU, stride 2
	4×4 upconv, number of channels, stride 2

Table B.2: Encoder and Decoder architecture for the main experiment. All the models studied shared the same architecture.

Parameter	**Values**
Batch size	64
Latent space dimension	10
Optimizer	Adam
Adam: beta1	0.9
Adam: beta2	0.999
Adam: epsilon	1e-8
Adam: learning rate	0.0001
Decoder type	Bernoulli
Training steps	300000

Table B.3: Common hyperparameters for training.

B.2.1 Comparing realistic and real-world images



(a) Realistically simulated sample. (b) Real-world sample.

Figure B.4: Comparison of the realistically simulated and the real-world dataset for one example. The detailed procedure and the availability of CAD files used e.g. for the 3D printed robotic arm ensures a close overlap between both examples. Only small details and a high resolution show the differences e.g. the crack in the floor in the left corner which is only visible in the right image or shadings from the light on the object. Both pictures have a resolution of 512×512.

B.3 Detailed experimental results

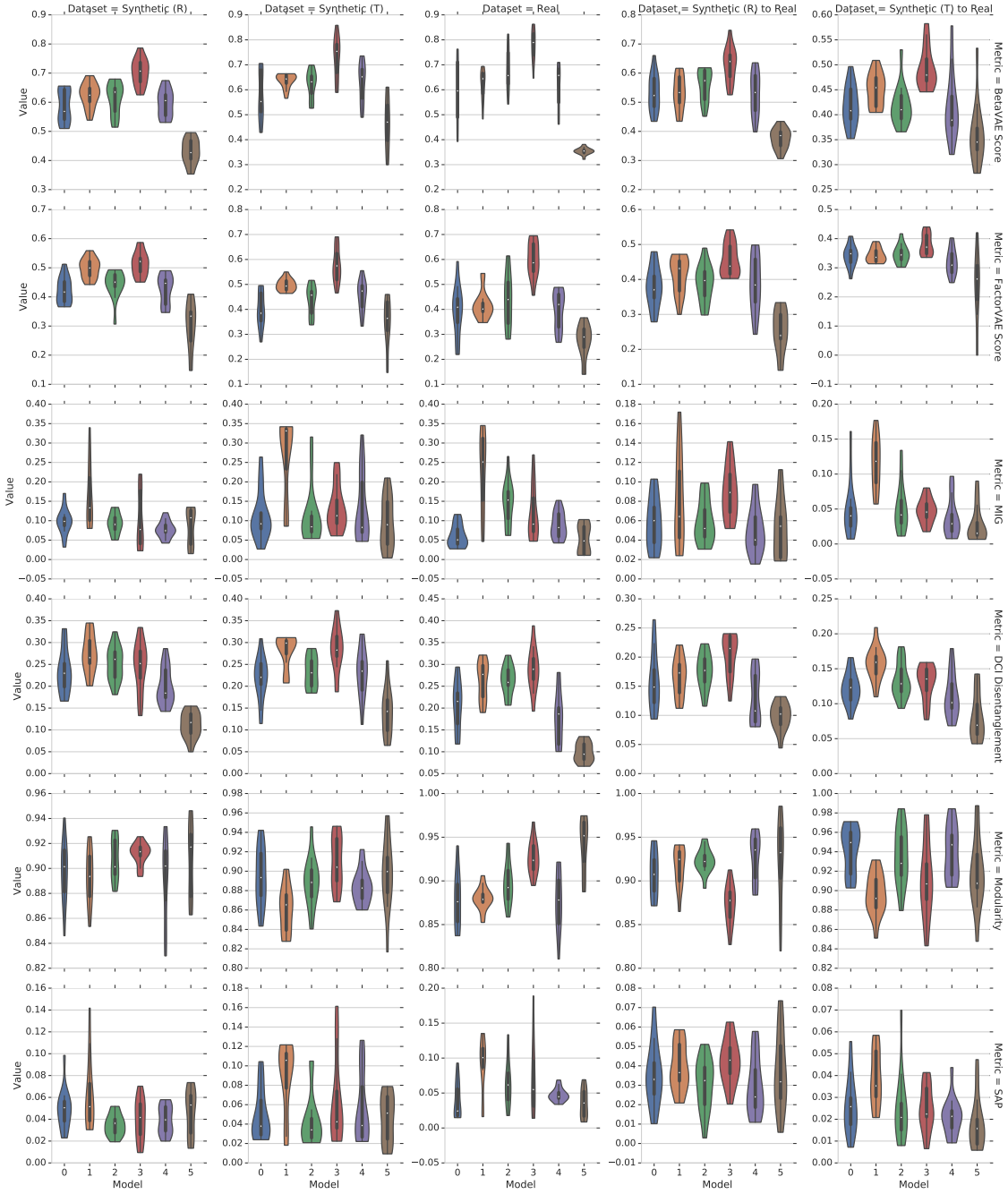


Figure B.5: Disentanglement scores attained by different methods for different metrics (row) and evaluations (column, from left to right): (a) trained and evaluated on synthetic realistic, (b) trained and evaluated on synthetic toy, (c) trained and evaluated on real, (d) trained on synthetic realistic and evaluated on real, (e) trained on synthetic toy and evaluated on real. Methods are abbreviated (0=β-VAE, 1=FactorVAE, 2=β-TCVAE, 3=DIP-VAE-I, 4=DIP-VAE-II, 5=AnnealedVAE).

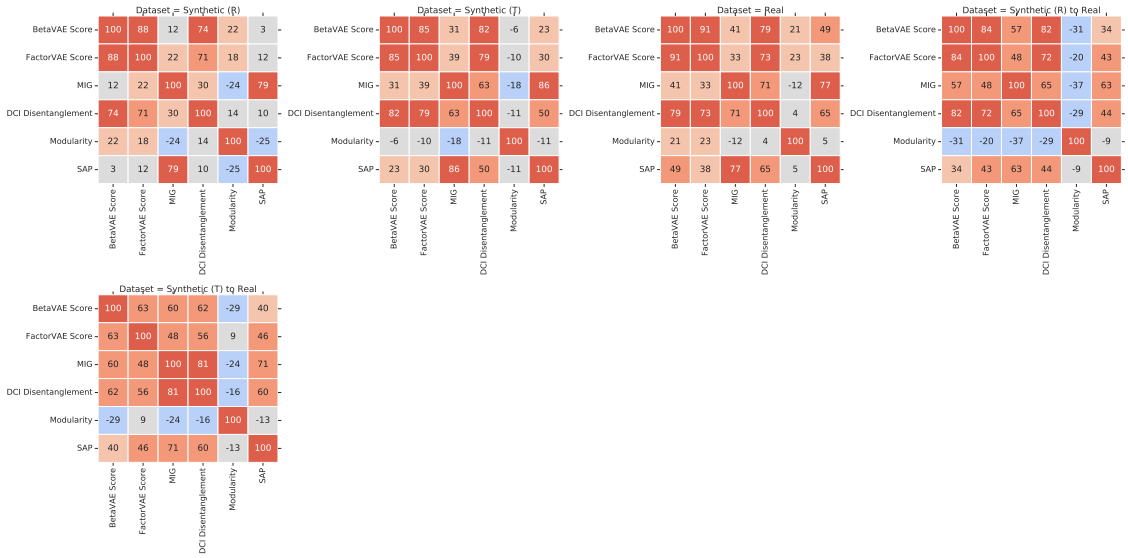


Figure B.6: Rank correlation of different metrics on different data sets. Overall, we observe that all metrics except for Modularity seem to be at least mildly correlated on all data sets.

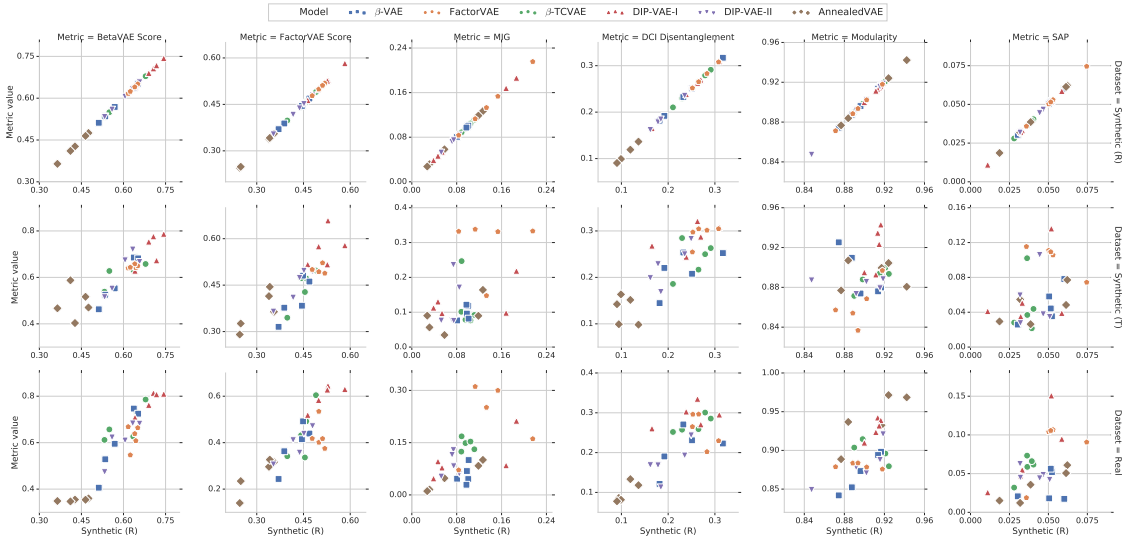


Figure B.7: Disentanglement scores on real data vs simulated datasets. There seems to be a positive correlation for at least three metrics.

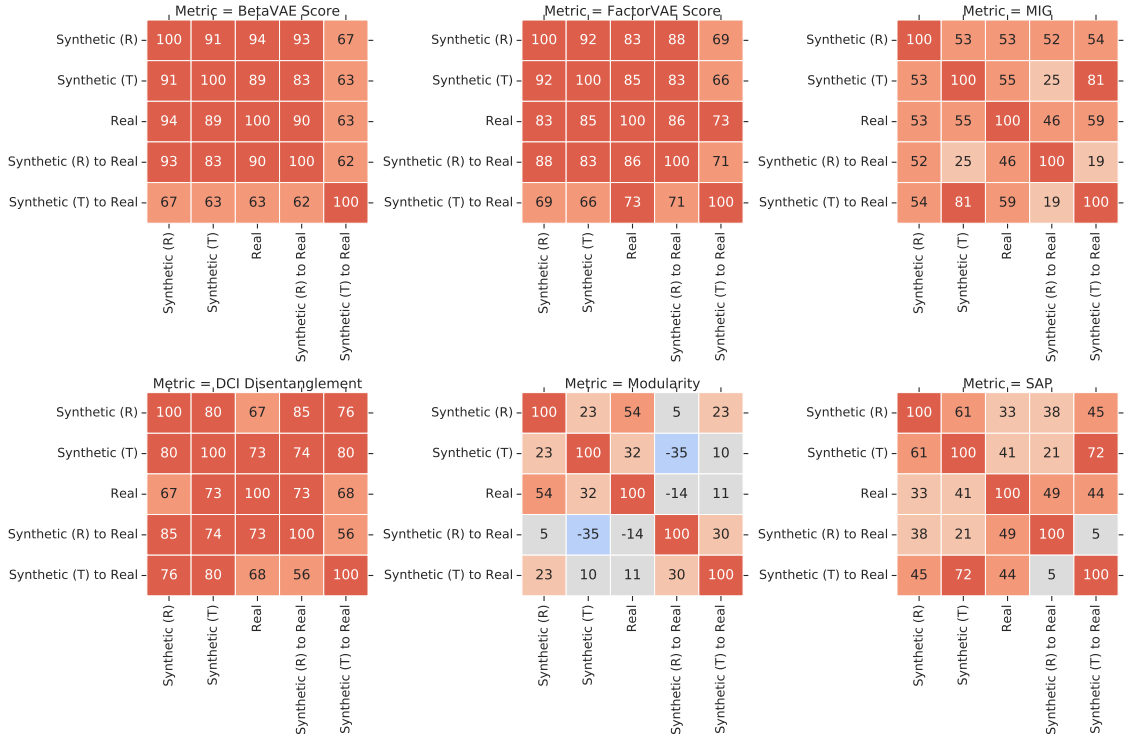


Figure B.8: Rank-correlation of different disentanglement metrics across different data sets. Good hyperparameters seem to transfer well between datasets according to at least three metrics.

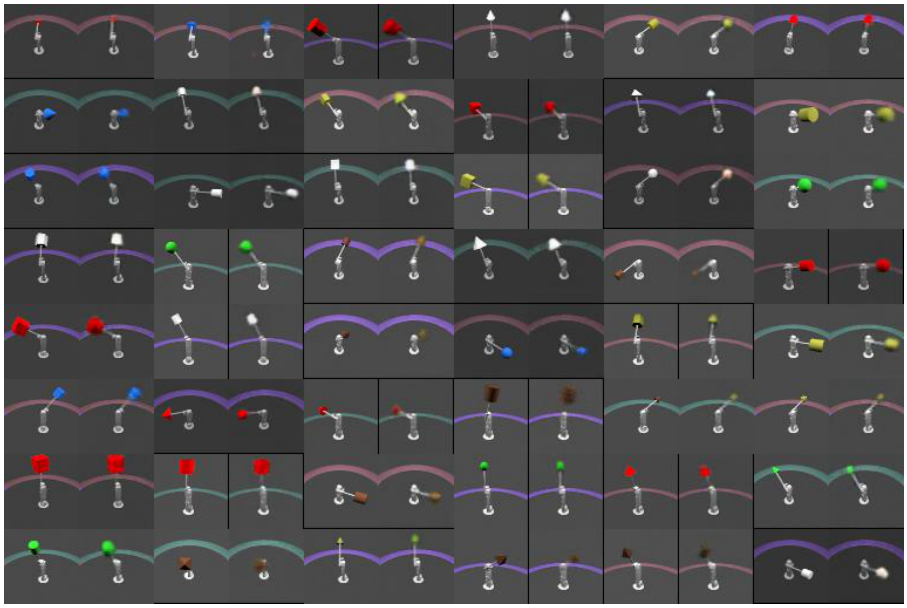


Figure B.9: Image reconstruction of Factor VAE model on the low quality simulated dataset.

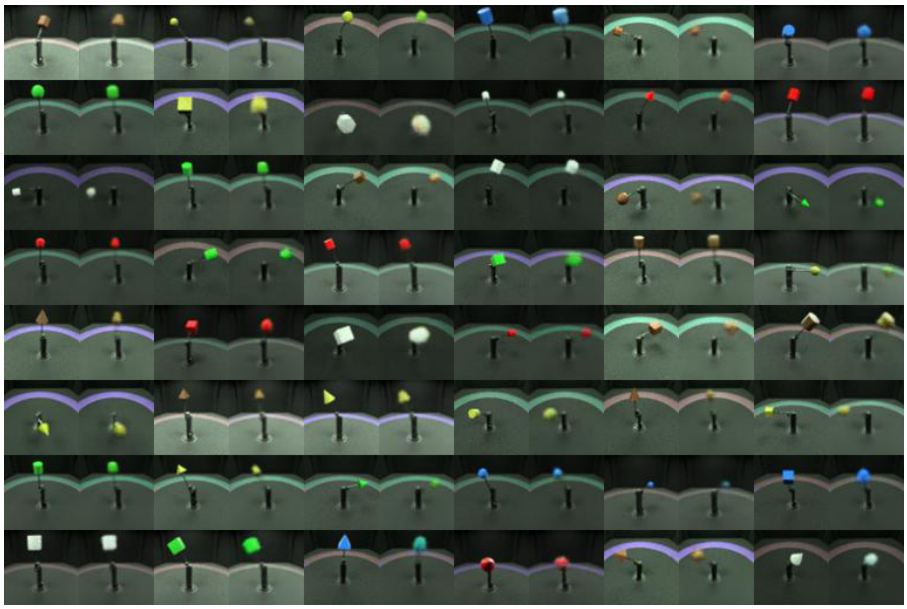


Figure B.10: Image reconstruction of Factor VAE model on the realistic simulated dataset.

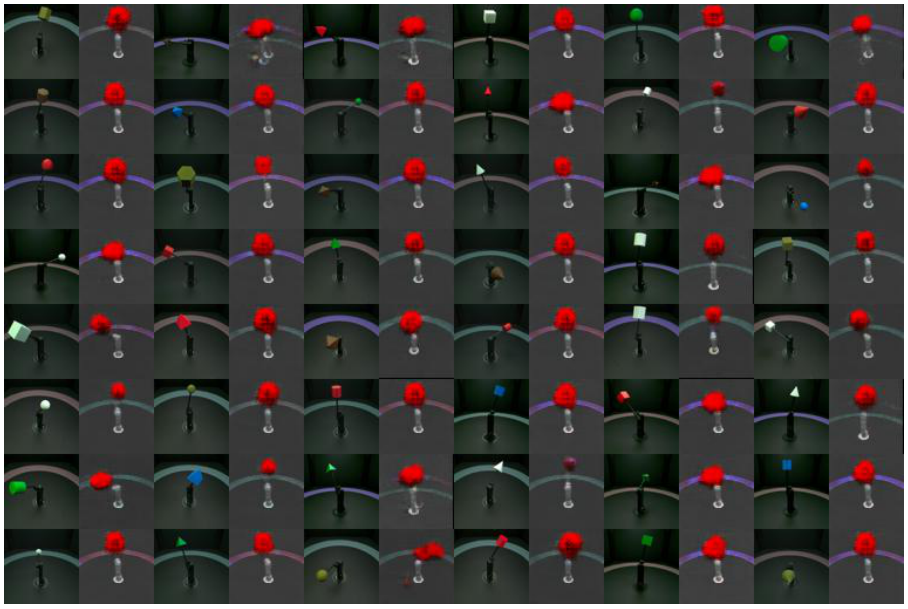


Figure B.11: Image reconstructions of the Factor VAE model trained on toy dataset and tested on the real-world dataset. All images in the uneven columns are real, and to the right of each real image is its reconstruction.

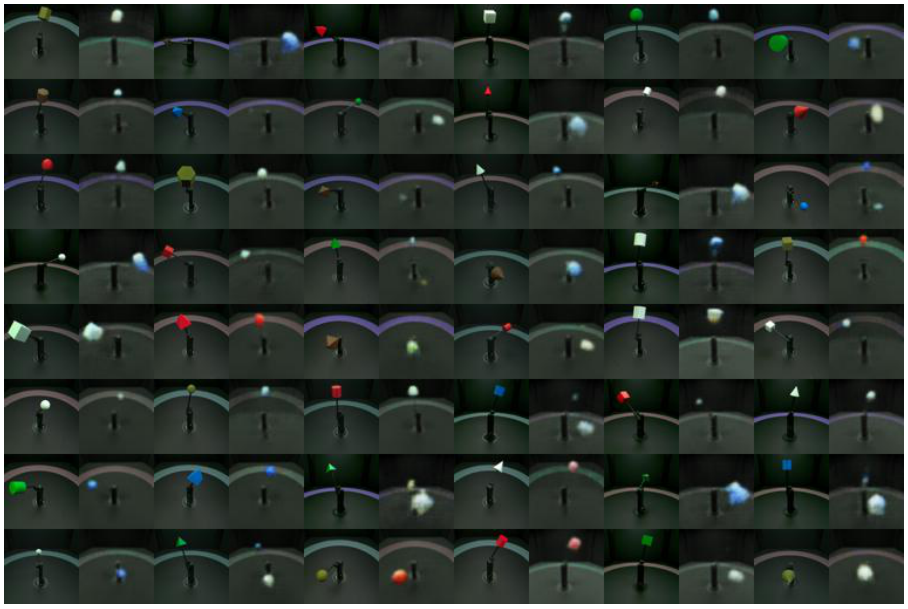


Figure B.12: Image reconstructions of the Factor VAE model trained on realistic simulated dataset and tested on the real-world dataset. All images in the uneven columns are real, and to the right of each real image is its reconstruction.

Appendix C

Appendix to Chapter 4

C.1 Sampling fuzzy boolean functions

In this section, we define a family of smooth functions mapping from the unit hyper-cube $[0,1]^N$ to $[0,1]$. To this end, consider again the primitives defined in Equation (4.15). Where $x_i, x_j \in [0,1]$, we define:

$$\texttt{and}(x_i, x_j) = x_i x_j \tag{C.1}$$

$$\texttt{not}(x_i) = \bar{x}_i = 1 - x_i \tag{C.2}$$

$$\texttt{or}(x_i, x_j) = x_i \oplus x_j = 1 - (1 - x_i)(1 - x_j) \tag{C.3}$$

Observe that if $x_i, x_j \in \{0,1\}$, these operations reduce to their Boolean namesakes, and Equation (C.3) is consistent with de Morgan's law. In this sense, the primitives described above induce a *relaxation* of Boolean logic to real numbers on the compact interval $[0,1]$. We note that this relaxation, called product fuzzy logic, is not unique: there exist other definitions of the `and` and `not` operations that define other logics (examples being Łukasiewicz and Gödel-Dummett logics).

Given these primitives, it is now possible to construct functions that resemble boolean functions in the cannonical disjunctive normal form (i.e., in the sum-of-products form). As an example, consider a vector $\mathbf{x} \in [0,1]^5$, whose components we call $a, b, c, d, e \in [0,1]$. One may now define a function:

$$f(\mathbf{x}) : [0,1]^5 \to [0,1],\ (a,b,c,d,e) \mapsto \bar{a}bcd\bar{e} \oplus a\bar{b}cd\bar{e} \oplus ab\bar{c}d\bar{e} \tag{C.4}$$

If a, b, c, d, e were to be boolean (i.e., $\in \{0,1\}$), the function f would have a truth table where $f = 1$ only if $a = 0, b = 1, c = 1, d = 1, e = 0$, or $a = 1, b = 0, c = 1, d = 0, e = 1$, or $a = 1, b = 1, c = 0, d = 1, e = 0$. Conversely, given this truth table, it is possible to reconstruct f in the sum-of-products form described above.

The above fact makes randomly sampling a fuzzy boolean function like sampling from the Bernoulli distribution: for all combinations of possible values of $a, b, c, d, e \in \{0,1\}^5$, we sample the value of a boolean function $f(a,b,c,d,e) \sim \text{Bernoulli}(0.5)$ in order to populate the truth-table of f. Given the randomly sampled truth table, we construct the

Parameters	Values
Batch size	128
Pretraining epochs	20
Finetuning epochs	3
Dimension of code vector (**c**)	128
Dimension of intermediate features	128
Number of scripts (n_s)	2
Number of function Iterations (n_i)	2
Number of LOCs (n_l)	1
Number of functions (n_f)	4
Number of heads per LOC	1
Number of features per LOC head	32
Type Inference MLP Depth	2
Type Inference MLP Width	128
Frozen Function Signatures	`False`
Frozen Function Codes	`False`
Truncation Parameter (τ)	1.6
Type Space Dimension (d_{type})	24
Optimizer	RAdam (Liu *et al.*, 2019)
Adam: learning rate (pre-training)	0.006
Adam: learning rate (finetuning)	0.05
Adam: β_1	0.9
Adam: β_2	0.999
Adam: ε	1e-8
Learning rate scheduler	None

Table C.1: Hyperparameters for learning fuzzy boolean expressions.

expression for f in the sum-of-product form. Finally, we interpret the boolean expression (mapping from $\{0,1\}^5 \to \{0,1\}$) as a fuzzy boolean expression mapping from $[0,1]^5 \to [0,1]$ using the corresponding primitives defined in Equation (C.1) et seq.

C.1.1 Hyperparameters

Please refer to Table C.1.

C.2 Multi-task image classification

C.2.1 The digits dataset

Figure C.1: Augmented samples from the digits dataset.

The Digits dataset is a concatenation of three datasets of labelled images of digits: SVHN (Netzer *et al.*, 2011), MNISTM (Ganin *et al.*, 2016), and MNIST (LeCun *et al.*, 2010). All images are up-sampled to RGB images of size 32×32, and the combined training set has 193257 samples, whereas the validation set has 46032 samples. In addition to the images and labels, we also preserve information about which of the constituent datasets a sampled image originates from.

We use RandAugment (Cubuk *et al.*, 2019) to augment the input images before feeding them to the model, and use the implementation from Pytorch Image Models (Wightman, 2019). Figure C.1 shows augmented samples from the dataset.

C.2.2 Hyperparameters

Pre-training. Table C.2 shows the hyperparameters used for pre-training the Neural Interpreter model considered in Figure 4.4. Table C.3 shows the same, but for the Vision Transformer model.

Finetuning. Both models shown in Figure 4.4 were fine-tuned for 10 epochs with varying number of samples. We used the same batch-size as in pre-training (128). The error bands are with respect to 6 random seeds, where the random seed also determines the subset of K-MNIST that was used. We used RAdam optimizer with a constant learning

Parameters	**Values**
Batch size	128
Pre-training epochs	100
Dimension of code vector (**c**)	192
Dimension of intermediate features	192
Number of scripts (n_s)	1
Number of function iterations (n_i)	8
Number of LOCs (n_l)	1
Number of functions (n_f)	5
Number of heads per LOC	4
Number of features per LOC head	128
Type Inference MLP Depth	2
Type Inference MLP Width	192
Frozen Function Signatures	`True`
Frozen Function Codes	`False`
Truncation Parameter (τ)	1.4
Type Space Dimension (d_{type})	24
Optimizer	RAdam (Liu *et al.*, 2019)
Adam: β_1	0.9
Adam: β_2	0.999
Adam: ε	1e-8
Learning rate scheduler	Cosine (no warm-up)
Scheduler: $\eta_{\max}$ (Max LR)	0.0008
Scheduler: $\eta_{\min}$ (Min LR)	0.000001
Scheduler: Number of decay steps	120000
Number of parameters	6.43×10^5
Accuracy on SVHN	96.2 %
Accuracy on MNISTM	98.4 %
Accuracy on MNIST	99.4 %

Table C.2: Hyperparameters for pre-training the Neural Interpreter model used in Figure 4.4.

Parameters	**Values**
Batch size	128
Pretraining epochs	100
Dimension of intermediate features	192
Number of MLP Features	192
Depth	8
Number of heads	3
Number of features per head	64
Optimizer	RAdam (Liu *et al.*, 2019)
Adam: β_1	0.9
Adam: β_2	0.999
Adam: ε	1e-8
Learning rate scheduler	Cosine (no warm-up)
Scheduler: $\eta_{\max}$ (Max LR)	0.0008
Scheduler: $\eta_{\min}$ (Min LR)	0.000001
Scheduler: Number of decay steps	120000
Number of parameters	1.80×10^6
Accuracy on SVHN	96.3 %
Accuracy on MNISTM	98.3 %
Accuracy on MNIST	99.6 %

Table C.3: Hyperparameters for pre-training the ViT Model used in Figure 4.4.

rate, which was found with a grid search (0.03 for ViT and 0.05 for NI). For the results shown in Figure 4.4 (right), the function codes and signatures were trained for 10 epochs on 8192 samples with Shampoo (Gupta *et al.*, 2018). We again used 6 random seeds, and for each set of trainable parameters, we grid-searched the learning rate. We did not see good performance with RAdam in this particular setting, suggesting that the loss landscape might necessitate the pre-conditioning that is present in Shampoo (but not in RAdam).

Positional encoding. We use a variant of the relative positional encoding scheme presented in (Cordonnier *et al.*, 2019), which we now describe. Consider an array $\mathbf{X}$ of shape $C \times H \times W$, where C is the number of channels, and H and W can be interpreted as height and width. Note that the array $\mathbf{X}$ need not be an image; it could (for instance) be a collection of embedding vectors of patches, i.e., $\mathbf{X}_{:,ij}$ could be the embedding vector (of dimension C) of the patch that is i-th from top and j-th from left.

We now denote with $\mathbf{e}_{\text{row}}[i_2 - i_1]$ a vector that is a learned embedding of the difference of row-indices i_2 and i_1. Likewise, we let $\mathbf{e}_{\text{col}}[j_2 - j_1]$ be a learned embedding of the

difference of column-indices j_2 and j_1. Where h indexes attention heads, we define:

$$e^h_{\text{row}}[i_2 - i_1] = \mathbf{w}^h_{\text{row}} \cdot \mathbf{e}_{\text{row}}[i_2 - i_1] \tag{C.5}$$
$$e^h_{\text{col}}[j_2 - j_1] = \mathbf{w}^h_{\text{col}} \cdot \mathbf{e}_{\text{col}}[j_2 - j_1] \tag{C.6}$$

Here, $\mathbf{w}^h_{\text{row}}$ and $\mathbf{w}^h_{\text{row}}$ are learned weight vectors, and $e^h_{\text{row}}[i_2 - i_1]$ and $e^h_{\text{col}}[i_2 - i_1]$ are scalars, one per attention head. This set-up allows each attention head to develop a positional bias independently from other heads, a feature we inherit from (Cordonnier *et al.*, 2019). In the context of Neural Interpreters, we additionally allow functions to have their own positional bias, conditioned on its code $\mathbf{c}_u$. We have:

$$p^{uh}_{\text{row}}[i_2 - i_1] = \text{ModLin}^h_{\text{row}}(\mathbf{e}_{\text{row}}[i_2 - i_1]; \mathbf{c}_u) \tag{C.7}$$
$$p^{uh}_{\text{col}}[j_2 - j_1] = \text{ModLin}^h_{\text{col}}(\mathbf{e}_{\text{col}}[j_2 - j_1]; \mathbf{c}_u) \tag{C.8}$$

Here, $p^{uh}_{\text{row}}[i_2 - i_1]$ and $p^{uh}_{\text{col}}[j_2 - j_1]$ are scalars specific to function u and attention head h. Finally, the overall positional bias is given as following, where broadcasting operations are implied:

$$b^{uh}[i_2 - i_1, j_2 - j_1] = (p^{uh}_{\text{row}}[i_2 - i_1] + e^h_{\text{row}}[i_2 - i_1]) + (p^{uh}_{\text{col}}[j_2 - j_1] + e^h_{\text{col}}[j_2 - j_1]) \tag{C.9}$$

Here, $b^{uh}[i_2 - i_1, j_2 - j_1]$ is the positional bias that is added to the pre-softmax dot-product attention weights coupling the embedding vectors $\mathbf{X}_{:,i_1 j_1}$ and $\mathbf{X}_{:,i_2 j_2}$ at function f_u and attention head at index h. We remark that this scheme only differs from (Cordonnier *et al.*, 2019) in that we allow each function to develop its own positional bias.

C.2.3 Additional results and ablations

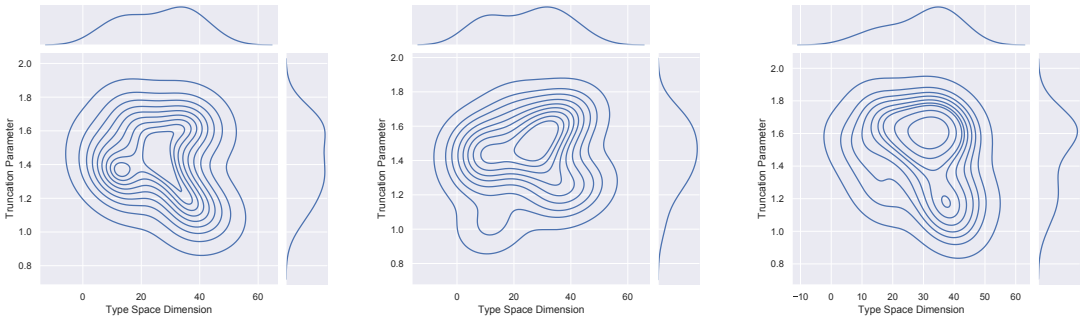
In order to understand the effect of various hyperparameters, we analyze the results of a random sweep over 100 runs on the Digits dataset. The distributions over sweep parameters are presented in Table C.4.

Kernel truncation and dimension of type space. In Figure C.2, we select for each dataset the top 10% of all runs (w.r.t. validation performance), and plot a Kernel Density Estimate of their type space dimensions (d_{type}) and truncation parameters (τ). We find that while the optimal τ and d_{type} only somewhat depend on each other, there are minor variations between the SVHN and MNIST-M vs. MNIST. We speculate that this is due to SVHN and MNIST-M having cluttered backgrounds; the flexibility afforded by a larger type space is less desirable when the model must learn to suppress background clutter by routing noisy patches through similar functions.

Parameters	**Distribution**
Truncation parameter (τ)	$\mathcal{U}([0.7, 1.7])$
Dimension of type space (d_{type})	$\mathcal{U}(\{4 * i \mid i \in \{2, 3, ..., 12\}\})$
Number of functions (n_f)	$\mathcal{U}(\{1, 2, 3, 4, 5\})$
Num. of scripts (n_s), function iterations (n_i), and LOCs (n_l)	$\mathcal{U}(\{(2, 2, 2), (2, 4, 1), (4, 2, 1), (1, 8, 1)\})$
Frozen function signatures	$\mathcal{U}(\{\texttt{True}, \texttt{False}\})$
Frozen function codes	$\mathcal{U}(\{\texttt{True}, \texttt{False}\})$
Frozen patch embeddings	$\mathcal{U}(\{\texttt{True}, \texttt{False}\})$

Table C.4: Distribution over hyperparameters used in the sweep. $\mathcal{U}$ denotes the uniform distribution.

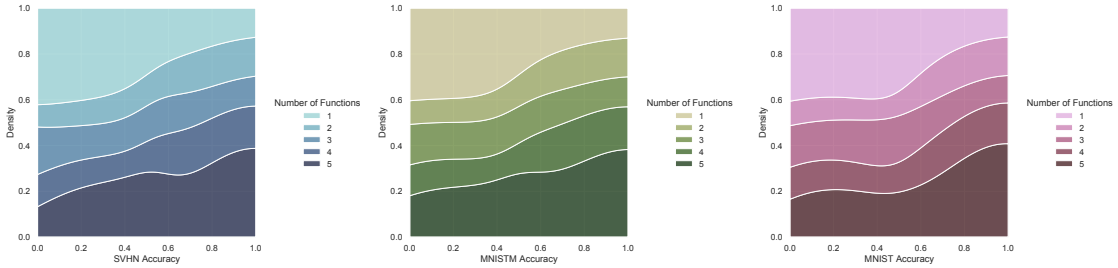
Number of functions. Figure C.3 shows the kernel density estimate of the validation performance of *all* 100 runs, conditioned on the number of functions. We read that on the one hand, runs with 5 functions perform consistently well; on the other hand, there exist runs with a single function that perform well, but most of these runs fail. This is consistent with the recommendation in Section 4.5.



(a) SVHN (b) MNIST-M (c) MNIST

Figure C.2: Kernel Density Estimates of truncation parameters τ and type space dimensions d_{type} of the top 10% of runs for each dataset.

Frozen patch Embeddings, function signatures and codes. Figure C.4 shows the validation performance of top 10% of runs (for the respective dataset), with or without frozen patch embeddings, function signatures and codes. As elaborated in Section 4.5, it is not surprising that Neural Interpreters can work well even when function codes and signatures remain frozen during training. We find that freezing function signatures can be marginally beneficial, but freezing function codes less so. We also experimented with



(a) SVHN (b) MNIST-M (c) MNIST

Figure C.3: Conditional Kernel Density Estimates of validation performance, conditioned on the number of functions n_f.

freezing the patch embeddings, as recommended in (Touvron *et al.*, 2021a; Chen *et al.*, 2021), and find that it slightly improves performance.

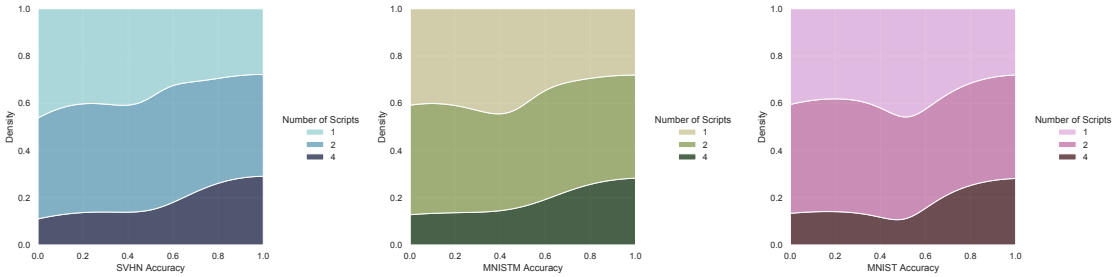
(a) Frozen Function Signatures (b) Frozen Function Codes (c) Frozen Patch Embeddings

Figure C.4: Box plots of validation performance of top 10% of all runs.

Number of scripts, function Iterations and LOCs. Figure C.7 shows the conditional kernel density estimates of validation performance, conditioned on the number of scripts. We again find that all evaluated configurations can work well. A larger number of scripts can stabilize training and lead to consistent in-distribution performance, as expected from Section 4.5.

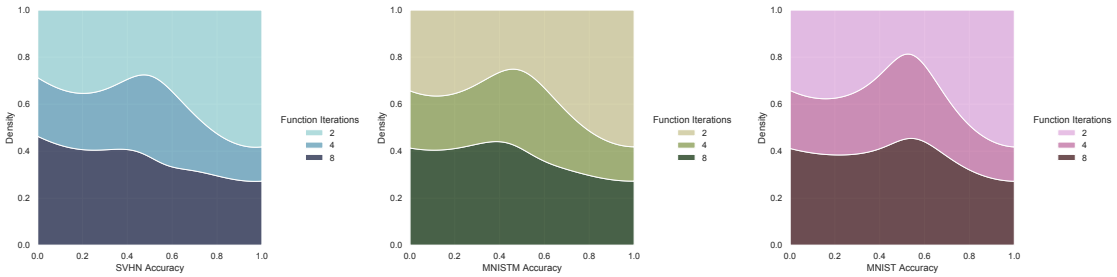
C.3 Abstract reasoning with PGMs

Progressively Generated Matrices (PGMs) (Barrett *et al.*, 2018) have been used as a diagnostic dataset to study the compositional generalization capability of machine learning models (Wang *et al.*, 2020; Steenbrugge *et al.*, 2018). The dataset consists of complex visual analogical reasoning tasks that require relational reasoning between attributes of different objects. The 'object types' include shape and line comprising of the 'attribute



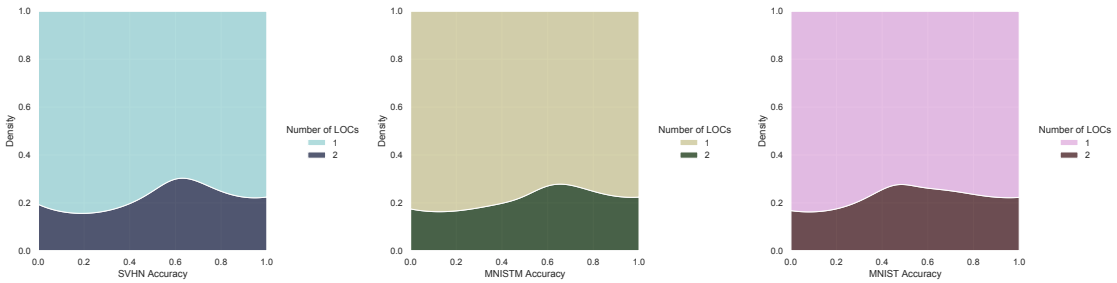
(a) SVHN (b) MNIST-M (c) MNIST

Figure C.5: Conditional Kernel Density Estimates of validation performance, conditioned on the number of scripts n_s.



(a) SVHN (b) MNIST-M (c) MNIST

Figure C.6: Conditional Kernel Density Estimates of validation performance, conditioned on the number of function iterations n_i.



(a) SVHN (b) MNIST-M (c) MNIST

Figure C.7: Conditional Kernel Density Estimates of validation performance, conditioned on the number of LOCs n_l.

types': size, type, color, position, and number, where each attribute can take one of a finite number of discrete values. The 'relationship types' consist of progression, XOR, OR, AND, and consistent union. The structure of a PGM task is governed by *triples*, which are defined by applying a certain relationship type to the attributes of the objects. On an average, one to four relationships are used per task.

To study the various aspects of generalization in models, (Barrett *et al.*, 2018) introduced 8 different sub-datasets, corresponding to different generalization regimes of compositional reasoning. Except for the *Neutral* regime, the test dataset in each regime measures the out-of-distribution generalization i.e., the test and the training datasets are different in a clearly defined manner. We use 6 of such regimes in this work, namely: Interpolation, Extrapolation, Held-out (H.O.) triples, H.O. pairs of triples, H.O. Attribute Pairs, and Neutral. The details on these regimes are provided below.

C.3.1 Generalization regimes

Neutral. The neutral regime measures the in-distribution generalization, i.e., the training and the test sets consist of any triples.

Interpolation. In the training dataset of interpolation regime, the discrete values of the attributes are restricted to even numbers whereas the test set consists of odd-valued attributes.

Extrapolation. For the training dataset of extrapolation regime, the attribute values were restricted to the lower half of the discrete set whereas the test set consists of values sampled from the upper half of the discrete set.

Held-out Triples. The PGM dataset contains 29 unique triples. In the test set of held-out triples, 7 of such triples were held-out, while the rest of the triples are used to create the training dataset.

Held-out Pairs of Triples. All tasks contain at least two triples, leading to 400 viable triple pairs (Barrett *et al.*, 2018). In *Held-out Pairs of Triples*, 360 such pairs are randomly allocated to the training dataset and rest to the test dataset.

Held-out Attribute Pairs. Here, each task consists of at least two triples, where there are 20 viable pairs of attributes. Of these 20 pairs, 16 have been used to create the training set while the remaining 4 are used in the test set.

Parameters	Ranges
Kernel truncation parameter (τ)	$[1.3, 1.7]$
Type features dimensions	$[20, 24, 28, 32, 36, 40]$
Detach Function Signatures	[`True`, `False`]
Number of scripts (n_s)	$[1, 2, 4]$
Function Iterations	$[4, 8, 16]$

Table C.5: Hyperparameters whose values are randomly sampled from the given ranges for each experiment.

C.3.2 Details of PGM experiments

For each PGM sub-dataset, we train multiple models for both Vision Transformers and Neural Interpreters. Each model is trained for 30 epochs and the model selection is done by evaluating its performance on validation datasets. The reported test accuracy in Table 4.2 corresponds to the best validation performance. We perform hyper-parameter sweeps to find the best configuration of Neural Interpreters in each regime.

Hyperparameter settings

We perform random sweeps to find the optimal hyperparameters for each PGM regime. Due to the huge computational overload and the massive size of the datasets, the number of experiments in each sweep is limited to 35. We carried forward the knowledge that we learned from the digits experiments (Section 4.4.2), and perturbed only those hyperparameters that had significant influence on the model's performance. Apart from changing these selected hyperparameters, the models are identical in all aspects. Table C.6 provides the hyperparameters that are kept the same in all the models, whereas Table C.5 shows the hyperparameters that we perturb and the ranges from which their values are randomly sampled.

For the sake of consistency, we make sure that the number of computational steps remain the same in all the experiments. For PGM sweeps, the number of computational steps are set to be 16. We varied the numbers of scripts n_s and function iterations n_i such that their product comes out to be 16.

Optimal Neural Interpreters configuration for PGMs

After running the set of experiments, we found that the one configuration that outperformed all other configurations was with 2 scripts i.e., $n_s = 2$ and 8 function iterations. There are small fluctuations in the selection of kernel truncation parameter τ and dimensions of type space d_{type} that we detail below in Table C.7.

Parameters	**Values**
Batch size	72
Epochs	30
Dimension of code vector (**c**)	192
Dimension of intermediate features	192
Number of scripts (n_s)	2
Number of function iterations (n_i)	8
Number of LOCs (n_l)	1
Number of functions (n_f)	5
Number of heads per LOC	4
Number of features per LOC head	32
Type Inference MLP Depth	2
Type Inference MLP Width	192
Variable Features dimensions	192
Frozen Function Codes	`False`
Optimizer	RAdam
Adam: beta1	0.9
Adam: beta2	0.999
Adam: epsilon	1e-8
Adam: learning rate	0.0004
Learning Rate Scheduler	Cosine
Cosine Scheduler Eta Max	0.0004
Cosine Scheduler Eta Min	0.0001
Number of parameters	1.6M

Table C.6: Hyperparameters and their values that are kept the same in all the PGM experiments.

Regime	Neutral	Interp.	Attribute P.	Triple P.	Triples	Extra.
Number of scripts (n_s)	2	2	2	2	2	2
Function iterations (n_i)	8	8	8	8	8	8
τ	1.62	1.62	1.40	1.66	1.42	1.42
d_{type}	20	20	32	24	24	24
Frozen function signatures	False	False	True	False	True	True

Table C.7: Hyperparameters of Neural Interpreters for the considered PGM datasets. Here, *V-WReN* refers to VAE-WReN, *Extra.* refers to Extrapolation, *Interp.* to Interpolation, *Attribute P.* to Attribute Pairs and *Triple P.* to Triple Pairs.

Appendix D

Appendix to Chapter 5

D.1 Additional details on 1D functions

We used the same encoder architecture for our method and the baselines (Garnelo *et al.*, 2018a,b) in all experiments. For 1D and 2D functions, the data is fed in the form of input-output pairs (x, y), where x and y are 1D values. We use MLPs with two hidden layer to encode the representations of these inputs. The number of hidden units in each layer is $d = 50$. All MLPs have relu non-linearities except the final layer, which has no non-linearity.
Encoder: Input(2) $\rightarrow 2 \times$ (FC(50), ReLU) $\rightarrow$ FC(50).
While learning the representations of sinusoid functions with FCRL, we also scale the output scores with temperature to be 0.07. We used the following hyper-parameter settings to train an encoder with FCRL.

Parameter	**Values**
Batch size	256
Latent space dimension	50
Temperature: τ	0.07
Number of subsets: J	2
Max number of context points: N	20
Epochs	30
Critic	Nonlinear
Optimizer	Adam
Adam: beta1	0.9
Adam: beta2	0.999
Adam: epsilon	1e-8
Adam: learning rate	0.0003
Learning Rate Scheduler	Cosine

Table D.1: Hyperparameters to train FCRL based encoder for 1D sinusoid functions.

To learn the subsequent task-specific decoders on the representations, we adapted the

same data processing pipeline as above. For 1D functions, we train decoders for two different tasks: *few-shot regression* and *few-shot parameter identification*. The decoders for each task are trained with the same training dataset as was used to train the encoders. The training procedure for both downstream tasks on sinusoid functions is as follows

- For Few-Shot Regression (FSR), we use an MLP architecture with two hidden layers. The same architecture are used in CNP (Garnelo *et al.*, 2018a), however in CNP the decoder and encoder are trained jointly. All the baselines and our model are trained for the same number of iterations. We used slightly higher learning rate to train the decoder as the training converges quite easily.
  FSR Decoder: Input(50) $\rightarrow$ 2 $\times$ (FC(50), ReLU) $\rightarrow$ FC(1) .

- For Few-Shot Parameter Identification (FSPI), we train a linear decoder without any activation layers on the representations learned via FCRL and the baseline methods. The decoder is trained for only one epoch.
  FSPI Decoder: Input(50) $\rightarrow$ FC(1) .

Parameter	**Values**
Batch size	256
Epochs	30
Critic	Nonlinear
Optimizer	Adam
Adam: beta1	0.9
Adam: beta2	0.999
Adam: epsilon	1e-8
Adam: learning rate	0.001
Learning Rate Scheduler	Cosine

Table D.2: Hyperparameters to train FSR Decoder on FCRL learned representations.

D.2 Additional details on 2D functions

In this experiment, we treat MNIST images as 2D functions. We adapt the architectures of encoders and decoders from the previous 1D experiments. However, due to the increased complexity of the function distributions we increase the number of hidden units of MLP to $d = 128$. Moreover, the input x is 2D as it corresponds to the cartesian coordinates of an image. The hyperparameter settings to train FCRL based encoder on such 2D function is given in Table D.3.

Parameter	Values
Batch size	16
Latent space dimension	128
Temperature: τ	0.007
Number of subsets: J	40
Max number of context points: N	200
Epochs	100
Critic	Nonlinear
Optimizer	Adam
Adam: beta1	0.9
Adam: beta2	0.999
Adam: epsilon	1e-8
Adam: learning rate	0.0006
Learning Rate Scheduler	Cosine

Table D.3: Hyperparameters to train FCRL based encoder for 2D functions.

Downstream Tasks. We consider two downstream tasks to evaluate the quality of the representations learned on 2D functions: *few-shot image completion* and *few-shot content classification*. A separate decoder is trained for both of these tasks.

- For Few-Shot Image Completion (FSIC), we use an MLP based decoder with two hidden layers. The decoder is trained on the same training data for the same number of iterations. Details are given in Table D.4.
- For Few-Shot Content Classification (FSCC), we train a linear regression on top of the representations obtained by both FCRL and the baselines. The decoder is trained for only one epoch.

D.3 Additional details on scenes functions

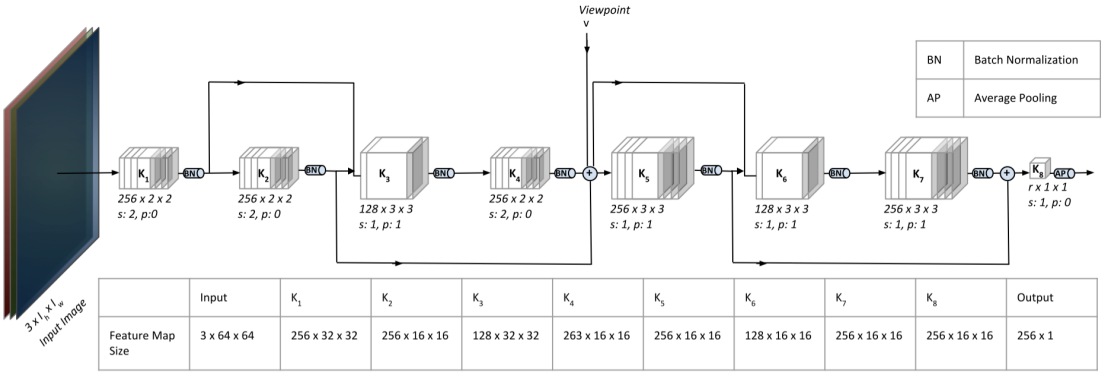
The architecture used to encode scene images is the same as "pool" architecture used in GQN. We further augment it batch normalization. Figure D.1 shows the detailed architecture used in experiments.

Learning scenes representations for MPI3D dataset

Since we have three view per each scene in MPI3D dataset, therefore we are restricted in defining the number of context points and the number of views. In all our experiments the number of context points N is fixed to three while the number of views J is

Parameter	Values
Batch size	16
Epochs	100
Critic	Nonlinear
Optimizer	Adam
Adam: beta1	0.9
Adam: beta2	0.999
Adam: epsilon	1e-8
Adam: learning rate	0.001
Learning Rate Scheduler	Cosine

Table D.4: Hyperparameters to train Few-Shot Image Completion (FSIC) Decoder trained on FCRL learned representations.



	Input	K_1	K_2	K_3	K_4	K_5	K_6	K_7	K_8	Output
Feature Map Size	3 x 64 x 64	256 x 32 x 32	256 x 16 x 16	128 x 32 x 32	263 x 16 x 16	256 x 16 x 16	128 x 16 x 16	256 x 16 x 16	256 x 16 x 16	256 x 1

Figure D.1: The ”pool” architecture used for learning representations for the scenes' datasets via FCRL. The architecture is the same as was used for training GQN (Eslami *et al.*, 2018).

set to two. In contrast to the experiments for regression datasets where more datapoints per each view resulted in better representations, the restriction to a single image view in MPI3D dataset did not hurt the representation quality, measured in terms of downstream performance. In the downstream experiments on MPI3D, we use only one image to train and validate the probes. Due to the limitation of available views, we could not measure the effect of varying the number of views on the downstream performance.

We use the GQNs pool architecture with batch normalization as encoder. As mentioned in the ablation study, we did a random sweep over the range of hyperparameters and selected the best performing model. Further details on the hyperparameters is provided in Table D.5.

Parameter	Values
Batch size	64
Representation dimension	128
Temperature: τ	0.88
Number of subsets: J	2
Max number of context points: 3	
Epochs	100
Critic	Nonlinear
Objective	NCE
Optimizer	Adam
Adam: beta1	0.9
Adam: beta2	0.999
Adam: epsilon	1e-8
Adam: learning rate	0.0005
Learning Rate Scheduler	Cosine
Number of workers	10
Batch normalization	Yes

Table D.5: Hyperparameters to train FCRL based encoder on the MPI3D Dataset.

Learning scenes representations for RLScenes dataset

To train the FCRL encoder, we randomly sample the number of views from each scene to lie within the range [2, 20]: upper-bounded by 20 to restrict the maximum number of images per scene to be the same as that used in (Eslami *et al.*, 2018), and lower bounded by 2 in case just the one view is not from a suitable angle. So, here, the maximum number of context points N is 20. The number of subsets J is set to 8. In the downstream reinforcement learning task, we use only one image to train the policy network, as is the usual practice, and the same as (Eslami *et al.*, 2018). We kept the joint ranges from which joint positions are uniformly sampled to randomly reset the robot at the beginning of every episode while training the policy network to be the same as the ranges used for sampling the robot position while generating the dataset to train the FCRL encoder. These joint ranges are selected so as to ensure that there are more scenes in which the robot finger is visible. However, in order to not constrain the agent's exploration, we let the action space for training the reaching agent to be less constrained, and be able to explore the entire range from $-pi$ to pi. So, effectively, the space seen by the robot during the training of the representations is a subspace of that seen while inferring the representations from the environment used for this downstream reaching task. Interestingly, the inferred representations can also work effectively on unseen robot configurations as demonstrated by the success of the reached arm.

Similar to the encoder training for MPI3D scenes, we learned the encoder for RLScenes.

However, since the downstream tasks is a reinforcement learning task, it was hard to judge the quality of representations. Therefore, we took some insights from the MPI3D experiments and selected the model, trained with hyperparameters, which performed the best on the RL downstream tasks. Further details on the hyperparameters is provided in Table D.6.

Parameter	**Values**
Batch size	32
Representation dimension	256
Temperature: τ	0.46
Number of subsets: J	4
Max number of context points: 20	
Epochs	100
Critic	Nonlinear
Objective	NCE
Optimizer	Adam
Adam: beta1	0.9
Adam: beta2	0.999
Adam: epsilon	1e-8
Adam: learning rate	0.0005
Learning Rate Scheduler	Cosine
Number of workers	10
Batch normalization	Yes

Table D.6: Hyperparameters to train FCRL based encoder on the RLScenes Dataset.

Bibliography

Agarwal, A., Kumar, A., Malik, J., and Pathak, D. (2023). Legged locomotion in challenging terrains using egocentric vision. In *Conference on Robot Learning*, pages 403–415. PMLR.

Allen, K. R., Shelhamer, E., Shin, H., and Tenenbaum, J. B. (2019). Infinite mixture prototypes for few-shot learning. *arXiv preprint arXiv:1902.04552*.

An, B., Lyu, J., Wang, Z., Li, C., Hu, C., Tan, F., Zhang, R., Hu, Y., and Chen, C. (2020). Repulsive attention: Rethinking multi-head attention as bayesian inference. *arXiv preprint arXiv:2009.09364*.

Anand, A., Racah, E., Ozair, S., Bengio, Y., Côté, M.-A., and Hjelm, R. D. (2019). Unsupervised state representation learning in atari. In *Advances in Neural Information Processing Systems*, pages 8766–8779.

Andreas, J., Rohrbach, M., Darrell, T., and Klein, D. (2016). Neural module networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 39–48.

Andrychowicz, M., Baker, B., Chociej, M., Jozefowicz, R., McGrew, B., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., *et al.* (2018). Learning dexterous in-hand manipulation. *arXiv preprint arXiv:1808.00177*.

Anokhin, I., Demochkin, K., Khakhulin, T., Sterkin, G., Lempitsky, V., and Korzhenkov, D. (2020). Image generators with conditionally-independent pixel synthesis.

Arora, S., Khandeparkar, H., Khodak, M., Plevrakis, O., and Saunshi, N. (2019). A theoretical analysis of contrastive unsupervised representation learning. *arXiv preprint arXiv:1902.09229*.

Bachman, P., Hjelm, R. D., and Buchwalter, W. (2019). Learning representations by maximizing mutual information across views. In *Advances in Neural Information Processing Systems*, pages 15509–15519.

Bahdanau, D., Murty, S., Noukhovitch, M., Nguyen, T. H., de Vries, H., and Courville, A. (2019). Systematic generalization: What is required and can it be learned? In *International Conference on Learning Representations*.

Barbu, A., Mayo, D., Alverio, J., Luo, W., Wang, C., Gutfreund, D., Tenenbaum, J., and Katz, B. (2019). Objectnet: A large-scale bias-controlled dataset for pushing the limits of object recognition models. *Advances in neural information processing systems*, **32**.

Barrett, D. G. T., Hill, F., Santoro, A., Morcos, A. S., and Lillicrap, T. (2018). Measuring abstract reasoning in neural networks.

Bayer, J. and Osendorfer, C. (2014). Learning stochastic recurrent networks. *arXiv preprint arXiv:1411.7610*.

Bengio, Y., LeCun, Y., *et al.* (2007). Scaling learning algorithms towards ai. *Large-scale kernel machines*, **34**(5), 1–41.

Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, **35**(8), 1798–1828.

Bengio, Y., Deleu, T., Rahaman, N., Ke, R., Lachapelle, S., Bilaniuk, O., Goyal, A., and Pal, C. (2019). A meta-transfer objective for learning to disentangle causal mechanisms.

Besserve, M., Sun, R., Janzing, D., and Schölkopf, B. (2020). A theory of independent mechanisms for extrapolation in generative models. *arXiv preprint arXiv:2004.00184*.

Bishop, G., Welch, G., *et al.* (2001). An introduction to the kalman filter.

Bommasani, R., Hudson, D. A., Adeli, E., Altman, R., Arora, S., von Arx, S., Bernstein, M. S., Bohg, J., Bosselut, A., Brunskill, E., *et al.* (2021). On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*.

Bouchacourt, D., Tomioka, R., and Nowozin, S. (2018). Multi-level variational autoencoder: Learning disentangled representations from grouped observations. In *AAAI*.

Bousmalis, K., Trigeorgis, G., Silberman, N., Krishnan, D., and Erhan, D. (2016). Domain separation networks. In *Advances in Neural Information Processing Systems*, pages 343–351.

Bousquet, O. and Elisseeff, A. (2002). Stability and generalization. *JMLR*, **2**, 499–526.

Bowman, S. R., Vilnis, L., Vinyals, O., Dai, A. M., Jozefowicz, R., and Bengio, S. (2015). Generating sentences from a continuous space. *arXiv preprint arXiv:1511.06349*.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., *et al.* (2020). Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

Buch, S., Fei-Fei, L., and Goodman, N. D. (2021). Neural event semantics for grounded language understanding. In *Transactions of the Association for Computational Linguistics (TACL)*.

Chang, M. B., Ullman, T., Torralba, A., and Tenenbaum, J. B. (2016). A compositional object-based approach to learning physical dynamics. *arXiv preprint arXiv:1612.00341*.

Chang, M. B., Gupta, A., Levine, S., and Griffiths, T. L. (2018). Automatically composing representation transformations as a means for generalization. *arXiv preprint arXiv:1807.04640*.

Chartsias, A., Joyce, T., Papanastasiou, G., Semple, S., Williams, M., Newby, D., Dharmakumar, R., and Tsaftaris, S. A. (2018). Factorised spatial representation learning: application in semi-supervised myocardial segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 490–498. Springer.

Chen, T., Kornblith, S., Norouzi, M., and Hinton, G. (2020). A simple framework for contrastive learning of visual representations. *arXiv preprint arXiv:2002.05709*.

Chen, T. Q., Li, X., Grosse, R. B., and Duvenaud, D. K. (2018). Isolating sources of disentanglement in variational autoencoders. In *Advances in Neural Information Processing Systems*, pages 2610–2620.

Chen, X., Duan, Y., Houthooft, R., Schulman, J., Sutskever, I., and Abbeel, P. (2016). Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in neural information processing systems*, pages 2172–2180.

Chen, X., Xie, S., and He, K. (2021). An empirical study of training self-supervised vision transformers.

Cheung, B., Livezey, J. A., Bansal, A. K., and Olshausen, B. A. (2015). Discovering hidden factors of variation in deep networks. In *Workshop at International Conference on Learning Representations*.

Choromanska, A., Henaff, M., Mathieu, M., Arous, G. B., and LeCun, Y. (2015). The loss surfaces of multilayer networks. In *Artificial intelligence and statistics*, pages 192–204. PMLR.

Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., *et al.* (2022). Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*.

Chung, J., Kastner, K., Dinh, L., Goel, K., Courville, A. C., and Bengio, Y. (2015). A recurrent latent variable model for sequential data. In *Advances in neural information processing systems*, pages 2980–2988.

Clanuwat, T., Bober-Irizar, M., Kitamoto, A., Lamb, A., Yamamoto, K., and Ha, D. (2018). Deep learning for classical japanese literature.

Cordonnier, J.-B., Loukas, A., and Jaggi, M. (2019). On the relationship between self-attention and convolutional layers. *arXiv preprint arXiv:1911.03584*.

Creager, E., Madras, D., Jacobson, J.-H., Weis, M., Swersky, K., Pitassi, T., and Zemel, R. (2019). Flexibly fair representation learning by disentanglement. In *ICML*, page to appear.

Cubuk, E. D., Zoph, B., Shlens, J., and Le, Q. V. (2019). Randaugment: Practical automated data augmentation with a reduced search space.

Dehghani, M., Gouws, S., Vinyals, O., Uszkoreit, J., and Łukasz Kaiser (2019). Universal transformers.

Denton, E. L. *et al.* (2017). Unsupervised learning of disentangled representations from video. In *Advances in Neural Information Processing Systems*, pages 4414–4423.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding.

Doersch, C., Gupta, A., and Efros, A. A. (2015). Unsupervised visual representation learning by context prediction. In *Proceedings of the IEEE international conference on computer vision*, pages 1422–1430.

Doersch, C., Gupta, A., and Zisserman, A. (2020). Crosstransformers: spatially-aware few-shot transfer. *arXiv preprint arXiv:2007.11498*.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., *et al.* (2020). An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*.

Durbin, J. and Koopman, S. J. (2012). *Time series analysis by state space methods*, volume 38. Oxford University Press.

Eastwood, C. and Williams, C. K. I. (2018). A framework for the quantitative evaluation of disentangled representations. In *International Conference on Learning Representations*.

Eslami, S. A., Rezende, D. J., Besse, F., Viola, F., Morcos, A. S., Garnelo, M., Ruderman, A., Rusu, A. A., Danihelka, I., Gregor, K., *et al.* (2018). Neural scene representation and rendering. *Science*, **360**(6394), 1204–1210.

Esmaeili, B., Wu, H., Jain, S., Narayanaswamy, S., Paige, B., and Van de Meent, J.-W. (2018). Hierarchical disentangled representations. *arXiv preprint arXiv:1804.02086*.

Fedus, W., Zoph, B., and Shazeer, N. (2021). Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *arXiv preprint arXiv:2101.03961*.

Fernando, C., Banarse, D., Blundell, C., Zwols, Y., Ha, D., Rusu, A. A., Pritzel, A., and Wierstra, D. (2017). Pathnet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734*.

Fidler, S., Dickinson, S., and Urtasun, R. (2012). 3d object detection and viewpoint estimation with a deformable 3d cuboid model. In *Advances in neural information processing systems*, pages 611–619.

Finn, C. and Levine, S. (2017). Deep visual foresight for planning robot motion. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 2786–2793. IEEE.

Finn, C., Abbeel, P., and Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1126–1135. JMLR. org.

Fraccaro, M., Sønderby, S. K., Paquet, U., and Winther, O. (2016). Sequential neural models with stochastic layers. In *Advances in neural information processing systems*, pages 2199–2207.

Fraccaro, M., Kamronn, S., Paquet, U., and Winther, O. (2017). A disentangled recognition and nonlinear dynamics model for unsupervised learning. In *Advances in Neural Information Processing Systems*, pages 3601–3610.

Ganin, Y., Ustinova, E., Ajakan, H., Germain, P., Larochelle, H., Laviolette, F., Marchand, M., and Lempitsky, V. (2016). Domain-adversarial training of neural networks. *The Journal of Machine Learning Research*, **17**(1), 2096–2030.

Garnelo, M., Rosenbaum, D., Maddison, C. J., Ramalho, T., Saxton, D., Shanahan, M., Teh, Y. W., Rezende, D. J., and Eslami, S. (2018a). Conditional neural processes. *arXiv preprint arXiv:1807.01613*.

Garnelo, M., Schwarz, J., Rosenbaum, D., Viola, F., Rezende, D. J., Eslami, S., and Teh, Y. W. (2018b). Neural processes. *arXiv preprint arXiv:1807.01622*.

Gawehn, E., Hiss, J. A., and Schneider, G. (2016). Deep learning in drug discovery. *Molecular informatics*, **35**(1), 3–14.

Geirhos, R., Jacobsen, J.-H., Michaelis, C., Zemel, R., Brendel, W., Bethge, M., and Wichmann, F. A. (2020). Shortcut learning in deep neural networks. *Nature Machine Intelligence*, **2**(11), 665–673.

Gelb, A. (1974). *Applied optimal estimation*.

Gick, M. L. and Holyoak, K. J. (1987). The cognitive basis of knowledge transfer. In *Transfer of learning*, pages 9–46. Elsevier.

Gondal, M. W., Wuthrich, M., Miladinovic, D., Locatello, F., Breidt, M., Volchkov, V., Akpo, J., Bachem, O., Schölkopf, B., and Bauer, S. (2019a). On the transfer of inductive bias from simulation to the real world: a new disentanglement dataset. In *Advances in Neural Information Processing Systems*, pages 15740–15751.

Gondal, M. W., Schölkopf, B., and Hirsch, M. (2019b). The unreasonable effectiveness of texture transfer for single image super-resolution. In *Computer Vision–ECCV 2018 Workshops: Munich, Germany, September 8-14, 2018, Proceedings, Part V 15*, pages 80–97. Springer.

Gondal, M. W., Joshi, S., Rahaman, N., Bauer, S., Wuthrich, M., and Schölkopf, B. (2021). Function contrastive learning of transferable meta-representations. In *International Conference on Machine Learning*, pages 3755–3765. PMLR.

Goodfellow, I., Lee, H., Le, Q. V., Saxe, A., and Ng, A. Y. (2009). Measuring invariances in deep networks. In *Advances in neural information processing systems*, pages 646–654.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT press.

Gordon, J., Bruinsma, W., Foong, A. Y., Requeima, J., Dubois, Y., and Turner, R. E. (2020). Convolutional conditional neural processes.

Goyal, A. and Bengio, Y. (2020). Inductive biases for deep learning of higher-level cognition. *arXiv preprint arXiv:2011.15091*.

Goyal, A. and Bengio, Y. (2022). Inductive biases for deep learning of higher-level cognition. *Proceedings of the Royal Society A*, **478**(2266), 20210068.

Goyal, A., Lamb, A., Hoffmann, J., Sodhani, S., Levine, S., Bengio, Y., and Schölkopf, B. (2019). Recurrent independent mechanisms. *arXiv preprint arXiv:1909.10893*.

Goyal, A., Lamb, A., Gampa, P., Beaudoin, P., Levine, S., Blundell, C., Bengio, Y., and Mozer, M. (2020). Object files and schemata: Factorizing declarative and procedural knowledge in dynamical systems. *arXiv preprint arXiv:2006.16225*.

Goyal, A., Didolkar, A., Ke, N. R., Blundell, C., Beaudoin, P., Heess, N., Mozer, M., and Bengio, Y. (2021). Neural production systems. *arXiv preprint arXiv:2103.01937*.

Goyal, A. G. A. P., Sordoni, A., Côté, M.-A., Ke, N. R., and Bengio, Y. (2017). Z-forcing: Training stochastic recurrent networks. In *Advances in neural information processing systems*, pages 6713–6723.

Graves, A. (2013). Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*.

Graves, A., Mohamed, A.-r., and Hinton, G. (2013). Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. Ieee.

Gretton, A., Borgwardt, K. M., Rasch, M. J., Schölkopf, B., and Smola, A. (2012). A kernel two-sample test. *Journal of Machine Learning Research*, **13**(Mar), 723–773.

Gupta, V., Koren, T., and Singer, Y. (2018). Shampoo: Preconditioned stochastic tensor optimization.

Gutmann, M. and Hyvärinen, A. (2010). Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 297–304.

Ha, D. and Schmidhuber, J. (2018). World models. *arXiv preprint arXiv:1803.10122*.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*.

Hafner, D., Lillicrap, T., Fischer, I., Villegas, R., Ha, D., Lee, H., and Davidson, J. (2018). Learning latent dynamics for planning from pixels. *arXiv preprint arXiv:1811.04551*.

He, K., Fan, H., Wu, Y., Xie, S., and Girshick, R. (2019). Momentum contrast for unsupervised visual representation learning. *arXiv preprint arXiv:1911.05722*.

Hendrycks, D. and Dietterich, T. (2019). Benchmarking neural network robustness to common corruptions and perturbations. *arXiv preprint arXiv:1903.12261*.

Hendrycks, D. and Gimpel, K. (2020). Gaussian error linear units (gelus).

Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., Mohamed, S., and Lerchner, A. (2016). beta-vae: Learning basic visual concepts with a constrained variational framework.

Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., Mohamed, S., and Lerchner, A. (2017a). beta-vae: Learning basic visual concepts with a constrained variational framework. In *International Conference on Learning Representations*.

Higgins, I., Pal, A., Rusu, A., Matthey, L., Burgess, C., Pritzel, A., Botvinick, M., Blundell, C., and Lerchner, A. (2017b). Darla: Improving zero-shot transfer in reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1480–1490. JMLR. org.

Higgins, I., Sonnerat, N., Matthey, L., Pal, A., Burgess, C. P., Bosnjak, M., Shanahan, M., Botvinick, M., Hassabis, D., and Lerchner, A. (2017c). Scan: Learning hierarchical compositional visual concepts. *arXiv preprint arXiv:1707.03389*.

Higgins, I., Sonnerat, N., Matthey, L., Pal, A., Burgess, C. P., Bošnjak, M., Shanahan, M., Botvinick, M., Hassabis, D., and Lerchner, A. (2018a). Scan: Learning hierarchical compositional visual concepts. In *International Conference on Learning Representations*.

Higgins, I., Amos, D., Pfau, D., Racaniere, S., Matthey, L., Rezende, D., and Lerchner, A. (2018b). Towards a definition of disentangled representations. *arXiv preprint arXiv:1812.02230*.

Hjelm, R. D., Fedorov, A., Lavoie-Marchildon, S., Grewal, K., Bachman, P., Trischler, A., and Bengio, Y. (2018). Learning deep representations by mutual information estimation and maximization. *arXiv preprint arXiv:1808.06670*.

Hsieh, J.-T., Liu, B., Huang, D.-A., Fei-Fei, L., and Niebles, J. C. (2018). Learning to decompose and disentangle representations for video prediction. *arXiv preprint arXiv:1806.04166*.

Hsu, W.-N., Zhang, Y., and Glass, J. (2017). Unsupervised learning of disentangled and interpretable representations from sequential data. In *Advances in neural information processing systems*, pages 1878–1889.

Hudson, D. A. and Manning, C. D. (2018). Compositional attention networks for machine reasoning. *arXiv preprint arXiv:1803.03067*.

Itseez (2015). Open source computer vision library. `https://github.com/itseez/opencv`.

James, S., Wohlhart, P., Kalakrishnan, M., Kalashnikov, D., Irpan, A., Ibarz, J., Levine, S., Hadsell, R., and Bousmalis, K. (2018). Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks. *arXiv preprint arXiv:1812.07252*.

Jitkrittum, W., Sangkloy, P., Gondal, M. W., Raj, A., Hays, J., and Schölkopf, B. (2019). Kernel mean matching for content addressability of gans. PMLR.

Jo, J. and Bengio, Y. (2017). Measuring the tendency of cnns to learn surface statistical regularities.

Joshi, S., Widmaier, F., Agrawal, V., and Wüthrich, M. (2020). `https://github.com/open-dynamic-robot-initiative/trifinger_simulation`.

Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A., *et al.* (2021). Highly accurate protein structure prediction with alphafold. *Nature*, **596**(7873), 583–589.

Kansky, K., Silver, T., Mély, D. A., Eldawy, M., Lázaro-Gredilla, M., Lou, X., Dorfman, N., Sidor, S., Phoenix, S., and George, D. (2017). Schema networks: Zero-shot transfer with a generative causal model of intuitive physics. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1809–1818. JMLR. org.

Karl, M., Soelch, M., Bayer, J., and van der Smagt, P. (2016). Deep variational bayes filters: Unsupervised learning of state space models from raw data. *arXiv preprint arXiv:1605.06432*.

Kendal, S. L. and Creen, M. (2007). *An introduction to knowledge engineering*. Springer.

Keysers, D., Schärli, N., Scales, N., Buisman, H., Furrer, D., Kashubin, S., Momchev, N., Sinopalnikov, D., Stafiniak, L., Tihon, T., *et al.* (2019). Measuring compositional generalization: A comprehensive method on realistic data. *arXiv preprint arXiv:1912.09713*.

Kim, H. and Mnih, A. (2018). Disentangling by factorising. *arXiv preprint arXiv:1802.05983*.

Kim, H., Mnih, A., Schwarz, J., Garnelo, M., Eslami, A., Rosenbaum, D., Vinyals, O., and Teh, Y. W. (2019). Attentive neural processes. *arXiv preprint arXiv:1901.05761*.

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Kipf, T., van der Pol, E., and Welling, M. (2019). Contrastive learning of structured world models. *arXiv preprint arXiv:1911.12247*.

Kirsch, L., Kunze, J., and Barber, D. (2018). Modular networks: Learning to decompose neural computation. *arXiv preprint arXiv:1811.05249*.

Koedinger, K. R. and Roll, I. (2012). Learning to think: Cognitive mechanisms of knowledge transfer. *The Oxford handbook of thinking and reasoning*, pages 789–806.

Koller, D., Friedman, N., and Bach, F. (2009). *Probabilistic graphical models: principles and techniques*.

Krishnan, R. G., Shalit, U., and Sontag, D. (2015). Deep kalman filters. *arXiv preprint arXiv:1511.05121*.

Krishnan, R. G., Shalit, U., and Sontag, D. (2017). Structured inference networks for nonlinear state space models.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, **25**, 1097–1105.

Kulkarni, T. D., Whitney, W. F., Kohli, P., and Tenenbaum, J. (2015). Deep convolutional inverse graphics network. In *Advances in neural information processing systems*, pages 2539–2547.

Kumar, A., Sattigeri, P., and Balakrishnan, A. (2017). Variational inference of disentangled latent concepts from unlabeled observations. In *International Conference on Learning Representations*.

Lake, B. and Baroni, M. (2018). Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *International Conference on Machine Learning*, pages 2873–2882. PMLR.

Lake, B. M., Salakhutdinov, R., and Tenenbaum, J. B. (2015). Human-level concept learning through probabilistic program induction. *Science*, **350**(6266), 1332–1338.

Lake, B. M., Ullman, T. D., Tenenbaum, J. B., and Gershman, S. J. (2017). Building machines that learn and think like people. *Behavioral and Brain Sciences*, **40**.

Lamb, A., He, D., Goyal, A., Ke, G., Liao, C.-F., Ravanelli, M., and Bengio, Y. (2021). Transformers with competitive ensembles of independent mechanisms. *arXiv preprint arXiv:2103.00336*.

Laversanne-Finot, A., Pere, A., and Oudeyer, P.-Y. (2018). Curiosity driven exploration of learned disentangled goal spaces. In *Conference on Robot Learning*, pages 487–504.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, **86**(11), 2278–2324.

LeCun, Y., Huang, F. J., Bottou, L., *et al.* (2004). Learning methods for generic object recognition with invariance to pose and lighting. In *CVPR (2)*, pages 97–104. Citeseer.

LeCun, Y., Cortes, C., and Burges, C. (2010). Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, **2**.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *nature*, **521**(7553), 436.

Lee, K., Maji, S., Ravichandran, A., and Soatto, S. (2019). Meta-learning with differentiable convex optimization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 10657–10665.

Lenc, K. and Vedaldi, A. (2015). Understanding image representations by measuring their equivariance and equivalence. In *IEEE conference on computer vision and pattern recognition*, pages 991–999.

Lesort, T., Díaz-Rodríguez, N., Goudou, J.-F., and Filliat, D. (2018). State representation learning for control: An overview. *Neural Networks*.

Li, Z., Zhou, F., Chen, F., and Li, H. (2017). Meta-sgd: Learning to learn quickly for few-shot learning. *arXiv preprint arXiv:1707.09835*.

Liu, L., Jiang, H., He, P., Chen, W., Liu, X., Gao, J., and Han, J. (2019). On the variance of the adaptive learning rate and beyond. *arXiv preprint arXiv:1908.03265*.

Liu, Y.-C., Yeh, Y.-Y., Fu, T.-C., Chiu, W.-C., Wang, S.-D., and Wang, Y.-C. F. (2017). Detach and adapt: Learning cross-domain disentangled deep representation. *arXiv preprint arXiv:1705.01314*.

Liu, Z., Luo, P., Wang, X., and Tang, X. (2015). Deep learning face attributes in the wild. In *Proceedings of the IEEE international conference on computer vision*, pages 3730–3738.

Locatello, F., Bauer, S., Lucic, M., Gelly, S., Schölkopf, B., and Bachem, O. (2018). Challenging common assumptions in the unsupervised learning of disentangled representations. *arXiv preprint arXiv:1811.12359*.

Locatello, F., Tschannen, M., Bauer, S., Rätsch, G., Schölkopf, B., and Bachem, O. (2019a). Disentangling factors of variation using few labels. *arXiv preprint arXiv:1905.01258*.

Locatello, F., Abbati, G., Rainforth, T., Bauer, S., Scölkopf, B., and Bachem, O. (2019b). On the fairness of disentangled representations. *arXiv preprint arXiv:1905.13662*.

Locatello, F., Weissenborn, D., Unterthiner, T., Mahendran, A., Heigold, G., Uszkoreit, J., Dosovitskiy, A., and Kipf, T. (2020). Object-centric learning with slot attention.

Logeswaran, L., Lee, A., Ott, M., Lee, H., Ranzato, M., and Szlam, A. (2020). Few-shot sequence learning with transformers. *arXiv preprint arXiv:2012.09543*.

Loula, J., Baroni, M., and Lake, B. M. (2018). Rearranging the familiar: Testing compositional generalization in recurrent networks. *arXiv preprint arXiv:1807.07545*.

Lu, C., Schölkopf, B., and Hernández-Lobato, J. M. (2018). Deconfounding reinforcement learning in observational settings. *arXiv preprint arXiv:1812.10576*.

Mao, J., Gan, C., Kohli, P., Tenenbaum, J. B., and Wu, J. (2019). The neuro-symbolic concept learner: Interpreting scenes, words, and sentences from natural supervision. *arXiv preprint arXiv:1904.12584*.

Marino, J., Cvitkovic, M., and Yue, Y. (2018). A general method for amortizing variational filtering. In *Advances in Neural Information Processing Systems*, pages 7867–7878.

Mathieu, M. F., Zhao, J. J., Zhao, J., Ramesh, A., Sprechmann, P., and LeCun, Y. (2016). Disentangling factors of variation in deep representation using adversarial training. In *Advances in Neural Information Processing Systems*, pages 5040–5048.

Miladinović, Đ., Gondal, W., Schölkopf, B., Buhmann, J. M., and Bauer, S. (2019a). Disentangled state space models: Unsupervised learning of dynamics across heterogeneous environments.

Miladinović, Đ., Gondal, M. W., Schölkopf, B., Buhmann, J. M., and Bauer, S. (2019b). Disentangled state space representations. *arXiv preprint arXiv:1906.03255*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., *et al.* (2015). Human-level control through deep reinforcement learning. *nature*, **518**(7540), 529–533.

Munch, E. (1893). The scream.

Nair, A. V., Pong, V., Dalal, M., Bahl, S., Lin, S., and Levine, S. (2018). Visual reinforcement learning with imagined goals. In *Advances in Neural Information Processing Systems*, pages 9209–9220.

Narayanaswamy, S., Paige, T. B., Van de Meent, J.-W., Desmaison, A., Goodman, N., Kohli, P., Wood, F., and Torr, P. (2017). Learning disentangled representations with semi-supervised deep generative models. In *Advances in Neural Information Processing Systems*, pages 5925–5935.

Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., and Ng, A. Y. (2011). Reading digits in natural images with unsupervised feature learning.

Nokes, T. J. (2009). Mechanisms of knowledge transfer. *Thinking & reasoning*, **15**(1), 1–36.

Øksendal, B. (2003). Stochastic differential equations. In *Stochastic differential equations*, pages 65–84. Springer.

Oord, A. v. d., Li, Y., and Vinyals, O. (2018). Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*.

Parascandolo, G., Kilbertus, N., Rojas-Carulla, M., and Schölkopf, B. (2017). Learning independent causal mechanisms. *arXiv preprint arXiv:1712.00961*.

Parascandolo, G., Kilbertus, N., Rojas-Carulla, M., and Schölkopf, B. (2018). Learning independent causal mechanisms. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, volume 80 of *Proceedings of Machine Learning Research*, pages 4033–4041. PMLR.

Pearl, J. (2009). *Causality: models, reasoning, and inference*. Cambridge university press.

Peters, J., Bühlmann, P., and Meinshausen, N. (2016). Causal inference by using invariant prediction: identification and confidence intervals. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **78**(5), 947–1012.

Peters, J., Janzing, D., and Schölkopf, B. (2017). *Elements of causal inference: foundations and learning algorithms*. MIT press.

Qiao, L., Shi, Y., Li, J., Wang, Y., Huang, T., and Tian, Y. (2019). Transductive episodic-wise adaptive metric for few-shot learning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3603–3612.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2019). Language models are unsupervised multitask learners. *OpenAI blog*, **1**(8), 9.

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., *et al.* (2021). Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pages 8748–8763. PMLR.

Rahaman, N., Baratin, A., Arpit, D., Draxler, F., Lin, M., Hamprecht, F. A., Bengio, Y., and Courville, A. (2019). On the spectral bias of neural networks.

Rahaman, N., Goyal, A., Gondal, M. W., Wuthrich, M., Bauer, S., Sharma, Y., Bengio, Y., and Schölkopf, B. (2020). S2rms: Spatially structured recurrent modules.

Rahaman, N., Gondal, M. W., Joshi, S., Gehler, P., Bengio, Y., Locatello, F., and Schölkopf, B. (2021). Dynamic inference with neural interpreters. *Advances in Neural Information Processing Systems*, **34**, 10985–10998.

Raven, J. C. and Court, J. H. (1998). *Raven's progressive matrices and vocabulary scales*, volume 759. Oxford pyschologists Press Oxford, England.

Ravi, S. and Larochelle, H. (2016). Optimization as a model for few-shot learning.

Reed, S. E., Zhang, Y., Zhang, Y., and Lee, H. (2015). Deep visual analogy-making. In *Advances in neural information processing systems*, pages 1252–1260.

Rezende, D. J., Mohamed, S., and Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*.

Ridgeway, K. and Mozer, M. C. (2018). Learning deep disentangled embeddings with the f-statistic loss. In *Advances in Neural Information Processing Systems*, pages 185–194.

Rosenbaum, C., Klinger, T., and Riemer, M. (2017). Routing networks: Adaptive selection of non-linear functions for multi-task learning. *arXiv preprint arXiv:1711.01239*.

Rosenbaum, C., Cases, I., Riemer, M., and Klinger, T. (2019). Routing networks and the challenges of modular and compositional computation. *arXiv preprint arXiv:1904.12774*.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, **65**(6), 386.

Sabour, S., Frosst, N., and Hinton, G. E. (2017). Dynamic routing between capsules. *arXiv preprint arXiv:1710.09829*.

Schmidhuber, J. (1992). Learning factorial codes by predictability minimization. *Neural Computation*, **4**(6), 863–879.

Schölkopf, B., Janzing, D., Peters, J., Sgouritsa, E., Zhang, K., and Mooij, J. (2012). On causal and anticausal learning. In *International Conference on Machine Learning*, pages 1255–1262.

Schölkopf, B., Locatello, F., Bauer, S., Nan Ke, R., Kalchbrenner, N., Goyal, A., and Bengio, Y. (2021). Towards causal representation learning. *Proceedings of the IEEE*.

Schölkopf, B., Locatello, F., Bauer, S., Ke, N. R., Kalchbrenner, N., Goyal, A., and Bengio, Y. (2021). Toward causal representation learning. *Proceedings of the IEEE*, **109**(5), 612–634.

Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., and Dean, J. (2017). Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., *et al.* (2016). Mastering the game of go with deep neural networks and tree search. *nature*, **529**(7587), 484–489.

Snell, J., Swersky, K., and Zemel, R. (2017). Prototypical networks for few-shot learning. In *Advances in neural information processing systems*, pages 4077–4087.

Sohn, K. (2016). Improved deep metric learning with multi-class n-pair loss objective. In *Advances in neural information processing systems*, pages 1857–1865.

Spirtes, P., Glymour, C., and Scheines, R. (1993). *Causation, prediction, and search*. Springer-Verlag. (2nd edition MIT Press 2000).

Srinivas, A., Laskin, M., and Abbeel, P. (2020). Curl: Contrastive unsupervised representations for reinforcement learning. *arXiv preprint arXiv:2004.04136*.

Srivastava, N., Mansimov, E., and Salakhudinov, R. (2015). Unsupervised learning of video representations using lstms. In *International conference on machine learning*, pages 843–852.

Steenbrugge, X., Leroux, S., Verbelen, T., and Dhoedt, B. (2018). Improving generalization for abstract reasoning tasks using disentangled feature representations. In *Workshop on Relational Representation Learning at Neural Information Processing Systems*.

Sünderhauf, N., Brock, O., Scheirer, W., Hadsell, R., Fox, D., Leitner, J., Upcroft, B., Abbeel, P., Burgard, W., Milford, M., *et al.* (2018). The limits and potentials of deep learning for robotics. *The International journal of robotics research*, **37**(4-5), 405–420.

Sung, F., Yang, Y., Zhang, L., Xiang, T., Torr, P. H., and Hospedales, T. M. (2018). Learning to compare: Relation network for few-shot learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1199–1208.

Suter, R., Miladinovic, D., Schölkopf, B., and Bauer, S. (2019). Robustly disentangled causal mechanisms: Validating deep representations for interventional robustness. In *International Conference on Machine Learning*, pages 6056–6065. PMLR.

Tian, Y., Krishnan, D., and Isola, P. (2019). Contrastive multiview coding. *arXiv preprint arXiv:1906.05849*.

Tian, Y., Sun, C., Poole, B., Krishnan, D., Schmid, C., and Isola, P. (2020). What makes for good views for contrastive learning. *arXiv preprint arXiv:2005.10243*.

Tomczak, J. M. and Welling, M. (2017). Vae with a vampprior. *arXiv preprint arXiv:1705.07120*.

Ton, J.-F., Chan, L., Teh, Y. W., and Sejdinovic, D. (2019). Noise contrastive meta-learning for conditional density estimation using kernel mean embeddings. *arXiv preprint arXiv:1906.02236*.

Touvron, H., Cord, M., Sablayrolles, A., Synnaeve, G., and Jégou, H. (2021a). Going deeper with image transformers.

Touvron, H., Cord, M., Douze, M., Massa, F., Sablayrolles, A., and Jégou, H. (2021b). Training data-efficient image transformers and distillation through attention.

Tschannen, M., Djolonga, J., Rubenstein, P. K., Gelly, S., and Lucic, M. (2019). On mutual information maximization for representation learning. *arXiv preprint arXiv:1907.13625*.

Tulyakov, S., Liu, M.-Y., Yang, X., and Kautz, J. (2017). Mocogan: Decomposing motion and content for video generation. *arXiv preprint arXiv:1707.04993*.

Van Steenkiste, S., Chang, M., Greff, K., and Schmidhuber, J. (2018). Relational neural expectation maximization: Unsupervised discovery of objects and their interactions. *arXiv preprint arXiv:1802.10353*.

van Steenkiste, S., Locatello, F., Schmidhuber, J., and Bachem, O. (2019). Are disentangled representations helpful for abstract visual reasoning? *arXiv preprint arXiv:1905.12506*.

Vapnik, V. (1995). *The Nature of Statistical Learning Theory*. Springer, NY.

Vapnik, V. N. (1999). An overview of statistical learning theory. *IEEE transactions on neural networks*, **10**(5), 988–999.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. *arXiv preprint arXiv:1706.03762*.

Villegas, R., Yang, J., Hong, S., Lin, X., and Lee, H. (2017). Decomposing motion and content for natural video sequence prediction. *arXiv preprint arXiv:1706.08033*.

Vinyals, O., Blundell, C., Lillicrap, T., Wierstra, D., *et al.* (2016). Matching networks for one shot learning. In *Advances in neural information processing systems*, pages 3630–3638.

Wang, D., Jamnik, M., and Lio, P. (2020). Abstract diagrammatic reasoning with multiplex graph networks. *arXiv preprint arXiv:2006.11197*.

Wang, T. and Isola, P. (2020). Understanding contrastive representation learning through alignment and uniformity on the hypersphere. *arXiv preprint arXiv:2005.10242*.

Watter, M., Springenberg, J., Boedecker, J., and Riedmiller, M. (2015). Embed to control: A locally linear latent dynamics model for control from raw images. In *Advances in neural information processing systems*, pages 2746–2754.

Watters, N., Matthey, L., Burgess, C. P., and Lerchner, A. (2019). Spatial broadcast decoder: A simple architecture for learning disentangled representations in vaes. *arXiv preprint arXiv:1901.07017*.

Weiss, M., Rahaman, N., Locatello, F., Pal, C., Bengio, Y., Schölkopf, B., Li, E. L., and Ballas, N. (2022). Neural attentive circuits. *Advances in Neural Information Processing Systems*, **35**, 7741–7754.

Wightman, R. (2019). Pytorch image models. `https://github.com/rwightman/pytorch-image-models`.

Wolpert, D. H. and Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, **1**(1), 67–82.

Wu, Y., Dong, H., Grosse, R., and Ba, J. (2020). The scattering compositional learner: Discovering objects, attributes, relationships in analogical reasoning. *arXiv preprint arXiv:2007.04212*.

Xu, J., Ton, J.-F., Kim, H., Kosiorek, A. R., and Teh, Y. W. (2019). Metafun: Meta-learning with iterative functional updates. *arXiv preprint arXiv:1912.02738*.

Yi, K., Gan, C., Li, Y., Kohli, P., Wu, J., Torralba, A., and Tenenbaum, J. B. (2019). Clevrer: Collision events for video representation and reasoning. *arXiv preprint arXiv:1910.01442*.

Yingzhen, L. and Mandt, S. (2018). Disentangled sequential autoencoder. In *International Conference on Machine Learning*, pages 5656–5665.

Young, T., Hazarika, D., Poria, S., and Cambria, E. (2018). Recent trends in deep learning based natural language processing. *ieee Computational intelligenCe magazine*, **13**(3), 55–75.

Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R. R., and Smola, A. J. (2017). Deep sets. In *Advances in neural information processing systems*, pages 3391–3401.

Zhang, A., McAllister, R., Calandra, R., Gal, Y., and Levine, S. (2020). Learning invariant representations for reinforcement learning without reconstruction. *arXiv preprint arXiv:2006.10742*.

Zhang, J., Tai, L., Yun, P., Xiong, Y., Liu, M., Boedecker, J., and Burgard, W. (2019). Vr-goggles for robots: Real-to-sim domain adaptation for visual control. *IEEE Robotics and Automation Letters*, **4**(2), 1148–1155.

Zhang, R., Isola, P., and Efros, A. A. (2017). Split-brain autoencoders: Unsupervised learning by cross-channel prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1058–1067.

Zilberstein, S. (1996). Using anytime algorithms in intelligent systems. *AI magazine*, **17**(3), 73–73.