Université de Montréal

On Sample Efficiency and Systematic Generalization of Grounded Language Understanding With Deep Learning

par

Dzmitry Bahdanau

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Thèse présentée à la Faculté des études supérieures
en vue de l'obtention du grade de
Philosophiæ Doctor (Ph.D.)
en informatique

le 2 mar 2020

Université de Montréal

Faculté des études supérieures

Cette thèse intitulée

On Sample Efficiency and Systematic Generalization of Grounded Language Understanding With Deep Learning

présentée par

Dzmitry Bahdanau

a été évaluée par un jury composé des personnes suivantes :

Aaron Courville
(président-rapporteur)

Yoshua Bengio
(directeur de recherche)

Christopher Pal
(membre du jury)

Jacob Andreas
(examinateur externe)

Aaron Courville
(représentant du doyen de la FAS)

Thèse acceptée le
le 20 février 2020

SOMMAIRE

En utilisant la méthodologie de l'apprentissage profond qui préconise de s'appuyer davantage sur des données et des modèles neuronaux flexibles plutôt que sur les connaissances de l'expert dans le domaine, la communauté de recherche a récemment réalisé des progrès remarquables dans la compréhension et la génération du langue naturel. Néanmoins, il reste difficile de savoir si une simple extension des méthodes d'apprentissage profond existantes sera suffisante pour atteindre l'objectif d'utiliser le langage naturel pour l'interaction homme-machine. Nous nous concentrons sur deux aspects connexes dans lesquels les méthodes actuelles semblent nécessiter des améliorations majeures. Le premier de ces aspects est l'inefficacité statistique des systèmes d'apprentissage profond: ils sont connus pour nécessiter de grandes quantités de données pour bien fonctionner. Le deuxième aspect est leur capacité limitée à généraliser systématiquement, à savoir à comprendre le langage dans des situations où la distribution des données change mais les principes de syntaxe et de sémantique restent les mêmes.

Dans cette thèse, nous présentons quatre études de cas dans lesquelles nous cherchons à apporter plus de clarté concernant l'efficacité statistique susmentionnée et les aspects de généralisation systématique des approches d'apprentissage profond de la compréhension des langues, ainsi qu'à faciliter la poursuite des travaux sur ces sujets. Afin de séparer le problème de la représentation des connaissances du monde réel du problème de l'apprentissage d'une langue, nous menons toutes ces études en utilisant des langages synthétiques ancrés dans des environnements visuels simples.

Dans le premier article, nous étudions comment former les agents à suivre des instructions compositionnelles dans des environnements avec une forme de supervision restreinte. À savoir pour chaque instruction et configuration initiale de l'environnement, nous ne fournissons qu'un état cible au lieu d'une trajectoire complète avec des actions à toutes les étapes. Nous adaptons les méthodes d'apprentissage adversariel par imitation à ce paramètre et démontrons qu'une telle forme restreinte de données est suffisante pour apprendre les significations compositionelles des instructions. Notre deuxième article se concentre également sur des agents qui apprennent à exécuter des instructions. Nous développons la plateforme BabyAI pour faciliter des études plus approfondies et plus rigoureuses de ce cadre d'apprentissage.

La plateforme fournit une langue BabyAI compositionnelle avec 10^{19} instructions, dont la sémantique est précisément définie dans un environnement partiellement observable. Nous rapportons des résultats de référence sur la quantité de supervision nécessaire pour enseigner à l'agent certains sous-ensembles de la langue BabyAI avec différentes méthodes de formation, telles que l'apprentissage par renforcement et l'apprentissage par imitation.

Dans le troisième article, nous étudions la généralisation systématique des modèles de réponse visuelle aux questions (VQA). Dans le scénario VQA, le système doit répondre aux questions compositionelles sur les images. Nous construisons un ensemble de données de questions spatiales sur les paires d'objets et évaluons la performance des différents modèles sur les questions concernant les paires d'objets qui ne se sont jamais produites dans la même question dans la distribution d'entraînement. Nous montrons que les modèles dans lesquels les significations des mots sont représentés par des modules séparés qui effectuent des calculs indépendants généralisent beaucoup mieux que les modèles dont la conception n'est pas explicitement modulaire. Cependant, les modèles modulaires ne généralisent bien que lorsque les modules sont connectés dans une disposition appropriée, et nos expériences mettent en évidence les défis de l'apprentissage de la disposition par un apprentissage de bout en bout sur la distribution d'entraînement. Dans notre quatrième et dernier article, nous étudions également la généralisation des modèles VQA à des questions en dehors de la distribution d'entraînement, mais cette fois en utilisant le jeu de données CLEVR, utilisé pour les questions complexes sur des scènes rendues en 3D. Nous générons de nouvelles questions de type CLEVR en utilisant des références basées sur la similitude (par exemple " la balle qui a la même couleur que ... ") dans des contextes qui se produisent dans les questions CLEVR mais uniquement avec des références basées sur la localisation (par exemple " le balle qui est à gauche de ... "). Nous analysons la généralisation avec zéro ou quelques exemples de CLOSURE après un entraînement sur CLEVR pour un certain nombre de modèles existants ainsi qu'un nouveau modèle.

Mots-clefs: apprentissage profond, compréhension du langue ancré, généralisation systématique, efficacité de l'échantillon, modèles de suivre des instructions, modèles de réponse visuelle aux questions.

SUMMARY

By using the methodology of deep learning that advocates relying more on data and flexible neural models rather than on the expert's knowledge of the domain, the research community has recently achieved remarkable progress in natural language understanding and generation. Nevertheless, it remains unclear whether simply scaling up existing deep learning methods will be sufficient to achieve the goal of using natural language for human-computer interaction. We focus on two related aspects in which current methods appear to require major improvements. The first such aspect is the data inefficiency of deep learning systems: they are known to require extreme amounts of data to perform well. The second aspect is their limited ability to generalize systematically, namely to understand language in situations when the data distribution changes yet the principles of syntax and semantics remain the same.

In this thesis, we present four case studies in which we seek to provide more clarity regarding the aforementioned data efficiency and systematic generalization aspects of deep learning approaches to language understanding, as well as to facilitate further work on these topics. In order to separate the problem of representing open-ended real-world knowledge from the problem of core language learning, we conduct all these studies using synthetic languages that are grounded in simple visual environments.

In the first article, we study how to train agents to follow compositional instructions in environments with a restricted form of supervision. Namely for every instruction and initial environment configuration we only provide a goal-state instead of a complete trajectory with actions at all steps. We adapt adversarial imitation learning methods to this setting and demonstrate that such a restricted form of data is sufficient to learn compositional meanings of the instructions. Our second article also focuses on instruction following. We develop the BabyAI platform to facilitate further, more extensive and rigorous studies of this setup. The platform features a compositional Baby language with 10^{19} instructions, whose semantics is precisely defined in a partially-observable gridworld environment. We report baseline results on how much supervision is required to teach the agent certain subsets of Baby language with different training methods, such as reinforcement learning and imitation learning.

In the third article we study systematic generalization of visual question answering (VQA) models. In the VQA setting the system must answer compositional questions about images. We construct a dataset of spatial questions about object pairs and evaluate how well different models perform on questions about pairs of objects that never occured in the same question in the training distribution. We show that models in which word meanings are represented by separate modules that perform independent computation generalize much better than models whose design is not explicitly modular. The modular models, however, generalize well only when the modules are connected in an appropriate layout, and our experiments highlight the challenges of learning the layout by end-to-end learning on the training distribution. In our fourth and final article we also study generalization of VQA models to questions outside of the training distribution, but this time using the popular CLEVR dataset of complex questions about 3D-rendered scenes as the platform. We generate novel CLEVR-like questions by using similarity-based references (e.g. "the ball that has the same color as ...") in contexts that occur in CLEVR questions but only with location-based references (e.g. "the ball that is to the left of ..."). We analyze zero- and few- shot generalization to CLOSURE after training on CLEVR for a number of existing models as well as a novel one.

Keywords: Deep learning, grounded language understanding, systematic generalization, sample efficiency, instruction following, visual question answering.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ACRONYMS AND ABBREVIATIONS

A2C	Advantage Actor-Critic
A3C	Asynchronous Advantage Actor-Critic
AGILE	Adversarial Goal-Induced Learning from Examples
BiRNN	Bidirectional Recurrent Neural Networks
CAN	Compositional Attention Networks
CNN	Convolutional Neural Network
DL	Deep Learning
FiLM	Feature-wIse Linear Modulation
GAIL	Generative Adversarial Imitation Learning
GRU	Gated Recurrent Unit
IL	Imitation Learning
LSTM	Long Short-Term Memory
MAC	Memory-Attention Composition
MDP	Markov Decision Process
ML	Machine Learning
MLP	Multi-Layer Perceptron
NLP	Natural Language Processing
NMN	Neural Module Networks
NS-VQA	Neural-Symbolic Visual Question Answering
POMDP	Partially-Observable Markov Decision Process
PPO	Proximal Policy Optimization
RE	Referring Expression
RL	Reinforcement Learning
RNN	Recurrent Neural Network
SL	Supervised Learning
SQOOP	Spatial Queries On Object Pairs
VQA	Visual Question Answering

ACKNOWLEDGEMENTS

It has been my enormous privilege to work with and receive advice from so many generous, inspiring and talented people on my way from discovering affection to programming and math to writing this PhD thesis.

I am very grateful to my advisor Yoshua Bengio for giving me the chance to do my PhD research in the unique environment of what used to be called LISA and has now become a part of Mila. His support, encouragement, and enthusiasm have been very valuable for me during my PhD years. I am likewise deeply thankful to my "unofficial co-advisor" Aaron Courville for advising me in my systematic generalization research. His commitment to discuss both the high-level aspects and low-level technical details of these studies over and over again made a huge difference to the quality of the end result. I would like to thank Edward Grefenstette for his great mentorship during my internship at DeepMind and the extra effort he put into wrapping up our AGILE project after the internship was over. I am grateful to Quoc Le for supporting and encouraging me during my internship at Google. Lastly, my MSc advisor Herbert Jaeger gets a lot of credit for contributing into my development as a researcher and a critical thinker. The memories of our reading meetings with coffee and gummy bears are still very dear to my heart.

In addition to the people who mentored me, I would like to thank the coauthors of the articles that make up this thesis: Felix Hill, Jan Leike, Edward Hughes, Arian Hosseini, Pushmeet Kohli, Maxime Chevalier-Boivert, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, Shikhar Murty, Michael Noukhovitch, Harm de Vries, Timothy O'Donnell, and Philippe Beaudoin. I am also very grateful to my earlier collaborators: Jan Chorowski, Dmitry Serdyuk, Philemon Brakel, Ryan Lowe, Tom Bosc, Stan Jastrzebski, Rosemary Ke, Jackie Cheung, and Joelle Pineau. It was a privilege and a great learning experience to work with all these inspiring, knowledgeable, and hard-working people.

I would like to further thank Luke Vilnis, Çaglar Gulcehre, Bart van Merriënboer, David Warde-Farley, Ziyu Wang and countless others for valuable interactions and discussions. Sometimes all you need to break an impasse is just to talk to the right person, and I feel lucky that I had a plenty of such opportunities during my PhD time.

I am grateful to my friends, especially Ben, Mathieu, and Satya, but also all others, for filling my life in Montreal with warmth and joy. C'est grace a Satya que j'ai finalement commencé parler français recemment, et je lui en suis vraiment reconnaissant.

Last, but not least, I would like to thank my family. My father Barys put a huge effort into developing my logical reasoning and mathematical skills starting from the tender age of six. My mother Dora and my sister Maryna were always supportive of my PhD ambitions.

Finally, words can not express how thankful I am to my dear wife Veranika for her patience, encouragement, and love. I will forever deeply appreciate her loving support over these years, bearing with internships, deadlines, and other hardships of the PhD life, as well as cheering me up after another batch of conference reviews came out.

Chapter 1

INTRODUCTION

Building machines that can understand natural language has been a key goal of artificial intelligence (AI) since the legendary Darmouth Proposal that is widely regarded as giving birth to the field (McCarthy et al., 1956). Initially, researchers approached this grand challenge by programming, that is by defining precisely all steps and aspects of the computation that took language as input and produced the desired response or behavior. In their programming efforts these pioneers were guided by formal theories of syntax, developed by linguists, and formal logical approaches to meaning, developed by mathematicians and philosophers. Some achievements from these early days remain impressive even now, e.g. the SHRDLU (Winograd, 1972) program that could engage with a human in a lengthy and contextualized conversation about a block world. Despite a handful of successful demonstrations as the one noted above, this classical approach to language understanding hit the same major roadblock of ever-growing program complexity as the attempts to model other human abilities, judgements, and behaviours. The endeavor to specify precisely how ambiguities and irregularities of real world language use should be dealt with in any possible context is seen by many, including the author of this thesis, as rather futile.

The alternative to fully codifying one's understanding of reality in a model (such as e.g. a program) is to let the model directly adapt to, or differently put, learn from, the reality. This general idea of learning from data lies at the foundation of the closely related disciplines of statistics and machine learning. Known under different names, data-driven methods have permeated numerous areas of science and engineering, natural language processing (NLP) among others. Rule extraction, estimating rule probabilities for stochastic grammars, counting frequencies of word correspondences are examples of numerous different forms of learning in classical NLP models, to name but a few (see Manning and Schütze (1999) for a review).

The subject of this thesis is the more recent deep learning approach to NLP. Deep learning (DL) is a general machine learning (ML) paradigm that advocates a learning-centered approach to building intelligent systems. While other ML methodologies often view learning

from data as a supplement to codifying our understanding of intelligence, DL advocates that the ability to learn and generalize should be the key priority in model design (Bengio, 2009). Key ideas and practices of deep learning stem from the artificial networks of neuron-like units (neural networks), that have been known since 80s as a model class with potentially unlimited learning capabilities (Rumelhart et al., 1986a; Cybenko, 1989). During the last decade this potential was finally realized thanks to the increased amounts of data and computational resources, as well as technical advances in neural network design (Glorot et al., 2011), regularization (Srivastava et al., 2014) and training (Ioffe and Szegedy, 2015). The unparalleled ability to learn and the plug-and-play ease with which neural building blocks (pretrained or not) can be recombined have made deep learning the default machine learning method for the cases when data is sufficient.

When the deep learning approach is applied to understanding or generating language, little space is left for linguistic considerations regarding syntax, semantics and pragmatics[1]. Instead, texts and utterances are viewed as data, namely as sequences of discrete tokens, and neural models are trained to learn dependencies in such sequential data from huge datasets of millions or even billions of words. Natural to some and counter-intuitive to others, the deep learning approach to NLP turned out to be extremely effective. Its greatest success story, arguably, is the wide-spread practical adoption of neural machine translation (Kalchbrenner and Blunsom, 2013; Cho et al., 2014; Sutskever et al., 2014; Bahdanau et al., 2015) and the resulting impressive improvement in the quality of automatic translations (Wu et al., 2016). In addition to this indisputable real-world impact, most (if not all) academic evaluations of language understanding or generation systems nowadays feature deep learning models in the top of the leaderboard.

While there are definitely valuable lessons to be learned from the DL revolution in NLP, it remains unclear whether it concludes the original quest for artificial systems that understand natural language. To avoid the philosophical debate about what it means to understand language, we pose a narrow engineering-minded formulation of this question: are existing DL methods sufficient to let humans productively interact with machines in natural language? Two following considerations suggest that the answer to this question might be "not yet". First, the blessing of unlimited learning abilities comes with the curse of sample inefficiency: DL systems require extreme amounts of data for best performance. While raw textual data is essentially unlimited and some other kinds of data (e.g. pairs of sentences from different languages) are relative easy to collect, it will be arguably much harder to collect millions of naturalistic examples of productive interaction using language. A related concern is that in adapting to the training data DL systems take advantage of whatever dependencies it

[1]Here we are talking about the mainstream models, such as e.g. LSTMs and Transformers (see Chapter 2). Integrating deep learning and linguistic considerations is a vibrant research area, but most of the impact outside academia currently comes from the aforementioned linguistics-free models.

contains. Many recent studies suggest that instead of learning broadly applicable principles of language, DL models often exhibit high empirical performance by exploiting dataset-specific regularities (Jia and Liang, 2017; McCoy et al., 2019). The spurious solutions learned thereby generalize badly when the data distribution changes. Hence, the expensive massive interactive data collection would have to be repeated as for many slightly different human-computer interaction scenarios.

We believe that the low sample efficiency and weak generalization of DL models for language understanding need to be addressed in order for conversational interfaces to computers and robots to empower human users and make them more productive. But how should research on these issues be conducted in order to be effective? The conventional approach of the NLP community is to evaluate systems on their ability to deal with real world texts coming from "naturally occurring unconstrained materials" (Marcus et al., 1993). Such unconstrained texts come from books, newspapers, websites, reports and other sources; they refer to a broad range of phenomena, events and activities humans individually or collectively engage in. Understanding such texts requires the system to represent and reason using knowledge about all domains of human life. It is therefore unclear what is measured by holistic evaluation on unconstrained texts: the model's grasp of generally applicable principles of natural language or the amount of world knowledge that it is able to access in some, possibly very superficial way. Crude models that can superficially process vast corpora can be championed by this evaluation methodology (Halevy et al., 2009). For example, n-gram counting approaches dominated the language modeling landscape for decades, despite the fact that obviously, the writer's choice of the word depends on more than the last 5-7 words.

The research presented in this thesis takes a different approach: it studies learning of synthetic natural-like languages that are grounded in simple visual environments. The hypothesis underlying the switch to synthetic data is that the amount and the kind of world knowledge that our evaluation methods require should be tightly controlled in order facilitate progress in data efficiency and generalization of language understanding models. It is likely that the path to improvements in these aspects of models' performance goes through having them learn something akin to the rules for composing meanings that are subject of formal semantics, a subfield of linguistics that attempts to explain precisely how humans construct meanings of utterances from meanings of individual words (Heim and Kratzer, 1998). The evaluation methods that use unconstrained natural text might not guide us well along this path, and instead distract researchers by tempting them to focus on knowledge mining from vast corpora, for this is what such benchmarks are particularly sensitive to.

Using synthetic data may seem like going against the rise of empiricism in NLP and AI in general that led to the radically data-driven deep learning models studied in this thesis. We would like to emphasize that we are not advocating here to go back to hand-designed rule-based systems, despite the fact that such systems could easily handle the synthetic grammars

used here and in similar studies. Trying to manually scale up such systems to handle all richness of natural language appears to be a dead-end. One immediate challenge is structural ambiguity: a sentence can have multiple correct parses, as illustrated by "time flies like an arrow" (which surprisingly can be parsed in the same way as "fruit flies like a banana", (Pinker, 1994)) and "the children ate the cake with the spoon" (which unexpectedly can be read as eating the cake together with the spoon, (Manning and Schütze, 1999)). Other is idioms: as "kick the bucket" does not mean the same thing as "kick the ball". Further challenges to the manual approach are pronoun resolution and elliptical constructions (such as "Other [challenge] is" in the previous sentence: the word "challenge" is omitted but is clear from the context). These phenomena and many others, as well as their interactions, do seem to be hard to address by manual modelling or by overly restricted classes of learning-based models, such as e.g. linear probabilistic context free grammars (Manning and Schütze, 1999). Deep learning models, on the other hand, can successfully tackle these challenges where they are provided enough empirical evidence. Yet, as we argued above, their data inefficiency and generalization capabilities are often dissatisfying.

Our reasoning for using synthetic languages for testing learning-based models is thus as follows. As long as neural models struggle to learn basic compositional rules that underlie simple synthetic languages in a generalizable way, it is likely that same keeps occuring when they are trained on natural data. On the other hand, if improving the models' generalization abilities in the controlled synthetic setups (that unlike natural data allow precise multifaceted analysis of generalization) can be done without sacrificing their flexibility and expressiveness, the gains for the field of NLP, and in especially its data-scarce usecases, could be dramatic.

As mentioned above, the meaning of words and utterances in the synthetic languages that we consider are grounded in perception and action. We judge that a model has learned a language if it reliably succeeds at following instructions in or asking questions about an environment that it visually perceives. A possible alternative approach would be to ground the language that the model is supposed to learn in a purely symbolic universe described by an ontology and a set of rules (see e.g. (Weston et al., 2016) and (Côté et al., 2018)). We have chosen, however, to embrace the challenge of grounding symbols in perception instead of shying away from it. It has been argued that certain core kinds of world knowledge may be challenging, if not impossible, to acquire from symbolic inputs alone (Harnad, 1990). Apart from this speculative consideration that concerns language understanding in general, there is also a practical argument for studying grounding: it will certainly be required from productive human-robot interaction in unstructured environments.

1.1. Thesis Structure

This thesis presents four articles in which we study grounded language learning with neural models. Chapter 2 covers the shared background of the four articles. Chapters 3 to

10 contain the articles themselves, as well as prologues for each of them. Lastly, Chapter 11 presents the general conclusion of the thesis.

Chapter 2

BACKGROUND

2.1. Machine Learning

At a high level, this thesis studies computational models that can be automatically tuned using data. Therefore, the broad academic discipline that this thesis fits best is *machine learning* (ML). It should be noted that ML heavily overlaps with a number of other disciplines that also study data-driven computational methods, including statistics and data mining. Since the articles we present here are written according to the standards of the ML community, it is the ML terminology and concepts that we will use throughout this thesis.

As the name may suggest, machine learning studies machines that learn. These "machines" are not physical devices, they are abstract computational models that are capable of automatic adaptation from experience. It is this adaptation process that is called "learning" or "training", depending on whether the machine is viewed as the subject or as the object.

While one can attempt to formalize learning in general (see e.g. (Mitchell, 1997)), it is much more common in ML to consider several distinct formal setups that make different assumptions about the nature of data, the goal of learning, and the possibility of interaction with the environment. We will introduce only two of these setups here, namely supervised learning and reinforcement learning, for these are the two main kinds of learning that can be found in the articles of this thesis.

2.1.1. Supervised Learning

The goal of *supervised learning* (SL) is to learn a mapping from a sample of input-output pairs. Crucially, we want the learned mapping to produce correct outputs for the inputs that were not seen during learning. A canonical example of a supervised learning problem is handwritten digit classification: given a dataset of handwritten digit images and the respective labels "0", "1", ..., "9", the task is to build a system that can label such images with as a few mistakes as possible.

SL allows a very clear formalization as follows. Let X be the space of all inputs and Y be the space of all outputs. Let $p(x, y)$ be the density (or probability in the discrete case) of the joint distribution of (x, y) pairs, $x \in X$, $y \in Y$. Lastly, let F be the family of candidate $X \to Y$ mappings that we consider and $L(\hat{y}, y)$ be the *loss* that we suffer for outputting $\hat{y}$ when the true output is y. We will focus on the mapping families that can be indexed using a parameter vector $\theta \in \mathbb{R}^n$ where n is a finite number of parameters. It is common to refer to F as the *model* and to θ as *model parameters* and to write f_θ to denote the mapping with parameters θ.

Using this notation, we can express the expected loss associated with a mapping $f \in F$ as $R(f) = \mathbb{E}_{x,y\sim p} L(f(x), y)$, an expression called *risk*. The task of supervised learning is to choose an f with a small $R(f)$ only using a sample $D = \{(x_i, y_i)\}_{i=1}^N$, where $(x_i, y_i) \sim p$. The sample D is typically called the *training set*. To achieve this goal most approaches target the average loss on the training set $\hat{R}(f) = \frac{1}{N} \sum_{i=1}^N L(f(x_i), y_i)$, which is called *empirical risk*. Naively choosing f with the lowest $\hat{R}(f)$ does not guarantee that $R(f)$ is low. Various approaches are thus used to prevent *overfitting*, i.e. a situation where $\hat{R}(f)$ is low and $R(f)$ is high. Choices include adding additional terms to $\hat{R}(f)$ such as e.g. L2 or L1 penalty (Tibshirani, 1994), restricting explicitly the model F, and in the context of neural networks, adding noise to weights or activations (Graves, 2011; Srivastava et al., 2014) . The umbrella term for techniques to bridge the discrepancy between $\hat{R}(f)$ and $R(f)$ is *regularization*.

A very common practice in SL is to use a per-example surrogate loss $l_\theta(x, y)$ instead of the task loss $L(f_\theta(x), y)$, and to use its respective average $\tilde{R}(f_\theta) = \frac{1}{N} \sum_{i=1}^N l_\theta(x_i, y_i)$ instead of $\hat{R}$. For one, $\hat{R}(f)$ is often hard to optimize directly, for example when Y is a discrete space, and consequently, L and $\hat{R}$ are discontinuous with respect to the model parameters θ. A typical example of such a situation is a classification problem with the zero-one loss $L_{01}(\hat{y}, y)$, which is equal to 0 when $\hat{y} = y$ and 1 otherwise. For two, substituting $\tilde{R}$ for $\hat{R}$ often has a desirable regularization effect. A very common surrogate loss is the cross-entropy loss. It is defined as $-\log q_\theta(y|x)$ for the cases where the model defines a conditional distribution $q_\theta(y|x)$ and where the mapping f selects (sometimes approximately) the output with the largest probability, $f(x) = \arg\max_{y'} q_\theta(y'|x)$.

2.1.2. Reinforcement Learning

The supervised learning setup assumes that the training set D of inputs and the corresponding outputs was collected prior to training to the model. The *reinforcement learning* (RL) setup gives the model more agency. An RL *agent* collects its own training data by acting in the *environment* and receiving *rewards* from it. The state of the environment serves as the input to the RL agent and the action that it chooses can be viewed as the output. A key difference between RL and SL is thus that the optimal output is not given for each input, instead it has to be found by the agent by trial-and-error, using rewards as the

guidance. Other characteristic traits of the RL setup are its temporal and stochastic aspects: the states in which the agent finds itself depend on the actions that the agent took earlier as well as on the (sometimes) stochastic dynamics of the environment. Below we introduce the key concepts of RL in a more formal way, mostly adapting the parts of the narrative from (Sutton and Barto, 2018) that are relevant for policy gradient methods.

The canonical formalization of RL relies on Markov Decision Processes (MDP) as the abstraction for the environment. Let $\mathcal{S}$ be the *state space* and $\mathcal{A}$ be the *action set*. The MDP is defined by the initial state distribution $p_0(s)$ and a transition probability distribution $p(s', r|s, a)$ for the next state $s' \in \mathcal{S}$ and the reward $r \in \mathbb{R}$, given that the agent took the action $a \in \mathcal{A}$ in the state $s \in \mathcal{S}$. Note that $p(s', r|s, a)$ is not a conditional probability in the usual sense[1], the vertical bar symbol "|" is only used here to indicate that $p(s', r|s, a)$ must be a proper probability distribution for each (s, a) pair. A decision-making agent that operates in an MDP is only allowed to base its choice of action on the MDP's current state s. The agent's behaviour is thus defined by an (optionally) stochastic state-to-action mapping called the *policy* and denoted $\pi(a|s), a \in \mathcal{A}, s \in \mathcal{S}$.

The agent interacts with the MDP as follows. First, the initial state S_0 is sampled from $p_0(s)$. At each time step $t = 0, 1, 2, 3, \ldots$ the agent chooses an action $A_t \sim \pi(A_t|S_t)$, to which the environment reacts by transitioning into the next state S_{t+1} and rewarding the agent with the reward R_{t+1}, $(S_{t+1}, R_{t+1}) \sim p(s', r|S_t, A_t)$. The resulting sequence $S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, \ldots$ is called a trajectory and denoted as τ. We will use the notation $\tau \sim \pi$ to refer to the trajectory generation process described above.

The goal of RL is improve the policy π. The optimality of the policy is measured by the expected return g, which is defined as the discounted sum of all future rewards:

$$g = \mathop{\mathbb{E}}_{\tau\sim\pi} \sum_{t=1}^{\infty} \gamma^t R_t, \tag{2.1.1}$$

where $\gamma \in [0; 1)$ is the so-called discount rate. Two other concepts that are related to the expected return g and will be helpful for further exposition are state- and state-action- value functions $v(s)$ and $q(s, a)$ respectively. They correspond to the agent's discounted expected return under different initial conditions. Namely, $v(s)$ is the expected return that the agent would receive if it started in s, and $q(s, a)$ assumes additionally that the agent's first action was a. Formally:

$$v(s) = \mathop{\mathbb{E}}_{\tau\sim\pi, S_0=s} \sum_{t=1}^{\infty} \gamma^t R_t, \tag{2.1.2}$$

$$q(s, a) = \mathop{\mathbb{E}}_{\tau\sim\pi, S_0=s, A_0=a} \sum_{t=1}^{\infty} \gamma^t R_t, \tag{2.1.3}$$

[1]An MDP together with a policy can be used to define a joint distribution $p(s', r, s, a)$. The transition probability distribution $p(s', r|s, a)$ could be considered a conditional for this joint. But given an MDP alone, $p(s', r|s, a)$ is just family of distributions indexed by an (s, a) tuple.

where $S_0 = s$ and $A_0 = a$ denote the conditions described above. Note, that g, v and q depend on π, but we will not indicate this dependency in the notation to keep it light-weight.

We asssume that in practice the interaction between the agent and the environment will be episodic. Specifically, the agent is allowed to act for T steps, after which the environment is restarted by drawing an initial state $S_0 \sim p_0(s)$. The expected return for such an episodic setting could be defined without discounting, as $g = \mathbb{E} \sum_{t=1}^{T} R_t$, but we chose an infinite-episode optimization objective g in Equation 2.1.1 to make sure that the state values $v(s)$ are stationary, i.e. do not depend on the time t.

Theoretical studies of RL often consider finite MDPs in which $\mathcal{S}$ and $\mathcal{A}$ are finite. In this case, the policy π can be represented as a probability table. In most applications of RL, the state space is effectively infinite, and is often represented by high-dimensional vectors. In this case more advanced machine learning models (such as e.g. neural networks) can be used to represent the policy π and extrapolate the learned behavior from the subset of states seen during training to all possible states.

A wide variety of RL algorithms has been proposed, and here we will give just one example, the advantage actor-critic algorithm (A2C, (Mnih et al., 2016; Schulman et al., 2015)). This algorithm trains a policy π and a value function estimate $\hat{v}$ that are parametrized by vectors $\theta \in \mathbb{R}^{D_\theta}$ and $\phi \in \mathbb{R}^{D_\phi}$ respectively. A2C belongs to the family of policy gradient methods as it relies on computing an approximation $\frac{\hat{dg}}{d\theta}$ of the gradient $\frac{dg}{d\theta}$ of the expected reward g. Once such an approximation is computed, the policy can be improved by performing an approximate gradient descent step, $\theta \leftarrow \theta + \alpha \frac{\hat{dg}}{d\theta}$, where α is the step size. The policy gradient theorem (Sutton et al., 2000) provides an expression for $\frac{dg}{d\theta}$ that is often used to construct gradient estimates:

$$\frac{dg}{d\theta} = \mathop{\mathbb{E}}_{s \sim \rho_{\pi,\gamma}} \sum_{a \in \mathcal{A}} q(s,a) \frac{d\pi(a|s)}{d\theta}, \tag{2.1.4}$$

where $\rho_{\pi,\gamma}$ is the γ-discounted distribution of states that the agent visits. This can be further rewritten as an expectation over trajectories

$$\frac{dg}{d\theta} = \mathop{\mathbb{E}}_{\tau \sim \pi} \sum_{t=0}^{\infty} \gamma^t q(S_t, A_t) \frac{d \log \pi(A_t|S_t)}{d\theta}, \tag{2.1.5}$$

To turn this expression into a gradient estimate, we replace $q(S_t, A_t)$ with a K-step approximation:

$$\hat{q}(S_t, A_t) = \sum_{i=t+1}^{t+K} \gamma^{i-t-1} R_i + \gamma^K \hat{v}(S_{t+K+1}). \tag{2.1.6}$$

where K is a hyperparameter. Furthermore, $\hat{v}(S_t)$ is subtracted at each step from $\hat{q}(S_t, A_t)$ as an action-independent baseline to reduce the estimate's variance without adding any extra

bias (Williams, 1992). The resulting estimate is as follows:

$$\frac{\hat{dg}}{d\theta} \approx \sum_{t=0}^{T-1} \gamma^t (\hat{q}(S_t, A_t) - \hat{v}(S_t)) \frac{d \log \pi(A_t|S_t)}{d\theta} \tag{2.1.7}$$

Note that the state distribution discount γ^t is in practice often omitted from Equation (2.1.7):

$$\frac{\hat{dg}}{d\theta} \approx \sum_{t=0}^{T-1} (\hat{q}(S_t, A_t) - \hat{v}(S_t)) \frac{d \log \pi(A_t|S_t)}{d\theta} \tag{2.1.8}$$

This way, the agent's optimality in all states that it visits is valued equally, regardless of how long it takes to get to these states from the initial state $s_0 \sim p_0(s)$. This can be seen as an approximation to the average reward formulation of RL that we do not cover here (Thomas, 2014). Another perspective is that Equation (2.1.8) can be viewed as a biased gradient of the undiscounted return (provided that such return is finite, Schulman et al. (2015)).

To complete the description of the algorithm, we have to specify how the value function estimate $\hat{v}$ is trained. This is typically done using a technique called temporal difference or bootstrapping (Sutton, 1988), where a target for the value estimate $\hat{v}(S_t)$ at time t is constructed using the estimate $\hat{v}(S_{t'})$ at a later time $t' > t$. In the case of A2C, $\hat{q}(S_t, A_t)$ can serve as such a target, leading to the following loss for $\hat{v}$:

$$L_{\hat{v}}(\phi) = \sum_{t=0}^{T-1} (\hat{v}(S_t) - \hat{q}(S_t, A_t))^2 \tag{2.1.9}$$

The gradient of L_{hatv} with respect to ϕ can then be used for improving the value function estimate $\hat{v}$.

The basic RL setup and the A2C algorithm that we have covered so far assume that at each step the agent observes the whole state of the MDP, which by virtue of the Markov assumption contains all useful information about the future. This might be not the case in many real-world scenario, e.g. a robot navigating in a house will only perceive the room in which it is located, but the best behavior for it might depend on what is in the other rooms. Such partially-observable, as opposed to fully-observable environments, are covered by a different mathematical abstraction, namely Partially-Observable Markov Decision Processes (POMDP, see (Monahan, 1982) for a survey). A key difference of a POMDP from MDP is that at each step the agent receives an observation O_t that contain only partial information about the state S_t. While POMDPs pose a much bigger challenge for theoretical analysis, for many practical usecases it is often sufficient to consider the history of observations $(O_0, O_1, O_2, \ldots)$ as the state and use the same RL algorithms as the ones used by POMDP.

2.2. Deep Learning

The previous section has given two examples of how learning can be formulated as optimization. In covering both the SL and RL formulation we abstracted away from the details

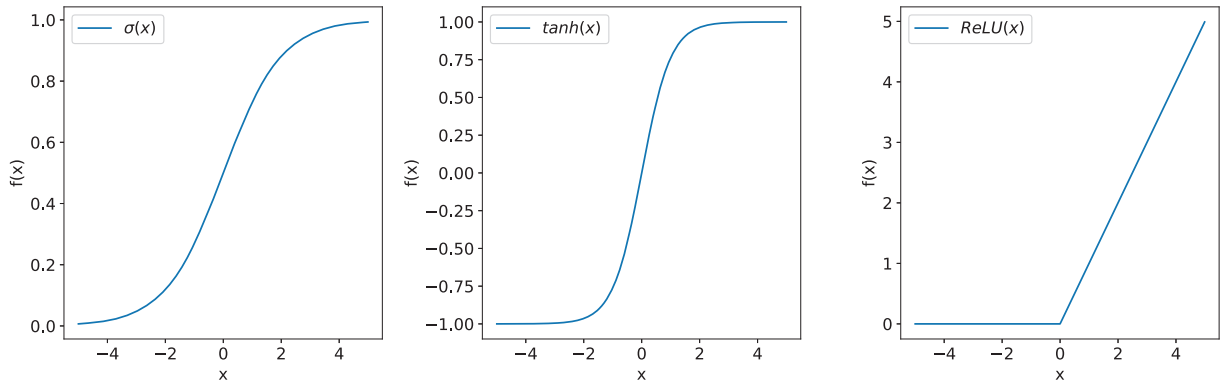


FIGURE 2.1. Popular activation functions. Sigmoid $\sigma(x) = \frac{1}{1+\exp -x}$ is on the left, hyperbolic tangent is in the middle, rectifier nonlinearity $ReLU(x) = \max(0, x)$ is on the right.

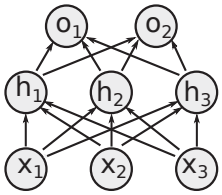


FIGURE 2.2. An MLP with 3 inputs, 3 hidden units and 2 output units. For simplicity we omit the upper index in h_i^1.

of what the optimized models do internally. Here, we give a quick introduction to the broad deep learning (DL) family of ML models that are the subject of this thesis. The majority of DL models are neural networks, although it should be noted that the meaning of the term "neural network" has been expanding over years. We will start from the simplest and historically first neural networks models and gradually transition to the more sophisticated ones.

2.2.1. Neural Networks

What is known as an artificial neural network (or just *neural network*) in the machine learning community traces its roots to investigations of psychologists and cognitive scientists (Hebb, 1949; Rosenblatt, 1958), in particular to the research tradition called *connectionism* (Rumelhart et al., 1986b). Connectionist argued that parallel distributed processing in large networks of homogeneous units might be an appropriate framework for modelling cognition, thereby challenging the dominant symbol manipulation paradigm (Newell and Simon, 1976; Fodor, 1975). Since the adoption of neural networks as ML models in late 80s, ML researchers have been developing their own perspective and terminology, to which I will adhere in this section.

The simplest example of a neural network is a fully-connected feedforward neural network, which is also often called Multi-Layer Perceptron (MLP). An MLP consists of an input layer, a number of hidden layers and an output layer. Each *layer* comprises a number of *units* (or *neurons*). We will use $m_0, m_1, \ldots m_{K+1}$ to refer to the number of units in the input layer, in each of the K hidden layers, and in the output layer. Each unit has an *activation* value associated with it. The activation values $x_1, \ldots, x_{m_0}$ of the input units represent the input to the network. The units of the first hidden layer compute their activations h_j^1 by applying their *activation function* f to a biased weighted sum of the input layer activations:

$$h_j^1 = f(\sum_i w_{ij}^1 x_i + b_j^1), \tag{2.2.1}$$

where $w_{i,j}^1$ are the unit-specific *weights* and b_j^1 is the unit-specific *bias* term. The units of the k-th layer perform the same computation but using the activations of the $k-1$-th layer instead of the inputs. While in theory units at different layers may have different activation functions, all hidden layers typically use the same one. Common choices for the activation function of hidden units include the rectifier $ReLU(x) = \max(0, x)$, the sigmoid $\sigma(x) = 1/(1+e^{-x})$ and the hyperbolic tangent, see Figure 2.1. Finally, units of the output layer are typically similar to the hidden units but use a different activation function. For example, when an MLP is used for classification, it is common to use the softmax operation to compute the activations of output units:

$$o_j = \exp\left[\sum_i w_{ij}^{K+1} h_i^K + b_j^K\right] / \left(\sum_{j'} \exp\left[\sum_i w_{ij'}^{K+1} h_i^K + b_{j'}^K\right]\right). \tag{2.2.2}$$

The outputs o_j can in this case be interpreted as a conditional probability distribution $q(j|x)$, where x is the network input. Figure 2.2 illustrates a basic MLP with one hidden layer.

Neural networks can also be described using the concise and expressive language of linear algebra. For example, the computation performed by an MLP with a softmax output layer can be written as

$$o = \text{softmax}(f(f(\ldots f(xW^1 + b^1)\ldots)W^K + b^K)W^{K+1} + b^{K+1}), \tag{2.2.3}$$

where o is the output vector, W^1, ..., W^{K+1} are weight matrices, b^1, ..., b^{K+1} are bias vectors. From the ML perspective, the weight matrices and bias vectors are the parameters θ of the MLP, as well as of the conditional distribution $q_\theta(j|x) = o_j$ that the MLP implements.

In order to train the MLP, one has to define a differentiable loss function $\tilde{R}(\theta)$, and most importantly a procedure to compute exactly or approximately its gradient $\frac{d\tilde{R}}{d\theta}$. In the case of supervised learning the loss function is typically defined as the average loss on a set of examples, $\tilde{R}(\theta) = \frac{1}{N}\sum_{i=1}^{N} l(o, y_i)$, where y_i is the desired output for the input x_i, $o = o(x_i)$ is

the network's vector-shaped output. The gradient is then computed as follows:

$$\frac{d\tilde{R}}{d\theta} = \frac{1}{N}\sum_{i=1}^{N}\frac{dl(o, y_i)}{do}\frac{do}{d\theta}, \tag{2.2.4}$$

where the latter term $\frac{do}{d\theta}$ is computed using the famous back-propagation algorithm (Rumelhart et al., 1986a; Werbos, 1974).

This gradient computation procedure is typically used to perform a version of stochastic gradient descent using small batches of examples. The reader is referred to (Goodfellow et al., 2016) for an overview of optimization procedures that are commonly used to train neural networks.

Despite its simplicity, the MLP is very expressive model. The famous universal approximation theorem (Cybenko, 1989) states that any continuous function can be approximated with an MLP with just one hidden layer, provided that it has enough hidden units. Notably, the Multi-Layer Perceptron's single-layer predecessor that was named Perceptron (Rosenblatt, 1958) was unable to approximate certain basic functions, such as e.g. XOR of binary inputs, (Minsky and Papert, 1969). The universal approximation theorem shows how adding just one hidden layer makes a huge difference. It has been further argued that deeper neural networks that with more than one hidden layer are more appropriate for modelling complex functions that are characteristic of intelligence, such as e.g. transformations from raw sensory inputs to abstract categories (Bengio, 2009). These considerations about the role of depth have given rise to the term "deep learning" that is now ubiquitously used to refer to neural networks, as well as to other models that involve chaining trainable non-linear transformations.

Let us pause and reflect on what an MLP is and what it can be used for. In principle, an MLP can be taught to perform any task that can be represented as transforming a vector into another vector. For example, images of handwritten digits can be represented as vectors by concatenating all rows in the image. The digit's category can be represented by a vector that has zeros in all components but for one, a so-called *one-hot vector*. Given enough data in that form, and provided that gradient-based optimization is successful, we can hope that the trained MLP will generalize to examples beyond its training set. An MLP can furthermore learn from trial-and-error as an RL model: it is sufficient to define how the MDP's state s can be represented as a vector and how the MLP's output defines the distribution $\pi_\theta(a|s)$ over the actions a. Last, but not least, an MLP can be a building block in a bigger neural network, trained by the gradient that the said neural network backpropagates to the MLP's outputs. Such an inner MLP block can furthermore backpropagate its gradient to the MLP's inputs to enable training of the preceeding neural parts.

2.2.1.1. *Recurrent Neural Networks*

Representing the input as a fixed size vector is appropriate for some machine learning usecases and less for others. A robot's state that describes the angles and velocities of all joints is an example of data that naturally comes in a vector form. The history of a robot's observations, financial time-series, natural language sentences or passages are examples of data which is non-trivial to represent as a fixed length vector. Recurrent Neural Networks (RNNs, Elman (1990)) take data that has a temporal or sequential aspect to it and transform it into a sequence of vector representations. We will skip the connectionist background of the RNNs and proceed directly to the formula, which in the simplest case is

$$h^t = \tanh(Wh^{t-1} + Vx^t + b), \tag{2.2.5}$$

where t is the time step, $x^t \in \mathbb{R}^{m_i}$ are input vectors, $h^t \in \mathbb{R}^{m_h}$ are the RNN *states* (sometimes called *hidden states*), W and V are weight matrices and b is a bias vector, m_i and m_h stand for the numbers of input and hidden units respectively. To complete the specification of an RNN one must define the initial state h^0, for example it can be set to a zero vector or it can be trained as a parameter. A desirable characteristic of the RNNs is that states may act as memory, that is the state h^t may contain the inputs $x^1, \ldots, x^t$, assuming that the training results in a successful setting of the RNN's weights. In such case the last state can represent the whole input sequence regardless of its length, and can be used, for example as an input to an MLP classifier, like the one described above. If the input sequence is too long to fit in the last state, the intermediate states h^t may be still useful. In this case it is common to use another RNN that reads the inputs $x^1, \ldots, x^t$ backwards. A combination of the forward state h^t and the backward state $\tilde{h}^t$ summarizes a number of inputs directly preceding and directly following x^t. Such a combination of a forward RNN and a backward RNN is called a bidirectional RNN (BiRNN), (Schuster and Paliwal, 1997), and the concatenation of their states is called the BiRNN state.

The simple RNNs described above are known to be very hard to train in practice as they suffer from the problem of gradient vanishing and gradient explosion (Bengio et al., 1994). The addition of soft gates to RNNs, as first proposed in the Long Short Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) model, made them a much more practical tool. For example, a Gated Recurrent Unit (GRU) (Cho et al., 2014) has a chance to retain or reset its state by modulating its gates. The following equations describe the computation of a GRU:

$$r^t = \sigma(W_r h^{t-1} + V_r x^t + b_r), \tag{2.2.6}$$

$$z^t = \sigma(W_z h^{t-1} + V_z x^t + b_z), \tag{2.2.7}$$

$$h^t = z^t \odot h^{t-1} + (1 - z^t) \tanh\left(W(r^t \odot h_{t-1}) + Vv^t + b\right). \tag{2.2.8}$$

The notation in the equation above is similar to the one used in Equation (2.2.5). $\odot$ denotes elementwise vector multiplication. One can see that compared to the basic RNN described by Equation (2.2.5), GRU is enhanced by the update gates z^t and the reset gates r^t, both implemented using sigmoid units. When the update gate of the unit i is closed ($z_i^t = 1$), the activation h_{i-1}^t is retained for the next step as h_i^t. When the reset gate is open, ($r_i^t = 0$) the unit i resets its state to zero before participating in computation of the next state h^t.

2.2.1.2. *Convolutional Neural Networks*

Another important type of neural networks are convolutional neural networks (CNN), in which units are locally connected according to the topology of the domain and share weights across positions (LeCun et al., 1989). In a CNN that operates on image data the units of each layer are organized in a tensor-like three-dimensional grid with two spatial and one channel dimension. The weight $w_{dx,dy,i,j}$ of the convolutional layer defines the contribution that the j-th unit of the previous layer at position $(x - dx, y - dy)$ makes to the i-th unit of the current layer at the position (x, y). The 4D tensor $W = (w_{dx,dy,i,j})$ is often called kernel. Formally, a CNN layer can be described by the following equation:

$$h_{x,y,i}^k = f(\sum_{dx=-S}^{S} \sum_{dy=-S}^{S} \sum_{j=1}^{C_{k-1}} w_{dx,dy,i,j} h_{x-dx,y-dy,j}^{k-1}), \tag{2.2.9}$$

where h^k and h^{k-1} are the layer's input and output respectively, S is the maximum offset, C_{k-1} is the number of channels in input. A more compact way to describe what a convolutional layer does is using the convolution operator $*$ which reduces Equation 2.2.9 to $h^k = f(W * h^{k-1})$.

The main application of CNNs is computer vision (Krizhevsky et al., 2012), however, they have also been successfully used in many other domains, including speech recognition and language tasks (Waibel et al., 1989; Sercu et al., 2016; Collobert et al., 2011; Gehring et al., 2017). For more information on CNNs we refer the reader to (Goodfellow et al., 2016).

2.2.2. Deep Learning for Language

From the viewpoint of deep learning, language is just a kind data that comes in a form of a sequence or multiple sequences of discrete tokens. Here, we review key methods for combining the basic building blocks of deep learning in models that can take language as input and/or produce language as output.

As a first approximation, language can be seen as sequence of words[2] that come from a large but finite vocabulary. The first step of feeding language into a neural network is to replace each word index w with the corresponding trainable embedding $e(w)$. An alternative

[2]It would be more precise to say "sequence of moprhemes", see (Bender, 2013) for a discussion how the notion of word can contentious in many languages.

perspective at the embedding step is that the words are represented as one-hot column vectors which are then multiplied by a trainable matrix E (Elman, 1990). Due to the mechanics of matrix multiplication, this is equivalent to taking the w-th column of the matrix E.

Once the words are embedded and represented as vectors, their representations are composed to produce more vectors by further neural machinery. In what was perhaps the first success story of using neural models on natural language data, Bengio et al. (2003) concatenate embeddings of each k subsequent words w_{t-k}, ..., w_{t-1} from a text corpus and feed them into an MLP that is trained to predict the next word w_t. At each step, such an MLP defines a distribution $p(w_t|w_{t-k}, ..., w_{t-1})$. If the cross-entropy loss $-\log p(w_t|w_{t-k}, ..., w_{t-1})$ is used for each of the predicted tokens, minimizing the sum of losses for all words w_t corresponds to maximizing the joint probability $p(w_1, \ldots, w_T) = \prod_{t=1}^{T} p(w_t|w_1, \ldots, w_{t-1})$ of the observed word sequence. The resulting probabilistic language model can be used to assess the likelihood of a given word sequence or even generate new ones.

The original feed-forward neural language model described above had a separate set of weights for reading each of $e(w_{t-1})$, $e(w_{t-2})$ and etc., and besides, its memory was limited to the last k words. These shortcomings can be addressed by replacing the MLP with an RNN (Elman, 1990; Mikolov et al., 2010), a sequence of convolutional (Gehring et al., 2017) or self-attention (Vaswani et al., 2017) layers, which we will briefly discuss later.

A neural language model can be trained to generate a sequence of words given an input x by using the encoder-decoder (also called sequence-to-sequence) approach. (Kalchbrenner and Blunsom, 2013; Cho et al., 2014; Sutskever et al., 2014). The key idea is to modulate the inputs and/or the initial state of a language model based on an additional input x. As a result, the distribution that the language model defines becomes conditioned on x, namely $p(w_1, \ldots, w_T|x)$. The classic example of sequence-to-sequence learning is the machine translation model by (Sutskever et al., 2014). In that work, an LSTM-RNN first reads the input sentence $x = (x_1, \ldots, x_L)$ and then continues (keeping the state) as a language model to output probabilities of the words $y = (y_1, \ldots, y_T)$ of the output sentence, thereby implementing a conditional distribution $p(y|x) = \prod_{t=1}^{T} p(y_t|y_1, \ldots, y_{t-1}, x)$. In this case the same RNN is both the encoder, as it reads x, and the decoder, as it outputs the probability distribution $p(y|x)$, but one can also use two different RNNs for these roles (Cho et al., 2014).

Conditional language generation in the encoder-decoder paradigm can be greatly facilitated by using an attention mechanism (Bahdanau et al., 2015). Here, we will explain the dot-product attention (Luong et al., 2015) using the query-key-value terminology proposed by Vaswani et al. (2017). Given a query vector q, a set of value vectors v_i and a set of respective key vectors k_i, the attention mechanism performs the following computation

$$c = \sum_i a_i v_i, \tag{2.2.10}$$

$$a_i = \frac{\exp q_i^T k_i}{\sum_{i'} \exp q_{i'}^T k_{i'}}, \tag{2.2.11}$$

where a_i are called attention weights. Informally speaking, an attention mechanism performs a soft selection of keys k_i that match the query q and returns a mixture of values that corresponds to the soft-selected keys. In the context of encoder-decoder models, the query q is computed from the state of the decoder RNN, and the states of the encoder BiRNN serve as both keys k_i and values v_i. Note, that one vector can indeed play both key and value roles without limiting the expressive power of the model because key and value signals can be carried by orthogonal subspaces of the joint key-value space. The output c of the attention mechanism serves as an additional input to the decoder RNN. The attention mechanism hence allows the decoder RNN to dynamically create shortcuts to the relevant locations in the input by producing the query q, performing the attention and consuming its output c.

In addition to be useful for connecting the encoder and the decoder, attention can replace the RNNs as the basic building block in both of these components. In the *self-attention* mechanism that is used for this purpose, each vector from a sequence of vectors attends to all other vectors in the sequence. When the vectors are arranged as rows in a matrix H, one step of single-head self-attention can be formally described as:

$$SelfAttention(H) = \text{softmax}(W^Q H (W^K H)^T) W^V H), \tag{2.2.12}$$

where W^Q, W^K and W^V are linear transformations that are used to produce queries, keys, and values respectively, and the softmax operator is applied row-wise. The highly successful Transformer model by Vaswani et al. (2017) builds a representation for the input sequence x by starting from a sequence of word vectors $H = ([e(x_1), p_1], \ldots, [e(x_L), p_L])$ and applying multi-head self-attention and MLP layers in alternation. Here, $[e(x_l), p_l]$ stands for concatenation of a word embedding $e(x_l)$ and a position embedding p_l.

The encoder-decoder paradigm can be used to train networks to generate language based on many different kinds of inputs, including generating captions from images (Kiros et al., 2014), generating the next response given a conversation history (Vinyals and Le, 2015) and generating a summary given a document (Rush et al., 2015). While ultimately systems that can both understand and generate language would be most desirable, studying language understanding alone has certain advantages from the research perspective. A daunting challenge in language generation is evaluation, for evaluating a generated utterance is a non-trivial language understanding task on its own and automated metrics are known to be unreliable (Liu et al., 2016). If a language understanding system produces its outputs in a restricted form instead of generating a full-fledged utterance, quantitative evaluation becomes easier. For example, researchers study question answering systems that select the answer as a span in the text (Rajpurkar et al., 2016), produce a single word answer (Hermann et al., 2015) or

select the answer out of a list of candidates. The answer accuracy or F1 score in the case of span selection can then be used to assess the system's performance. Another popular setup that permits quantitative evaluation is recognizing entailment, where the system has to predict if one sentence entails another or not (Bowman et al., 2015; Williams et al., 2017). Deep learning systems excel in such pure language understanding scenarios by encoding their inputs with the neural components such as BiRNNs and Transformers and training classifiers on top of produced representations. A common approach to compensate for the relatively small size of language understanding datasets is to use pretrained word embeddings (Mikolov et al., 2013; Pennington et al., 2014) or encoders (Peters et al., 2018; Devlin et al., 2018).

Other cases in which deep learning methods can be used to process language include text classification (Socher et al., 2013), syntactic and semantic parsing (Vinyals et al., 2015; Dong and Lapata, 2016) and the traditional sequence-tagging tasks of statistical NLP (Collobert et al., 2011).

2.3. Grounded Language Understanding

The conventional approach to studying language understanding focuses on the system's ability to read text and perform inference based on what it has read. Such capabilities are very practically important, for humanity has produced vast amounts of texts and systems that automatically process these texts, such as e.g. search engines, can be highly beneficial. Still, human's ability to understand language is not just about reading: it is also (or perhaps even mostly) about referring to objects and events that we perceive and experience through our senses. Differently put, the meanings of words and expressions that we use in our daily life are very often grounded in what we perceive currently or in memory of such perceptions. It is therefore interesting to study artificial systems that can understand language that is grounded in the sensory context. While there might be less practical need for such systems at the moment, the situation will rapidly change if (or when) robotics gets more mature and robots are capable of operating autonomously in unstructured environments, such as e.g. apartments and big city streets.

Another argument for studying language grounding is that it is at the moment rather unclear how basic commonsense knowledge that is required for human-level reading can be otherwise put into language understanding systems. A class of examples that is often used to illustrate the importance of commonsense knowledge in reading are Winograd Schemas (Winograd, 1972; Levesque et al., 2012). Consider e.g. the following sentence: "The trophy would not fit in the brown suitcase because it was too big/small". Here, one needs to know that a suitcase can not be put in the trophy to correctly resolve the pronoun "it" depending on whether "big" or "small" is chosen as the last word. While this example can be solved just by knowing the sizes and purposes of trophies and suitcases, other cases may require knowing more, e.g. that suitcases are typically parallelepipeds, that they can fall when put

vertically, that trophies are often statues or cups, etc. Neither knowledge engineering or crowdsourcing efforts, like CyC (Lenat, 1995) and ConceptNet (Speer and Havasi, 2013), nor representation learning from large corpora (Peters et al., 2018; Devlin et al., 2018) can at the moment be confidently considered compehensive solutions to the problem of equipping machines with commonsense knowledge. The former struggles with the challenges of knowledge representation, ambiguity, database incompleteness and inference, while the latter suffer from reporting bias: texts tend to not state obvious facts the reader already knows (Gordon and Van Durme, 2013).

Motivated by these and other considerations, researchers in several different fields attempted to build systems that can understand languages that is grounded in (that is refers to) extra-linguistic context. We review some of this research below.

2.3.1. Before Deep Learning

The robotics community has a long tradition of building simulated or real-world robots that can follow natural language instructions. To this end, the instruction must be transformed into a plan (such as e.g. a sequence of viewpoints or robot states) that the robot is capable of executing (see e.g. Macmahon et al. (2006); Kollar et al. (2010); Tellex et al. (2011); Kollar et al. (2017)). Perhaps the most advanced and flexible framework of this kind is Generalized Grounding Graphs (G^3, Tellex et al. (2011); Kollar et al. (2017)), in which correspondence between instruction constituents (noun phrases, prepositional phrases, verb phrases, etc.) and the so-called groundings that come from the environment (objects, places, paths, events) is learned by a sparse syntax-aware graphical model. The G^3 approach allows to teach real-world robots to follow free-form instructions from several hundreds examples. To achieve such efficiency, G^3 relies heavily on preprocessing and domain-specific engineering. First, the environment is represented as a semantic map with pre-identified objects. As discussed in (Kollar et al., 2017), what is not identified as an object by the robot's perception (their examples include dirt piles, parked cars, but we would also add objects parts and object groups), can not be referred to. The objects that are identified are represented by using a fixed set of pre-defined semantic labels. Futhermore, place groundings need to be pre-specified (such as the place on top of an object, or the empty space between the objects). The geometry of paths, places, objects and combinations thereof needs to be pre-featurized by the engineer to make learning possible. The model relies on dependency parses from an off-the-shelf parser, which is not always appropriate for the given domain (Kollar et al. (2017) report that "down" in "down the hall" is incorrectly labeled as "adverb" by an off-the-shelf parser). Lastly, detailed supervision that assigns groundings to constituents of training instructions is required to train G^3 (although this was addressed in (Tellex et al., 2014)). Another line of work considers parsing the instruction into a symbolic task specification (Chen and Mooney, 2011; Artzi and Zettlemoyer, 2013; Gopalan et al., 2018). A limitation

of these approaches is that the vocabulary of symbols as well as the grounded meaning of symbols are assumed to be known, and no learning procedure for adding new symbols is offered. Less restrictive in this regard is the approach by Andreas and Klein (2015) that models the alignment between the instruction and the plan.

All the above approaches are based on planning, that is translating the instructions into sequences of unambiguous primitive operations to be executed. This can be problematic when instructions refer to unknown parts of environments that are not mapped, although in principle the plan can include exploration behavior (Matuszek et al., 2013). Another approach is to avoid planning and instead view behaviour as a sequential decision-making process, in the spirit of MDP and POMDP formalisms discussed in Section 2.1.2. Duvallet et al. (2013) use this approach to train a policy to follow instructions in a partially-observable environment using the DAGGER algorithm for imitation learning.

In addition to studying instruction-following, researchers presented systems that can answer questions or understand references to objects in images (Feldman et al., 1996; Matuszek et al., 2012; Krishnamurthy and Kollar, 2013; Malinowski and Fritz, 2014). We will refer to such systems as performing visual question answering (VQA) to be consistent with the current deep learning terminology. A typical before-deep-learning VQA system comprises a perception system and a semantic parser. The former recognizes and labels entities and relations between them. The latter converts the question into a logical form by using either Combinatory Categorical Grammar (Steedman, 1996; Zettlemoyer and Collins, 2005) or Dependency-Compositional Semantics (Liang et al., 2013) as the probabilistic parsing framework. Perception and parsing components are sometimes trained together. For example, in (Krishnamurthy and Kollar, 2013) the meaning of one- and two- place predicates from the CCG lexicon is learnt jointly with the rule probabilities. Such a joint learning, however, has a limited influence, because all the aforementioned systems start from a pre-detected set of entities, and the entity detection (i.e. object recognition or image segmentation) system does not participate in joint optimization.

2.3.2. Deep Learning Approaches

It is relatively easy to build a deep learning system that produces an output based on perceptual and linguistic inputs: it is sufficient to combine neural networks that convert both inputs into vector (or tensor) representations as well as a network that fuses the said representations and produces an output based on them. Such an approach is attractive for its simplicity and also for the fact that visual processing is trained together with language processing, and hence can construct representations that are appropriate for the performed task. There is no problem, for example, with entities likes piles of dust that might not be detected by the off-the-shelf object detector: the system as a whole should be capable to adapt to instructions or questions that mention dust piles. This flexibility comes at a price

of needing more data to train and lacking a clear understanding of how the resulting opaque system generalizes.

2.3.2.1. *Visual Question Answering*

A wide variety of deep-learning-based models that can understand grounded language have been presented in the context of VQA research. The VQA setup gained popularity when large-scale datasets of crowdsourced questions about natural images were collected (e.g. the VQA 1.0 dataset by Antol et al. (2015)) and when it started being viewed as a visual alternative to the Turing test (Geman et al., 2015). The most basic deep learning model for VQA fuses an image representation produced by a CNN and a question representation produced by an LSTM (Antol et al., 2015). More complicated models perform visual attention over the image or over bounding boxes produced by an object detector (Yang et al., 2016; Anderson et al., 2018a). Another notable family of models are Neural Module Networks (NMN) (Andreas et al., 2016). The NMN approach draws inspiration from logical forms that were used to express the question meanings in earlier work (Krishnamurthy and Kollar, 2013; Malinowski and Fritz, 2014). Similarly to how the meaning is contructed from atoms (e.g. one- and two- place predicates) in logical forms, a unique network is constructed for every question from reusable modules in the NMN approach.

A common concern regarding the VQA research discussed above is that results that are obtained on crowd-sourced open-ended datasets are difficult to interpret. The original VQA 1.0 dataset (Antol et al., 2015) was collected by asking crowdsourcing workers to pose questions that would "stump the smart robot". The workers were free to ask whatever questions they want, and as a result some of the questions are very short and specific ("What color is the hydrant?") and others require advanced commonsense reasoning ("Does this man have children?", "Is this pizza vegetarian?", etc). Another side effect of the unconstrained open-ended data collection is that workers would ask similar questions in similar circumstances. The resulting dataset regularities (often called "biases") allow models to answer questions using strategies that are clearly different from the way humans understand language. Answers to the questions in many cases could be predicted from questions only in the first version of the VQA dataset, such as e.g the answer to the question "What covers the ground?" was always "snow" (Agrawal et al., 2016). In the subsequent VQA 2.0 dataset (Goyal et al., 2016) this issue was alleviated by making sure that for each question there is at least two different images with different answers. Further attempts to make more challenging splits were made. In the Compositional VQA dataset (Agrawal et al., 2017) the training and test sets are forced to not have similar question-answer pairs, and in VQA under Changing Priors dataset (Agrawal et al., 2018) the distribution of answers at test time is different from the training one. The aforementioned efforts seek to counteract the effect of one kind of bias that is discussed above: the possibility to predict the answer without looking at the image

at all. The impact of other regularities that open-ended data collection could entail (e.g. people will probably ask more questions about salient objects) has not been discussed yet, and thus we believe that the challenge of interpreting the open-ended VQA results remains open.

In order to understand better what deep learning approaches to VQA are capable of, researchers constructed synthetic datasets with automatically generated images and templated languages (Johnson et al., 2016; Kuhnle and Copestake, 2017). The CLEVR dataset (Johnson et al., 2016), that is arguably is the most well-known dataset of this kind, features 700K complex questions that require multiple steps of reasoning, counting and logical operations. CLEVR was initially found to be challenging for basic deep learning models, such as e.g. those combining CNN- and LSTM- produced image and question representations in a simple way. More complex models with much better performance were quickly proposed, including Relation Networks (RN) (Santoro et al., 2017), Feature-wise Independent Linear Modulation (FiLM) (Perez et al., 2017) and Compositional Attention Networks (CAN) (Hudson and Manning, 2018). RN treat the 3D-tensor representation of the images as a set of object representations and consider all pair-wise interactions between them. The FiLM approach views question-answering with deep learning models as modulation of visual processing in lower levels of the CNN. The CAN model performs a sequence of visual attentions over the image. A number of NMN-style approaches have also been proposed (Hu et al., 2017, 2018; Johnson et al., 2016; Mascharka et al., 2018). Lastly, several recent papers feature two-stage models that combine object detection and logical reasoning (Yi et al., 2018; Mao et al., 2018) in a way that is reminiscent to the older VQA models that were reviewed above. The model by Mao et al. (2018) in particular, is remarkably similar to the older work by Krishnamurthy and Kollar (2013). The semantic parsers are learned jointly with predicate classifiers in both papers, but Mao et al. (2018) uses a neural parser and neural classifiers instead of linear ones.

Similarly to the open-ended VQA, interpreting results of synthetic data VQA research is non-trivial. The known composition of datasets makes it possible for researchers to construct specialized models that work well on a particular dataset, and often such models are not evaluated on any other data (in particular, this is very often the case for CLEVR). Furthermore, while the diversity and complexity of the datasets can be quite high, it is still limited (e.g. CLEVR has 90 question families for 700K questions), and it is somewhat unsurprising that flexible deep learning models can handle these templates provided enough data. A number of papers are now reporting how model's performance varies with the amount of training data, and sometimes these additional evaluations are quite insightful. For example, while the differences between 97.6% and 98.9% accuracies of FiLM and MAC respectively after training on CLEVR can be hard to interpret, $> 90\%$ and $< 60\%$ performances with one

fifth of the data suggest a significant difference in ease with which the said models learn to perform the task (Hudson and Manning, 2018).

Perhaps most importantly, the lack of real-world and data collection regularities that plague the open-ended VQA research can be seen as both a blessing and a curse for synthetic data VQA research. In the real world, unlike e.g. CLEVR, an object is not equally likely to be of any color (e.g. the fire hydrants tend to be red and car tires tend to be black). Likewise, objects of certain classes (e.g. a monitor and a keyboard) have a higher chance to be referred to within one utterance than others (e.g. a car and a violin). A good model for grounded language understanding should be robust to such biases in order to deal successfully with long tail situations. Such robustness is not studied by default in synthetic data benchmarks, increasing the uncertainty regarding the significance of findings obtained on them.

Researchers have been attempting to construct synthetic setups that incorporate a controlled bias in the training distribution, thereby measuring out-of-distribution generalization. The original CLEVR paper by Johnson et al. (2016) features a dataset version called Compositional Generalization Test (CoGenT[3]) in which objects have different colors at training and test times. CoGenT measures models' robustness to regularities in the image distribution and does not explicitly control the difference between training and test questions. In contrast, in the article presented in Chapter 8 we focus on the models' ability to generalize from a biased question distribution while making sure that training and test image distributions are similar. In doing so, we aimed to adapt the notion of systematic (also often called compositional, algebraic or combinatorial generalization) generalization (Marcus, 2003; Lake and Baroni, 2018) to studies of grounded language understanding. The concept of systematicity corresponds to a human's ability to consider and interpret arbitrary combinations of known atoms of meaning (Fodor and Pylyshyn, 1988). It is closely related to the principle of compositionality that states that humans construct meanings of linguistic constituents from the meanings of their parts (see e.g. the textbook by Heim and Kratzer (1998)). Both concepts are hard to define in the most general case, but compelling instantiations can be constructed in specific cases, such as e.g. spatial reasoning about pairs of objects. We refer the reader to Chapter 8 for more details.

In addition to lacking the real world regularities, synthetic VQA datasets can exhibit biases of their own. The very fact that the language is templated in CLEVR and the same templates are used during training and during testing can be considered a bias. A uniform coverage of all possible question structures will arguably be very difficult to achieve when performing natural data collection, yet we would desire the system to understand more rare

[3]We find the use of the word "compositional" in the name CoGenT rather misleading. The test does not focus on whether meanings of words are composed to produce meanings phrases, instead it tests the robustness of individual word grounding to biases in the image distribution.

constructs as well. In Chapter 10 we discuss this issue in more detail and present new templates to evaluate systematic generalization of models that were trained on CLEVR.

To conclude the discussion of recent VQA efforts, we note that several datasets were recently presented that combine natural and synthetic data. This includes the GQA dataset (Hudson and Manning, 2019), which features synthetic questions about real world images, and the NLVR dataset (Suhr et al., 2017), which on the contrary features crowdsourced natural language assertions about synthetic images. Combined with accurate generalization testing of the kind that we perform in Chapters 8 and 10, this hybrid paradigm could be fruitful and facilitate development of models that can robustly understand more kinds of grounded language.

2.3.2.2. *Instruction Following*

A agent that can learn to follow instructions can be constructed using deep learning in a way that is similar to how a basic VQA model is built. The simplest approach is to view behaviour as sequential decision-making and train a deep learning policy to output the next action given a history of the observations and the instruction. Such an approach sidesteps intermediate procedures that are typical for most robotic systems (such as transforming the environment into a symbolic semantic map and planning in such a map) and is thus, in principle, free of the limitations that these procedures bring along (such as "dust piles" discussed earlier in this thesis and such as challenges of planning in unknown environments). In fact, the advantages of this approach were recognized before the deep learning age (Duvallet et al., 2013), yet with deep learning, sequential decision-making paradigm becomes especially appealing, as feature engineering is no longer required. The price for the flexibility is the same as for VQA models: lower data efficiency and transparency.

The studies of grounded instruction-following typically take place in abstract simulated environments. Researchers have demonstrated the deep learning policies can be trained to follow instructions using reinforcement learning (RL) (Hermann et al., 2017; Chaplot et al., 2018; Yu et al., 2018) imitation learning (IL) (Mei et al., 2016) and methods that combine aspects of both RL and IL (Janner et al., 2017). A problematic aspect of the works that rely on pure RL is that they rely on a given instruction-conditioned reward function in order to train the agent. Given how data-inefficient the current deep RL methods are, pure RL approaches to training instruction-following agents are effectively limited to synthetic languages with known unambiguous semantics. Indeed, this is the case because using RL with natural language instructions would require a human in the loop to constantly supervise and reward the agent during its lengthy training process.

Arguably, aspects of imitation learning (Pomerleau, 1991; Ng and Russell, 2000), whereby the desired behaviour is illustrated by expert demonstrations, would be required in order to make real world applications of deep learning methods for instruction-following possible.

The expert demonstrations can reduce or eliminate the need for trial-and-error learning of what the instructions require. Notably, almost all the approaches for teaching robots to follow instructions that we reviewed in Section 2.3.1 are relying on expert demonstrations. Most imitation learning methods require complete trajectories (i.e. action sequences) as demonstrations. This can be restrictive because to demonstrate a behaviour to the agent the human needs to control it remotely, which can be difficult. In Chapter 4.2 we explore how reward functions can be induced from examples of instructions and the corresponding goal states only, without using complete trajectories.

While instruction-following research that relies on RL often uses synthetic data, imitation-learning studies tend to use instructions and demonstrations collected from humans (Anderson et al., 2018b; Mei et al., 2016; Misra et al., 2017; Fried et al., 2018). The success rates obtained by agents trained in these studies tend to be around 60-70%, perhaps due to the limited size of the demonstrations datasets (from $\approx 1K$ to $\approx 20K$ demonstrations). It is unclear whether systems that perform so unreliably can be useful. It is therefore interesting to study how many demonstrations, or more generally, how much supervision is at the moment required for deep learning models to learn to follow instructions robustly. Chapter 6 presents the BabyAI platform that we developed to facilitate studies of this topic.

Chapter 3

PROLOGUE TO FIRST ARTICLE

3.1. Article Details

Learning to Understand Goal Specifications by Modelling Reward. Dzmitry Bahdanau, Felix Hill, Jan Leike, Edward Hughes, Arian Hosseini, Pushmeet Kohli, and Edward Grefenstette. International Conference on Learning Representations 2019. Presented is a revised version of the article, see Section 4.7 for details.

Personal Contribution. The article is the outcome of my Fall 2017 internship at DeepMind, London. The idea to study instruction-following from examples of instructions and goal states was born in discussions with Edward Grefenstette, Pushmeet Kohli and Jan Leike. I developed and implemented the initial algorithm as well as the GridLU-Relations environment for testing it. Felix Hill contributed the GridLU-Arrangements environment and experiments on it. Edward Grefenstette implemented the false negatives rejection heuristic based on my idea. Arian Hosseini helped me to debug the PPO-based reimplementation of the algorithm. Edward Hughes's help with DeepMind's infrastructure as well as with running baseline experiments was extremely valuable. I wrote the first version of the paper, which was later edited by all project participants, especially by Edward Grefenstette.

3.2. Context

The project was inspired by a series of papers that showed how deep RL agents can be trained to follow synthetic instructions. All such methods relied on a programmatic implementation of instruction-conditioned reward function, which assumption we found very restrictive. It was natural for us to inquire whether such a reward function can be learned from a relatively cheap data source, such as goal-state demonstrations paired with the corresponding instructions.

3.3. Contributions

The paper introduces and experimentally proves the concept of learning instruction-conditioned reward functions from examples of instructions and corresponding goal states. We show that the learned reward functions can be reused to adapt to a change in the dynamics of the environment. The paper is the first to report the difficulties of achieving near-perfect performance with reward modelling methods and deep learning models, as well as to propose a remedy for this issue: the false negatives rejection heuristic.

Chapter 4

LEARNING TO UNDERSTAND GOAL SPECIFICATIONS BY MODELLING REWARD

4.1. Introduction

Developing agents that can learn to follow user instructions pertaining to an environment is a long-standing goal of AI research (Winograd, 1972). Recent work has shown deep reinforcement learning (RL) to be a promising paradigm for learning to follow language-like instructions in both 2D and 3D worlds (e.g. Hermann et al. (2017); Chaplot et al. (2018), see Section 4.4 for a review). In each of these cases, being able to reward an agent for successfully completing a task specified by an instruction requires the implementation of a full interpreter of the instruction language. This interpreter must be able to evaluate the instruction against environment states to determine when reward must be granted to the agent, and in doing so requires full knowledge (on the part of the designer) of the semantics of the instruction language relative to the environment. Consider, for example, 4 arrangements of blocks presented in Figure 4.1. Each of them can be interpreted as a result of successfully executing the instruction "build an L-like shape from red blocks", despite the fact that these arrangements differ in the location and the orientation of the target shape, as well as in the positioning of the irrelevant blue blocks. At best (e.g. for instructions such as the aforementioned one), implementing such an interpreter is feasible, although typically onerous in terms of engineering efforts to ensure reward can be given—for any admissible instruction in the language—in potentially complex or large environments. At worst, if we wish to scale to the full complexity of natural language, with all its ambiguity and underspecification, this requires solving fundamental problems of natural language understanding.

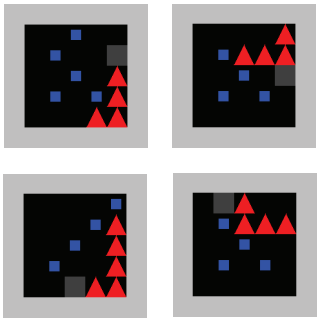


Figure 4.1. Different valid goal states for the instruction "build an L-like shape from red blocks".

If instruction-conditional reward functions cannot conveniently or tractably be implemented, can we somehow learn them in order to then train instruction-conditional policies? When there is a single implicit task, Inverse Reinforcement Learning (IRL; Ng and Russell, 2000; Ziebart et al., 2008) methods in general, and Generative Adversarial Imitation Learning (Ho and Ermon, 2016) in particular, have yielded some success in jointly learning reward functions from expert data and training policies from learned reward models. In this paper, we wish to investigate whether such mechanisms can be adapted to the more general case of jointly learning to understand language which specifies task objectives (e.g. instructions, goal specifications, directives), and use such understanding to reward language-conditional policies which are trained to complete such tasks. For simplicity, we explore a facet of this general problem in this paper by focussing on the case of declarative commands that specify sets of possible goal-states (e.g. "arrange the red blocks in a circle."), and where expert examples need only be goal states rather than full trajectories or demonstrations, leaving such extensions for further work. We introduce a framework—Adversarial Goal-Induced Learning from Examples (AGILE)—for jointly training an instruction-conditional reward model using expert examples of completed instructions alongside a policy which will learn to complete instructions by maximising the thus-modelled reward. In this respect, AGILE relies on familiar RL objectives, with free choice of model architecture or training mechanisms, the only difference being that the reward comes from a learned reward model rather than from the environment.

We first verify that our method works in settings where a comparison between AGILE-trained policies with policies trained from environment reward is possible, to which end we implement instruction-conditional reward functions. In this setting, we show that the learning speed and performance of A3C agents trained with AGILE reward models is superior to A3C agents trained against environment reward, and comparable to that of true-reward A3C agents supplemented by auxiliary unsupervised reward prediction objectives. To simulate an instruction-learning setting in which implementing a reward function would be problematic, we construct a dataset of instructions and goal-states for the task of building colored orientation-invariant arrangements of blocks. On this task, without us ever having to implement the reward function, the agent trained within AGILE learns to construct arrangements as instructed. Finally, we study how well AGILE's reward model generalises beyond the examples on which it was trained. Our experiments show it can be reused to allow the policy to adapt to changes in the environment.

4.2. Adversarial Goal-Induced Learning from Examples

Here, we introduce AGILE ("Adversarial Goal-Induced Learning from Examples", in homage to the adversarial learning mechanisms that inspire it), a framework for jointly learning to model reward for instructions, and learn a policy from such a reward model.

Specifically, we learn an instruction-conditional *policy* π_θ with parameters θ, from a data stream $\mathcal{G}^{\pi_\theta}$ obtained from interaction with the environment, by adjusting θ to maximise the expected total reward $R_\pi(\theta)$ based on stepwise reward $\hat{r}_t$ given to the policy, exactly as done in any normal Reinforcement Learning setup. The difference lies in the source of the reward: we introduce an additional discriminator network D_ϕ, the *reward model*, whose purpose is to define a meaningful reward function for training π_θ. We jointly learn this reward model alongside the policy by training it to predict whether a given state s is a goal state for a given instruction c or not. Rather than obtain positive and negative examples of ⟨instruction, state⟩ pairs from a purely static dataset, we sample them from a policy-dependent data stream. This stream is defined as follows: positive examples are drawn from a fixed dataset $\mathcal{D}$ of instructions c_i paired with goal states s_i; negative examples are drawn from a constantly-changing buffer of states obtained from the policy acting on the environment, paired with the instruction given to the policy. Formally, the policy is trained to maximize a return $R_\pi(\theta)$ and the reward model is trained to minimize a cross-entropy loss $L_D(\phi)$, the equations for which are:

$$R_\pi(\theta) = \mathop{\mathbb{E}}_{(c,s_{1:\infty})\sim\mathcal{G}^{\pi_\theta}} \sum_{t=1}^{\infty} \gamma^{t-1}\hat{r}_t + \alpha H(\pi_\theta), \tag{4.2.1}$$

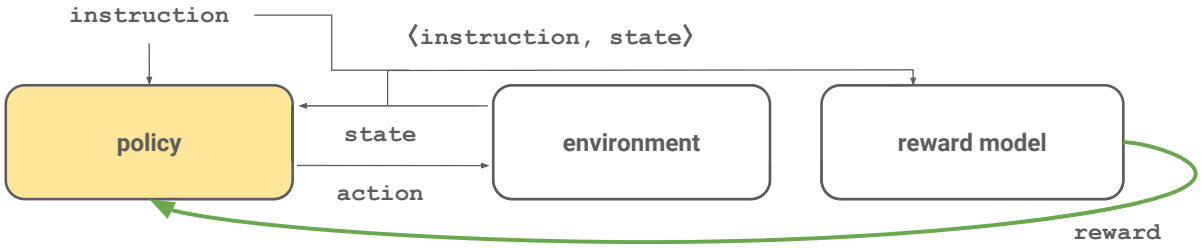
$$L_D(\phi) = \mathop{\mathbb{E}}_{(c,s)\sim\mathcal{B}} -\log(1 - D_\phi(c,s)) + \mathop{\mathbb{E}}_{(c_i,g_i)\sim\mathcal{D}} -\log D_\phi(c_i,g_i). \tag{4.2.2}$$

where

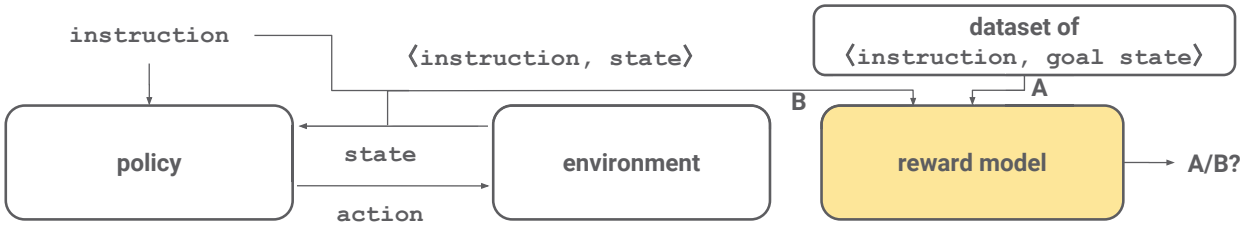
$$\hat{r}_t = [D_\phi(c,s_t) > 0.5]$$

In the equations above, the Iverson Bracket $[\ldots]$ maps truth to 1 and falsehood to 0, e.g. $[x > 0] = 1$ iff $x > 0$ and 0 otherwise. γ is the discount factor. With $(c, s_{1:\infty}) \sim \mathcal{G}^{\pi_\theta}$, we denote a state trajectory that was obtained by sampling $(c, s_0) \sim \mathcal{G}$ and running π_θ conditioned on c starting from s_0. $\mathcal{B}$ denotes a replay buffer to which (c, s) pairs from T-step episodes are added; i.e. it is the undiscounted occupancy measure over the first T steps. $D_\phi(c, s)$ is the probability of (c, s) having a positive label according to the reward model, and thus $[D_\phi(c, s_t) > 0.5]$[1] indicates that a given state s_t is more likely to be a goal state for instruction c than not, according to D. $H(\pi_\theta)$ is the expected per-step policy entropy, and α is a hyperparameter. The approach is illustrated in Fig 4.2. Pseudocode is available in Section 4.6. We note that Equation 4.2.1 differs from a traditional RL objective only in that the modelled reward $\hat{r}_t$ is used instead of the ground-truth reward r_t. Indeed, in Section 4.3, we will compare policies trained with AGILE to policies trained with traditional RL, simply by varying the reward source from the reward model to the environment.

[1]While all results reported in this paper are obtained using a discretized reward $r_t = [D_\phi(c, s_t) > 0.5]$, in additional experiments we found that using just $D_\phi(c, s_t)$ as the reward is equally effective. See Section 4.7 for more context on the choice of the reward function.



(A) Policy Training



(B) Reward Model Training

FIGURE 4.2. Information flow during AGILE training. The policy acts conditioned on the instruction and is trained using the reward from the reward model (Figure 4.2a). The reward model is trained, as a discriminator, to distinguish between "A", the ⟨instruction, goal-state⟩ pairs from the dataset (Figure 4.2b), and "B", the ⟨instruction, state⟩ pairs from the agent's experience.

4.2.1. Dealing with False Negatives

Let us call $\Gamma(c)$ the objective set of goal states which satisfy instruction c (which is typically unknown to us). Compared to the ideal case where all (c, s) would be deemed positive if-and-only-if $s \in \Gamma(c)$, the labelling of examples implied by Equation 4.2.2 has a fundamental limitation when the policy performs well. As the policy improves, by definition, a increasing share of $(c, s) \in \mathcal{B}$ are objective goal-states from $\Gamma(c)$. However, as they are treated as negative examples in Equation 4.2.2, the discriminator accuracy drops, causing the policy to get worse. We therefore propose the following simple heuristic to rectify this fundamental limitation by approximately identifying the false negatives. We rank (c, s) examples in $\mathcal{B}$ according to the reward model's output $D_\phi(c, s)$ and discard the top $1 - \rho$ percent as potential false negatives. Only the other ρ percent are used as negative examples of the reward model. Formally speaking, the first term in Equation 4.2.2 becomes $\mathbb{E}_{(c,s)\sim\mathcal{B}_{D_\phi,\rho}} - \log(1 - D_\phi(c, s))$, where $\mathcal{B}_{D_\phi,\rho}$ stands for the ρ percent of $\mathcal{B}$ selected, using D_ϕ, as described above. We will henceforth refer to ρ as the anticipated negative rate. Setting ρ to 100% means using $\mathcal{B}_{D_\phi,100} = \mathcal{B}$ like in Equation 4.2.2, but our preliminary experiments have shown clearly that this inhibits the reward model's capability to correctly learn a reward function. Using too small a value for ρ on the other hand may deprive the reward model of

the most informative negative examples. We thus recommend to tune ρ as a hyperparameter on a task-specific basis.

4.2.2. Reusability of the Reward Model

An appealing advantage of AGILE is the fact that the reward model D_ϕ and the policy π_θ learn two related but distinct aspects of an instruction: the reward model focuses on recognizing the goal-states (what should be done), whereas the policy learns what to do in order to get to a goal-state (how it should be done). The intuition motivating this design is that the knowledge about how instructions define goals should generalize more strongly than the knowledge about which behavior is needed to execute instructions. Following this intuition, we propose to reuse a reward model trained in AGILE as a reward function for training or fine-tuning policies.

4.2.3. Relation to GAIL

AGILE is strongly inspired by—and retains close relations to—Generative Adversarial Imitation Learning (GAIL; Ho and Ermon, 2016), which likewise trains both a reward function and a policy. The former is trained to distinguish between the expert's and the policy's trajectories, while the latter is trained to maximize the modelled reward. GAIL differs from AGILE in a number of important respects. First, AGILE is conditioned on instructions c so a single AGILE agent can learn combinatorially many skills rather than just one. Second, in AGILE the reward model observes only states s_i (either goal states from an expert, or states from the agent acting on the environment) rather than state-action traces $(s_1, a_1), (s_2, a_2), \ldots$, learning to reward the agent based on "what" needs to be done rather than according to "how" it must be done. Finally, in AGILE the policy's reward is the thresholded probability $[D_\phi(c, s_t)]$ as opposed to $-\log(1 - D_\phi(s_t, a_t))$ used in GAIL (see Section 4.7 for additional context on this choice).

4.3. Experiments

We experiment with AGILE in a grid world environment that we call GridLU, short for Grid Language Understanding and after the famous SHRDLU world (Winograd, 1972). GridLU is a fully observable grid world in which the agent can walk around the grid (moving up, down left or right), pick blocks up and drop them at new locations (see Figure 4.3 for an illustration and Appendix A.2 for a detailed description of the environment).

4.3.1. Models

All our models receive the world state as a 56x56 RGB image. With regard to processing the instruction, we will experiment with two kinds of models: Neural Module Networks

(NMN) that treat the instruction as a structured expression, and a generic model that takes an unstructured instruction representation and encodes it with an LSTM.

Because the language of our instructions is generated from a simple grammar, we perform most of our experiments using policy and reward model networks that are constructed using the NMN (Andreas et al., 2016) paradigm. NMN is an elegant architecture for grounded language processing in which a tree of neural modules is constructed based on the language input. The visual input is then fed to the leaf modules, which send their outputs to their parent modules, which process is repeated until the root of the tree. We mimick the structure of the instructions when constructing the tree of modules; for example, the NMN corresponding to the instruction c_1=*NorthFrom(Color('red', Shape('circle', SCENE)), Color('blue', Shape('square', SCENE)))* performs a computation $h_{NMN} = m_{NorthFrom}(m_{red}(m_{circle}(h_s)), m_{blue}(m_{square}(h_s))))$, where m_x denotes the module corresponding to the token x, and h_s is a representation of state s. Each module m_x performs a convolution (weights shared by all modules) followed by a token-specific Feature-Wise Linear Modulation (FiLM) (Perez et al., 2017): $m_x(h_l, h_r) = ReLU((1+\gamma_x)\odot(W_m * [h_l; h_r])\oplus\beta_x)$, where h_l and h_r are module inputs, γ_x is a vector of FiLM multipliers, β_x are FiLM biases, $\odot$ and $\oplus$ are element-wise multiplication and addition with broadcasting, $*$ denotes convolution. The representation h_s is produced by a convnet. The NMN's output h_{NMN} undergoes max-pooling and is fed through a 1-layer MLP to produce action probabilities or the reward model's output. Note, that while structure-wise our policy and reward model are mostly similar, they do not share parameters.

NMN is an excellent model when the language structure is known, but this may not be the case for natural language. To showcase AGILE's generality we also experiment with a very basic structure-agnostic architecture. We use FiLM to condition a standard convnet on an instruction representation h_{LSTM} produced by an LSTM. The k-th layer of the convnet performs a computation $h_k = ReLU((1+\gamma_k)\odot(W_k * h_{k-1})\oplus\beta_k)$, where $\gamma_k = W_k^\gamma h_{LSTM} + b_k^\gamma$, $\beta_k = W_k^\beta h_{LSTM} + b_k^\beta$. The same procedure as described above for h_{NMN} is used to produce the network outputs using the output h_5 of the 5^{th} layer of the convnet.

In the rest of the paper we will refer to the architectures described above as FiLM-NMN and FiLM-LSTM respectively. FiLM-NMN will be the default model in all experiments unless explicitly specified otherwise. Detailed information about network architectures can be found in Appendix A.6.

4.3.2. Training Details

For the purpose of training the policy networks both within AGILE, and for our baseline trained from ground-truth reward r_t instead of the modelled reward $\hat{r}_t$, we used the Asynchronous Advantage Actor-Critic (A3C; Mnih et al., 2016). Any alternative training mechanism which uses reward could be used—since the only difference in AGILE is the source

of the reward signal, and for any such alternative the appropriate baseline for fair comparison would be that same algorithm applied to train a policy from ground-truth reward. We will refer to the policy trained within AGILE as AGILE-A3C. The A3C's hyperparameters γ and λ were set to 0.99 and 0 respectively, i.e. we did not use without temporal difference learning for the baseline network. The length of an episode was $T = 30$, but we trained the agent on advantage estimation rollouts of length 15. Every experiment was repeated 5 times. We considered an episode to be a success if the final state was a goal state as judged by a task-specific *success criterion*, which we describe for the individual tasks below. We use the success rate (i.e. the percentage of successful episodes) as our main performance metric for the agents. Unless otherwise specified we use the NMN-based policy and reward model in our experiments. Full experimental details can be found in Appendices A.1 and A.3.

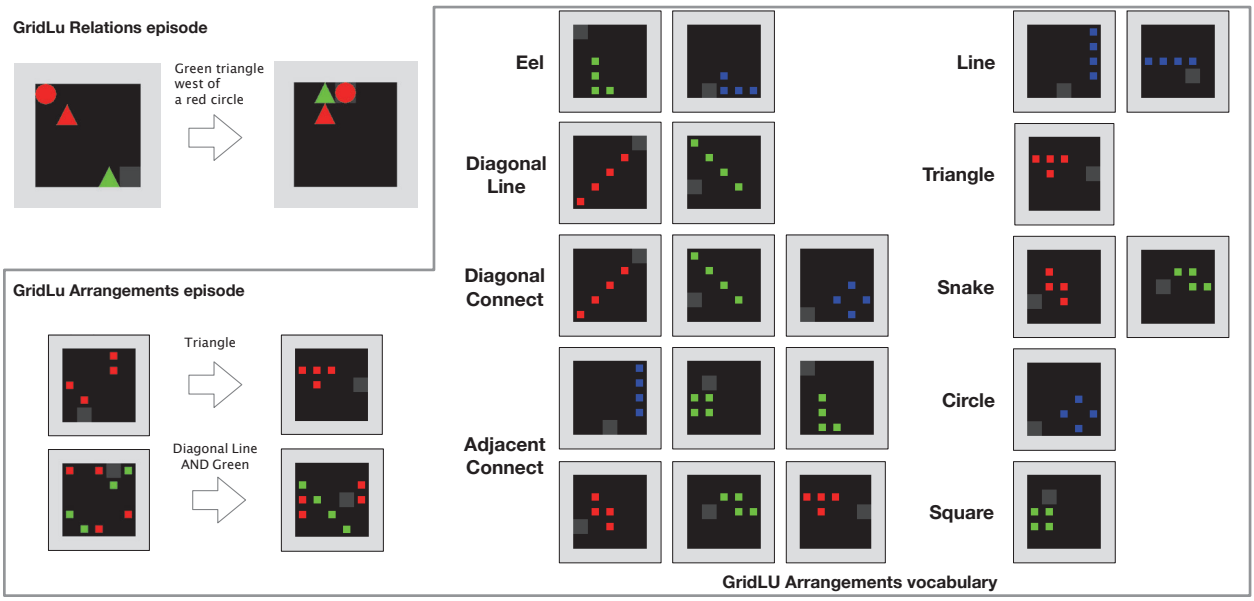


FIGURE 4.3. Initial state and goal state for GridLU-Relations (top-left) and GridLU-Arrangements episodes (bottom-left), and the complete GridLU-Arrangements vocabulary (right), each with examples of some possible goal-states.

4.3.3. GridLU-Relations

Our first task, GridLU-Relations, is an adaptation of the SHAPES visual question answering dataset (Andreas et al., 2016) in which the blocks can be moved around freely. GridLU-Relations requires the agent to induce the meaning of spatial relations such as *above* or *right of*, and to manipulate the world in order to instantiate these relationships. Named GridLU-Relations, the task involves five spatial relationships (*NorthFrom*, *SouthFrom*, *EastFrom*, *WestFrom*, *SameLocation*), whose arguments can be either the blocks, which are referred to by their shapes and colors, or the agent itself. To generate the full set of possible

instructions spanned by these relations and our grid objects, we define a formal grammar that generates strings such as:

NorthFrom(Color('red', Shape('circle', SCENE)), Color('blue', Shape('square', SCENE))) (4.3.1)

This string carries the meaning 'put a red circle north from (above) a blue square'. In general, when a block is the argument to a relation, it can be referred to by specifying both the shape and the color, like in the example above, or by specifying just one of these attributes. In addition, the *AGENT* constant can be an argument to all relations, in which case the agent itself must move into a particular spatial relation with an object. Figure 4.3 shows two examples of GridLU-Relations instructions and their respective goal states. There are 990 possible instructions in the GridLU-Relations task, and the number of distinct training instances can be loosely lower-bounded by $1.8 \cdot 10^7$ (see Appendix A.4 for details).

Notice that, even for the highly concrete spatial relationships in the GridLU-Relations language, the instructions are underspecified and somewhat ambiguous—is a block in the top-right corner of the grid *above* a block in the bottom left corner? We therefore decided (arbitrarily) to consider all relations to refer to immediate adjacency (so that Instruction (4.3.1) is satisfied if and only if there is a red circle in the location *immediately* above a blue square). Notice that the commands are still underspecified in this case (since they refer to the relationship between two entities, not their absolute positions), even if the degree of ambiguity in their meaning is less than in many real-world cases. The policy and reward model trained within AGILE then have to infer this specific sense of what these spatial relations mean from goal-state examples, while the baseline agent is allowed to access our programmed ground-truth reward. The binary ground-truth reward (true if the state is a goal state) is also used as the success criterion for evaluating AGILE.

Having formally defined the semantics of the relationships and programmed a reward function, we compared the performance of an AGILE-A3C agent against a priviliged baseline A3C agent trained using ground-truth reward. Interestingly, we found that AGILE-A3C learned the task more easily than standard A3C (see the respective curves in Figure 4.4). We hypothesize this is because the modeled rewards are easy to learn at first and become more sparse as the reward model slowly improves. This naturally emerging curriculum expedites learning in the AGILE-A3C when compared to the A3C-trained policy that only receives signal upon reaching a perfect goal state.

We did observe, however, that the A3C algorithm could be improved significantly by applying the auxiliary task of reward prediction (RP; Jaderberg et al., 2016), which was applied to language learning tasks by Hermann et al. (2017) (see the A3C and A3C-RP curves in Figure 4.4). This objective reinforces the association between instructions and states by having the agent replay the states immediately prior to a non-zero reward and

predict whether or not it the reward was positive (i.e. the states match the instruction) or not. This mechanism made a significant difference to the A3C performance, increasing performance to 99.9%. AGILE-A3C also achieved nearly perfect performance (99.5%). We found this to be a very promising result, since within AGILE, we induce the reward function from a limited set of examples.

The best results with AGILE-A3C were obtained using the anticipated negative rate $\rho = 25\%$. When we used larger values of ρ AGILE-A3C training started quicker but after 100-200 million steps the performance started to deteriorate (see AGILE curves in Figure 4.4), while it remained stable with $\rho = 25\%$.

Data efficiency

These results suggest that the AGILE reward model was able to induce a near perfect reward function from a limited set of ⟨instruction, goal-state⟩ pairs. We therefore explored how small this training set of examples could be to achieve reasonable performance. We found that with a training set of only 8000 examples, the AGILE-A3C agent could reach a performance of 60% (massively above chance). However, the optimal performance was achieved with more than 100,000 examples. The full results are available in Appendix A.3.

Generalization to Unseen Instructions

In the experiments we have reported so far the AGILE agent was trained on all 990 possible GridLU-Relation instructions. In order to test generalization to unseen instructions we held out 10% of the instructions as the test set and used the remaining 90% as the training set. Specifically, we restricted the training instances and ⟨instruction, goal-state⟩ pairs to only contain instructions from the training set. The performance of the trained model on the test instructions was the same as on the training set, showing that AGILE did not just memorise the training instructions but learnt a general interpretation of GridLU-Relations instructions.

AGILE with Structure-Agnostic Models

We report the results for AGILE with a structure-agnostic FILM-LSTM model in Figure 4.4 (middle). AGILE with $\rho = 25\%$ achieves a high 97.5% success rate, and notably it trains almost as fast as an RL-RP agent with the same architecture.

Analyzing the reward model

We compare the binary reward provided by the reward model with the ground-truth from the environment during training on the GridLU-Relation task. With $\rho = 25\%$ the accuracy of the reward model peaks at 99.5%. As shown in Figure 4.4 (right) the reward model learns faster in the beginning with larger values of ρ but then deteriorates, which confirms our

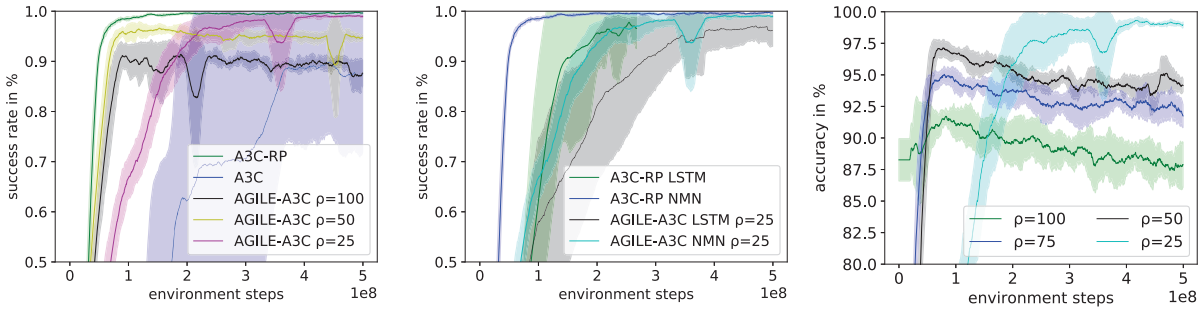


FIGURE 4.4. **Left:** learning curves for A3C, A3C-RP (both using ground truth reward), and AGILE-A3C with different values of the anticipated negative rate ρ on the GridLU-Relations task. We report success rate (see Section 4.3). **Middle:** learning curves for policies trained with ground-truth RL, and within AGILE, with different model architectures. **Right:** the reward model's accuracy for different values of ρ.

intuition about why ρ is an important hyperparameter and is aligned with the success rate learning curves in Figure 4.4 (left). We also observe during training that the false negative rate is always kept reasonably low (<3% of rewards) whereas the reward model will initially be more generous with false positives (20–50% depending on ρ during the first 20M steps of training) and will produce an increasing number of false positives for insufficiently small values of ρ (see plots in Appendix A.4). We hypothesize that early false positives may facilitate the policy's training by providing it with a sort of curriculum, possibly explaining the improvement over agents trained from ground-truth reward, as shown above.

The reward model as general reward function

An instruction-following agent should be able to carry-out known instructions in a range of different contexts, not just settings that match identically the specific setting in which those skills were learned. To test whether the AGILE framework is robust to (semantically-unimportant) changes to the environment dynamics, we first trained the policy and reward model as normal and then modified the effective physics of the world by making all red square objects immovable. In this case, following instructions correctly is still possible in almost all cases, but not all solutions available during training are available at test time. As expected, this change impaired the policy and the agent's success rate on the instructions referring to a red square dropped from 98% to 52%. However, after fine-tuning the policy (additional training of the policy on the test episodes using the reward from the previously-trained-then-frozen reward model), the success rate went up to 69.3% (Figure 4.5). This experiment suggests that the AGILE reward model learns useful and generalisable linguistic knowledge. The knowledge can be applied to help policies adapt in scenarios where the high-level meaning of commands is familiar but the low-level physical dynamics is not.

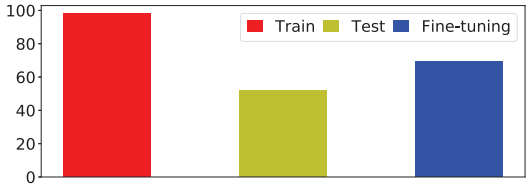


FIGURE 4.5. Fine-tuning for an immovable red square.

4.3.4. GridLU-Arrangements Task

The experiments thus far demonstrate that even without directly using the reward function AGILE-A3C performs comparably to its pure A3C counter-part. However, the principal motivation for the AGILE framework is to avoid programming the reward function. To model this setting more explicitly, we developed the task **GridLU-Arrangements**, in which each instruction is associated with multiple viable goal-states that share some (more abstract) common form. The complete set of instructions and forms is illustrated in Figure 4.3. To get training data, we built a generator to produce random instantiations (i.e. any translation, rotation, reflection or color mapping of the illustrated forms) of these goal-state classes, as positive examples for the reward model. In the real world, this process of generating goal-states could be replaced by finding, or having humans annotate, labelled images. In total, there are 36 possible instructions in GridLU-Arrangements, which together refer to a total of 390 million correct goal-states (see Appendix A.5 for details). Despite this enormous space of potentially correct goal-states, we found that for good performance it was necessary to train AGILE on only 100,000 (less than 0.3%) of these goal-states, sampled from the same distribution as observed in the episodes. To replicate the conditions of a potential AGILE application as close as possible, we did not write a reward function for GridLU-Arrangements (even though it would have been theoretically possible), and instead carried out all evaluation manually.

The training regime for GridLU-Arrangements involved two classes of episodes (and instructions). Half of the episodes began with four square blocks (all of the same color), and the agent, in random unique positions, and an instruction sampled uniformly from the list of possible arrangement words. In the other half of the episodes, four square blocks of one color and four square blocks of a different color were initially each positioned randomly. The instruction in these episodes specified one of the two colors together with an arrangement word. We trained policies and reward models using AGILE with 10 different seeds for each level, and selected the best pair based on how well the policy maximised *modelled* reward. We then manually assessed the final state of each of 200 evaluation episodes, using human judgement that the correct shape has been produced as success criterion to evaluate AGILE. We found that the agent made the correct arrangement in 58% of the episodes.

The failure cases were almost always in the episodes involving eight blocks[2]. In these cases, the AGILE agent tended towards building the correct arrangement, but was impeded by the randomly positioned non-target-color blocks and could not recover. Nonetheless, these scores, and the compelling behaviour observed in the video (https://www.youtube.com/watch?v=07S-x3MkEoQ), demonstrate the potential of AGILE for teaching agents to execute semantically vague or underspecified instructions.

4.4. Related Work

Learning to follow language instructions has been approached in many different ways, for example by reinforcement learning using a reward function programmed by a system designer. Janner et al. (2017); Oh et al. (2017); Hermann et al. (2017); Chaplot et al. (2018); Denil et al. (2017); Yu et al. (2018) consider instruction-following in 2D or 3D environments and reward the agent for arriving at the correct location or object. Janner et al. (2017) and Misra et al. (2017) train RL agents to produce goal-states given instructions. As discussed, these approaches are constrained by the difficulty of programming language-related reward functions, a task that requires a programming expert, detailed access to the state of the environment and hard choices above how language should map to the world. Agents can be trained to follow instructions using complete demonstrations, that is sequences of correct actions describing instruction execution for given initial states.

Chen and Mooney (2011); Artzi and Zettlemoyer (2013) train semantic parsers to produce a formal representation of the query that when fed to a predefined execution model matches exactly the sequence of actions from the demonstration. Andreas and Klein (2015); Mei et al. (2016) sidestep the intermediate formal representation and train a Conditional Random Field (CRF) and a sequence-to-sequence neural model respectively to directly predict the actions from the demonstrations. An underlying assumption behind all these approaches is that the agent and the demonstrator share the same actuation model, which might not always be the case. In the case of navigational instructions the trajectories of the agent and the demonstrators can sometimes be compared without relying on the actions, like e.g. Vogel and Jurafsky (2010), but for other types of instructions such a hard-coded comparison may be infeasible. Tellex et al. (2011) train a log-linear model to map instruction constituents into their groundings, which can be objects, places, state sequences, etc. Their approach requires access to a structured representation of the world environment as well as intermediate supervision for grounding the constituents.

Our work can be categorized as apprenticeship (imitation) learning, which studies learning to perform tasks from demonstrations and feedback. Many approaches to apprenticeship learning are variants of inverse reinforcement learning (IRL), which aims to recover a reward

[2]The agent succeeded on 92% (24%) with 4 (8) blocks.

function from expert demonstrations (Abbeel and Ng, 2004; Ziebart et al., 2008). As stated at the end of Section 4.2, the method most closely related to AGILE is the GAIL algorithm from the IRL family (Ho and Ermon, 2016). There have been earlier attempts to use IRL-style methods for instruction following (MacGlashan et al., 2015; Williams et al., 2018), but unlike AGILE, they relied on the availability of a formal reward specification language. To our knowledge, ours and the concurrent work by Fu et al. (2018) are the first works to showcase learning reward models for instructions from pixels directly. Besides IRL-style approaches, other apprenticeship learning methods involve training a policy (Knox and Stone, 2009; Warnell et al., 2017) or a reward function (Wilson et al., 2012; Christiano et al., 2017) directly from human feedback. Several recent imitation learning works consider using goal-states directly for defining the task (Ganin et al., 2018; Pathak et al., 2018). AGILE differs from these approaches in that goal-states are only used to train the reward module, which we show generalises to new environment configurations or instructions, relative to those seen in the expert data.

4.5. Discussion

We have proposed AGILE, a framework for training instruction-conditional RL agents using rewards from learned reward models, which are jointly trained from data provided by both experts and the agent being trained, rather than reward provided by an instruction interpreter within the environment. This opens up new possibilities for training language-aware agents: in the real world, and even in rich simulated environments (Brodeur et al., 2017; Wu et al., 2018), acquiring such data via human annotation would often be much more viable than defining and implementing reward functions programmatically. Indeed, programming rewards to teach robust and general instruction-following may ultimately be as challenging as writing a program to interpret language directly, an endeavour that is notoriously laborious, and some say, ultimately futile (Winograd, 1972).

As well as a means to learn from a potentially more prevalent form of data, our experiments demonstrate that policies trained in the AGILE framework perform comparably with and can learn as fast as those trained against ground-truth reward and additional auxiliary tasks. Our analysis of the reward model's classifications gives a sense of how this is possible; the false positive decisions that it makes early in the training help the policy to start learning. The fact that AGILE's objective attenuates learning issues due to the sparsity of reward states within episodes in a manner similar to reward prediction suggests that the reward model within AGILE learns some form of shaped reward (Ng et al., 1999), and could serve not only in the cases where a reward function need to be learned in the absence of true reward, but also in cases where environment reward is defined but sparse. As these cases are not the focus of this study, we note this here, but leave such investigation for future work.

As the policy improves, false negatives can cause the reward model accuracy to deteriorate. We determined a simple method to mitigate this, however, leading to robust training that is comparable to RL with reward prediction and unlimited access to a perfect reward function. Another attractive aspect of AGILE is that learning "what should be done" and "how it should be done" is performed by two different model components. Our experiments confirm that the "what" kind of knowledge generalizes better to new environments. When the dynamics of the environment changed at test time, fine-tuning using frozen reward model allowed to the policy recover some of its original capability in the new setting.

While there is a large gap to be closed between the sort of tasks and language that we considered in this paper and those which might be presented in "real world" situations or more complex environments, our results provide an encouraging first step in this direction. Indeed, it is interesting to consider how AGILE could be applied to more realistic learning settings, for instance involving first-person vision of 3D environments. Two issues would need to be dealt with, namely training the agent to factor out the difference in perspective between the expert data and the agent's observations, and training the agent to ignore its own body parts if they are visible in the observations. Future work could focus on applying third-person imitation learning methods recently proposed by Stadie et al. (2017) learn the aforementioned invariances. Most of our experiments were conducted with a formal language with a known structure, however AGILE also performed very well when we used a structure-agnostic FiLM-LSTM model which processed the instruction as a plain sequence of tokens. This result suggest that in future work AGILE could be used with natural language instructions.

4.6. AGILE Pseudocode

See Algorithms 1 and 2 for pseudocode descriptions of policy and discriminator training respectively.

4.7. More On Relation to GAIL

In an earlier version of the paper we argued why AGILE reward $[D_\phi(c, s_t) > 0.5]$ is preferable to a "GAIL-style reward" that we defined as $\log D_\phi(c, s_t)$. In retrospect, the proper adaptation of the GAIL policy objective $\log D(s, a)$ (see Equation 16 in (Ho and Ermon, 2016)) to our notation would be $-\log(1 - D_\phi(c, s_t))$, not $\log D_\phi(c, s_t)$. This holds because (a) in GAIL the policy objective is a cost to be minimized, whereas in AGILE it is a reward to be maximized (b) in GAIL the discriminator is trained to output higher values for the agent's state-action pairs, whereas in AGILE (and most of the literature on adversarial methods) it is trained to do the opposite. We have not tried running AGILE with $-\log(1 - D_\phi(c, s_t))$ as the reward function, but we hypothesize that it would work.

Algorithm 1 AGILE Discriminator Training

Require: The policy network π_θ, the discriminator network D_ϕ, the anticipated negative rate ρ, a dataset $\mathcal{D}$, a replay buffer B, the batch size BS, a stream of training instances $\mathcal{G}$, the episode length T, the rollout length R.

while Not Converged **do**
  Sample a training instance $(c, s_0) \in \mathcal{G}$.
  $t \leftarrow 0$
  while t < T **do**
    Act with $\pi_\theta(c, s)$ and produce a rollout $(c, s_{t...t+R})$.
    Add (c, s) pairs from $(c, s_{t...t+R})$ to the replay buffer B. Remove old pairs from B if it is overflowing.
    Sample a batch D_+ of $BS/2$ positive examples from $\mathcal{D}$.
    Sample a batch D_- of $BS/(2 \cdot (1-\rho))$ negative (c, s) examples from B.
    Compute $\kappa = D_\phi(c, s)$ for all $(c, s) \in D_-$ and reject the top $1-\rho$ percent of D_- with the highest κ. The resulting D_- will contain $BS/2$ examples.
    Compute $\tilde{L}_D(\phi) = \frac{1}{BS} \sum_{(c,s)\in D_-} -\log(1 - D_\phi(c, s)) + \sum_{(c,g)\in D_+} -\log D_\phi(c_i, g_i)$.
    Compute the gradient $\frac{d\tilde{L}_D(\phi)}{d\phi}$ and use it to update ϕ.
    Synchronise θ and ϕ with other workers.
    $t \leftarrow t + R$
  end while
end while

Algorithm 2 AGILE Policy Training

Require: The policy network π_θ, the discriminator network D_ϕ, a dataset $\mathcal{D}$, a replay buffer B, a stream of training instances $\mathcal{G}$, the episode length T.

while Not Converged **do**
  Sample a training instance $(c, s_0) \in \mathcal{G}$.
  $t \leftarrow 0$
  while t < T **do**
    Act with $\pi_\theta(c, s)$ and produce a rollout $(c, s_{t...t+R})$.
    Use the discriminator D_ϕ to compute the rewards $r_\tau = [D_\phi(c, s_\tau) > 0.5]$.
    Perform an RL update for θ using the rewards r_τ.
    Synchronise θ and ϕ with other workers.
    $t \leftarrow t + R$
  end while
end while

Chapter 5

PROLOGUE TO SECOND ARTICLE

5.1. Article Details

BabyAI: A Platform to Study the Sample Efficiency of Grounded Language Learning. Maxime Chevalier-Boivert*, Dzmitry Bahdanau*, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio (* denotes equal contribution). International Conference on Learning Representations 2019.

Personal Contribution BabyAI was a team effort that was initiated by Yoshua Bengio. Initially, I participated in the project as an advisor and helped to shape its vision as a data efficiency study. Gradually, my involvement became more substantial as I started closely supervising Salem Lahlou, Lucas Willems, Chitwan Saharia and Thien Huu Nguyen in their work on exploration, curriculum learning, reinforcement learning, imitation learning and neural architectures for the platform. Eventually, I inherited the code that other project participants had written and made it all work together by fixing several critical bugs and tuning the hyperparameters. I designed BabyAI language and levels. I rewrote and optimized the bot that was originally implemented by Maxime Chevalier-Boivert and later developed by Salem Lahlou. I wrote a large part of the paper and reran all the experiments to ensure reproducibility. The credit for development of the environment, implementation of BabyAI language and all levels goes entirely to Maxime Chevalier-Boisvert.

5.2. Context

The original goal of the project was to construct a game-like setup in which human players interactively teach a deep learning agent to understand language (hence the name BabyAI). We decided to conduct a feasibility study first to investigate how much time a human would have to spend in order to teach such an agent. Preliminary investigations showed a huge gap between what deep learning methods are capable of and what would be required to actually put a human in the loop of learning. Quantifying this gap and building a platform that supports further work on data efficiency became the new and the final focus of the project.

5.3. Contributions

The main contribution is the BabyAI platform with 19 levels of instruction-following tasks, as well as the comparatively rich synthetic instruction language that it employs. We report baseline data efficiency results for a number of approaches, including reinforcement learning, imitation learning and basic curriculum learning.

5.4. Aftermath

In our follow-up work using the BabyAI platform we have recently discovered that sample efficiency of imitation learning can be improved by a factor of ≈ 3.5 by applying a relatively minor change to the neural model (namely removing the max-pooling operations that were performed too early in the visual pipeline). We are planning to investigate this phenomenon further and share the findings in a technical report.

Chapter 6

BABYAI: A PLATFORM TO STUDY THE SAMPLE EFFICIENCY OF GROUNDED LANGUAGE LEARNING

6.1. Introduction

How can a human train an intelligent agent to understand natural language instructions? We believe that this research question is important from both technological and scientific perspectives. No matter how advanced AI technology becomes, human users will likely want to customize their intelligent helpers to better understand their desires and needs. On the other hand, developmental psychology, cognitive science and linguistics study similar questions but applied to human children, and a synergy is possible between research in grounded language learning by computers and research in human language acquisition.

In this work, we present the BabyAI research platform, whose purpose is to facilitate research on grounded language learning. In our platform we substitute a simulated human expert for a real human; yet our aspiration is that BabyAI-based studies enable substantial progress towards putting an actual human in the loop. The current domain of BabyAI is a 2D gridworld in which synthetic natural-looking instructions (e.g. "put the red ball next to the box on your left") require the agent to navigate the world (including unlocking doors) and move objects to specified locations. BabyAI improves upon similar prior setups (Hermann et al., 2017; Chaplot et al., 2018; Yu et al., 2018) by supporting simulation of certain essential aspects of the future human in the loop agent training: *curriculum learning* and *interactive teaching*. The usefulness of curriculum learning for training machine learning models has been demonstrated numerous times in the literature (Bengio et al., 2009; Kumar et al., 2010; Zaremba and Sutskever, 2015; Graves et al., 2016), and we believe that gradually increasing the difficulty of the task will likely be essential for achieving efficient human-machine teaching, as in the case of human-human teaching. To facilitate curriculum learning studies, BabyAI currently features 19 levels in which the difficulty of the environment configuration

and the complexity of the instruction language are gradually increased. Interactive teaching, i.e. teaching differently based on what the learner can currently achieve, is another key capability of human teachers. Many advanced agent training methods, including DAGGER (Ross et al., 2011), TAMER (Warnell et al., 2017) and learning from human preferences (Wilson et al., 2012; Christiano et al., 2017), assume that interaction between the learner and the teacher is possible. To support interactive experiments, BabyAI provides a bot agent that can be used to generate new demonstrations on the fly and advise the learner on how to continue acting.

Arguably, the main obstacle to language learning with a human in the loop is the amount of data (and thus human-machine interactions) that would be required. Deep learning methods that are used in the context of imitation learning or reinforcement learning paradigms have been shown to be very effective in both simulated language learning settings (Mei et al., 2016; Hermann et al., 2017) and applications (Sutskever et al., 2014; Bahdanau et al., 2015; Wu et al., 2016). These methods, however, require enormous amounts of data, either in terms of millions of reward function queries or hundreds of thousands of demonstrations. To show how our BabyAI platform can be used for sample efficiency research, we perform several case studies. In particular, we estimate the number of demonstrations/episodes required to solve several levels with imitation and reinforcement learning baselines. As a first step towards improving sample efficiency, we additionally investigate to which extent pretraining and interactive imitation learning can improve sample efficiency.

The concrete contributions of this paper are two-fold. First, we contribute the BabyAI research platform for learning to perform language instructions with a simulated human in the loop. The platform already contains 19 levels and can easily be extended. Second, we establish baseline results for all levels and report sample efficiency results for a number of learning approaches. The platform and pretrained models are available online. We hope that BabyAI will spur further research towards improving sample efficiency of grounded language learning, ultimately allowing human-in-the-loop training.

6.2. Related Work

There are numerous 2D and 3D environments for studying synthetic language acquisition. (Hermann et al., 2017; Chaplot et al., 2018; Yu et al., 2018; Wu et al., 2018). Inspired by these efforts, BabyAI augments them by uniquely combining three desirable features. First, BabyAI supports world state manipulation, missing in the visually appealing 3D environments of Hermann et al. (2017), Chaplot et al. (2018) and Wu et al. (2018). In these environments, an agent can navigate around, but cannot alter its state by, for instance, moving objects. Secondly, BabyAI introduces partial observability (unlike the gridworld of Bahdanau et al. (2019a)). Thirdly, BabyAI provides a systematic definition of the synthetic

language. As opposed to using instruction templates, the Baby Language we introduce defines the semantics of all utterances generated by a context-free grammar (Section 6.3.2). This makes our language richer and more complete than prior work. Most importantly, BabyAI provides a simulated human expert, which can be used to investigate human-in-the-loop training, the aspiration of this paper.

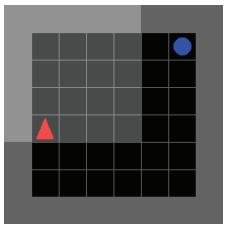
Currently, most general-purpose simulation frameworks do not feature language, such as PycoLab (DeepMind, 2017), MazeBase (Sukhbaatar et al., 2015), Gazebo (Koenig and Howard, 2004), VizDoom (Kempka et al., 2016), DM-30 (Espeholt et al., 2018), and AI2-Thor (Kolve et al., 2017). Using a more realistic simulated environment such as a 3D rather than 2D world comes at a high computational cost. Therefore, BabyAI uses a gridworld rather than 3D environments. As we found that available gridworld platforms were insufficient for defining a compositional language, we built a MiniGrid environment for BabyAI.

General-purpose RL testbeds such as the Arcade Learning Environment (Bellemare et al., 2013), DM-30 (Espeholt et al., 2018), and MazeBase (Sukhbaatar et al., 2015) do not assume a human-in-the-loop setting. In order to simulate this, we have to assume that all rewards (except intrinsic rewards) would have to be given by a human, and are therefore rather expensive to get. Under this assumption, imitation learning methods such as behavioral cloning, Searn (Daumé III et al., 2009), DAGGER (Ross et al., 2011) or maximum-entropy RL (Ziebart et al., 2008) are more appealing, as more learning can be achieved per human-input unit.

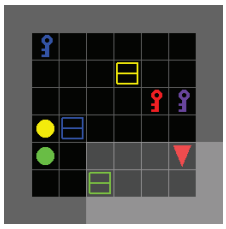
Similar to BabyAI, studying sample efficiency of deep learning methods was a goal of the bAbI tasks (Weston et al., 2016), which tested reasoning capabilities of a learning agent. Our work differs in both of the object of the study (grounded language with a simulated human in the loop) and in the method: instead of generating a fixed-size dataset and measuring the performance, we measure how much data a general-purpose model would require to get close-to-perfect performance.

There has been much research on instruction following with natural language (Tellex et al., 2011; Chen and Mooney, 2011; Artzi and Zettlemoyer, 2013; Mei et al., 2016; Williams et al., 2018) as well as several datasets including SAIL (Macmahon et al., 2006; Chen and Mooney, 2011) and Room-to-Room (Anderson et al., 2018b). Instead of using natural language, BabyAI utilises a synthetic Baby language, in order to fully control the semantics of an instruction and easily generate as much data as needed.

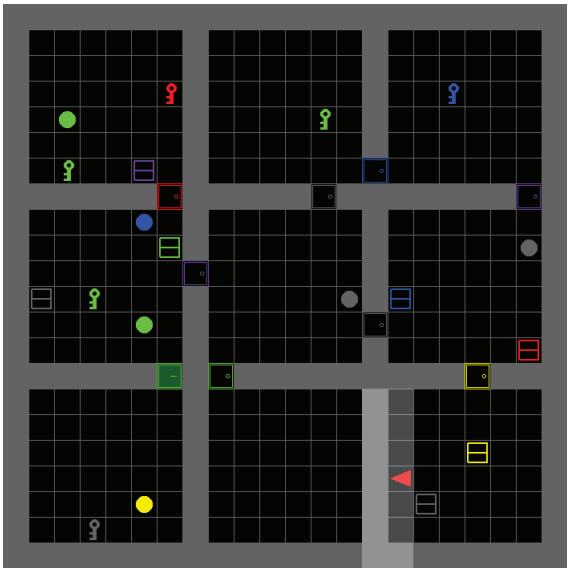
Finally, Wang et al. (2016) presented a system that interactively learned language from a human. We note that their system relied on substantial amounts of prior knowledge about the task, most importantly a task-specific executable formal language.



GoToObj: "go to the blue ball"



PutNextLocal: "put the blue key next to the green ball"



BossLevel: "pick up the grey box behind you, then go to the grey key and open a door". Note that the green door near the bottom left needs to be unlocked with a green key, but this is not explicitly stated in the instruction.

FIGURE 6.1. Three BabyAI levels built using the MiniGrid environment. The red triangle represents the agent, and the light-grey shaded area represents its field of view (partial observation).

6.3. BabyAI Platform

The BabyAI platform that we present in this work comprises an efficiently simulated gridworld environment (MiniGrid) and a number of instruction-following tasks that we call *levels*, all formulated using subsets of a synthetic language (Baby Language). The platform also includes a bot that can generate successful demonstrations for all BabyAI levels. All the code is available online at https://github.com/mila-iqia/babyai/tree/iclr19.

6.3.1. MiniGrid Environment

Studies of sample efficiency are very computationally expensive given that multiple runs are required for different amounts of data. Hence, in our design of the environment, we have aimed for a minimalistic and efficient environment which still poses a considerable challenge for current general-purpose agent learning methods. We have implemented MiniGrid, a partially observable 2D gridworld environment. The environment is populated with entities of different colors, such as the agent, balls, boxes, doors and keys (see Figure 6.1). Objects can be picked up, dropped and moved around by the agent. Doors can be unlocked with keys matching their color. At each step, the agent receives a 7x7 representation of its field of view (the grid cells in front of it) as well as a Baby Language instruction (textual string).

The MiniGrid environment is fast and lightweight. Throughput of over 3000 frames per second is possible on a modern multi-core laptop, which makes experimentation quicker and more accessible. The environment is open source, available online, and supports integration with OpenAI Gym. For more details, see Appendix B.1.

6.3.2. Baby Language

We have developed a synthetic Baby Language to give instructions to the agent as well as to automatically verify their execution. Although Baby Language utterances are a comparatively small subset of English, they are combinatorially rich and unambigously understood by humans. The language is intentionally kept simple, but still exhibits interesting combinatorial properties, and contains 2.48×10^{19} possible instructions. In this language, the agent can be instructed to go to objects, pick up objects, open doors, and put objects next to other objects. The language also expresses the conjunction of several such tasks, for example "put a red ball next to the green box after you open the door". The Backus-Naur Form (BNF) grammar for the language is presented in Figure 6.2 and some example instructions drawn from this language are shown in Figure 6.3. In order to keep the resulting instructions readable by humans, we have imposed some structural restrictions on this language: the *and* connector can only appear inside the *then* and *after* forms, and instructions can contain no more than one *then* or *after* word.

⟨Sent⟩	⊨	⟨Sent1⟩ | ⟨Sent1⟩ ',' then ⟨Sent1⟩ | ⟨Sent1⟩ after you ⟨Sent1⟩
⟨Sent1⟩	⊨	⟨Clause⟩ | ⟨Clause⟩ and ⟨Clause⟩
⟨Clause⟩	⊨	go to ⟨Descr⟩ | pick up ⟨DescrNotDoor⟩ | open ⟨DescrDoor⟩ | put ⟨DescrNotDoor⟩ next to ⟨Descr⟩
⟨DescrDoor⟩	⊨	⟨Article⟩ ⟨Color⟩ door ⟨LocSpec⟩
⟨DescrBall⟩	⊨	⟨Article⟩ ⟨Color⟩ ball ⟨LocSpec⟩
⟨DescrBox⟩	⊨	⟨Article⟩ ⟨Color⟩ box ⟨LocSpec⟩
⟨DescrKey⟩	⊨	⟨Article⟩ ⟨Color⟩ key ⟨LocSpec⟩
⟨Descr⟩	⊨	⟨DescrDoor⟩ | ⟨DescrBall⟩ | ⟨DescrBox⟩ | ⟨DescrKey⟩
⟨DescrNotDoor⟩	⊨	⟨DescrBall⟩ | ⟨DescrBox⟩ | ⟨DescrKey⟩
⟨LocSpec⟩	⊨	ϵ | on your left | on your right | in front of you | behind you
⟨Color⟩	⊨	ϵ | red | green | blue | purple | yellow | grey
⟨Article⟩	⊨	the | a

FIGURE 6.2. BNF grammar productions for the Baby Language

The BabyAI platform includes a *verifier* which checks if an agent's sequence of actions successfully achieves the goal of a given instruction within an environment. Descriptors in the language refer to one or to multiple objects. For instance, if an agent is instructed to "go to a

go to the red ball

open the door on your left

put a ball next to the blue door

open the yellow door and go to the key behind you

put a ball next to a purple door after you put a blue box next to a grey box and pick up the purple box

FIGURE 6.3. Example Baby Language instructions

red door", it can successfully execute this instruction by going to any of the red doors in the environment. The *then* and *after* connectors can be used to sequence subgoals. The *and* form implies that both subgoals must be completed, without ordering constraints. Importantly, Baby Language instructions leave details about the execution implicit. An agent may have to find a key and unlock a door, or move obstacles out of the way to complete instructions, without this being stated explicitly.

6.3.3. BabyAI Levels

There is abundant evidence in prior literature which shows that a curriculum may greatly facilitate learning of complex tasks for neural architectures (Bengio et al., 2009; Kumar et al., 2010; Zaremba and Sutskever, 2015; Graves et al., 2016). To investigate how a curriculum improves sample efficiency, we created 19 *levels* which require understanding only a limited subset of Baby Language within environments of varying complexity. Formally, a level is a distribution of *missions*, where a mission combines an instruction within an initial environment state. We built levels by selecting *competencies* necessary for each level and implementing a generator to generate missions solvable by an agent possessing only these competencies. Each competency is informally defined by specifying what an agent should be able to do:

- **Room Navigation (ROOM):** navigate a 6x6 room.
- **Ignoring Distracting Boxes (DISTR-BOX):** navigate the environment even when there are multiple distracting grey box objects in it.
- **Ignoring Distractors (DISTR):** same as DISTR-BOX, but distractor objects can be boxes, keys or balls of any color.
- **Maze Navigation (MAZE):** navigate a 3x3 maze of 6x6 rooms, randomly interconnected by doors.
- **Unblocking the Way (UNBLOCK):** navigate the environment even when it requires moving objects out of the way.
- **Unlocking Doors (UNLOCK):** to be able to find the key and unlock the door if the instruction requires this explicitly.

- **Guessing to Unlock Doors (IMP-UNLOCK):** to solve levels that require unlocking a door, even if this is not explicitly stated in the instruction.
- **Go To Instructions (GOTO):** understand "go to" instructions, e.g. "go to the red ball".
- **Open Instructions (OPEN):** understand "open" instructions, e.g. "open the door on your left".
- **Pickup Instructions (PICKUP):** understand "pick up" instructions, e.g. "pick up a box".
- **Put Instructions (PUT):** understand "put" instructions, e.g. "put a ball next to the blue key".
- **Location Language (LOC):** understand instructions where objects are referred to by relative location as well as their shape and color, e.g. "go to the red ball in front of you".
- **Sequences of Commands (SEQ):** understand composite instructions requiring an agent to execute a sequence of instruction clauses, e.g. "put red ball next to the green box after you open the door".

Table 6.1 lists all current BabyAI levels together with the competencies required to solve them. These levels form a progression in terms of the competencies required to solve them, culminating with the BossLevel, which requires mastering all competencies. The definitions of competencies are informal and should be understood in the minimalistic sense, i.e. to test the ROOM competency we have built the GoToObj level where the agent needs to reach the only object in an empty room. Note that the GoToObj level does not require the GOTO competency, as this level can be solved without any language understanding, since there is only a single object in the room. However, solving the GoToLocal level, which instructs the agent to go to a specific object in the presence of multiple distractors, requires understanding GOTO instructions.

6.3.4. The Bot Agent

The bot is a key ingredient intended to perform the role of a simulated human teacher. For any of the BabyAI levels, it can generate demonstrations or suggest actions for a given environment state. Whereas the BabyAI learner is meant to be generic and should scale to new and more complex tasks, the bot is engineered using knowledge of the tasks. This makes sense since the bot stands for the human in the loop, who is supposed to understand the environment, how to solve missions, and how to teach the baby learner. The bot has direct access to a tree representation of instructions, and so does not need to parse the Baby Language. Internally, it executes a stack machine in which instructions and subgoals are represented. The stack-based design allows the bot to interrupt what it is currently doing

TABLE 6.1. BabyAI Levels and the required competencies

	ROOM	DISTR-BOX	DISTR	MAZE	UNBLOCK	UNLOCK	IMP-UNLOCK	GOTO	OPEN	PICKUP	PUT	LOC	SEQ
GoToObj	x												
GoToRedBallGrey	x	x											
GoToRedBall	x	x	x										
GoToLocal	x	x	x					x					
PutNextLocal	x	x	x								x		
PickupLoc	x	x	x							x		x	
GoToObjMaze	x			x									
GoTo	x	x	x	x				x					
Pickup	x	x	x	x						x			
UnblockPickup	x	x	x	x	x					x			
Open	x	x	x	x					x				
Unlock	x	x	x	x		x			x				
PutNext	x	x	x	x							x		
Synth	x	x	x	x	x	x		x	x	x	x		
SynthLoc	x	x	x	x	x	x		x	x	x	x	x	
GoToSeq	x	x	x	x				x					x
SynthSeq	x	x	x	x	x	x		x	x	x	x	x	x
GoToImpUnlock	x	x	x	x			x	x					
BossLevel	x	x	x	x	x	x	x	x	x	x	x	x	x

to achieve a new subgoal, and then resume the original task. For example, going to a given object will require exploring the environment to find that object.

The subgoals which the bot implements are:

- **Open:** Open a door that is in front of the agent.
- **Close:** Close a door that is in front of the agent.
- **Pickup:** Execute the pickup action (pick up an object).
- **Drop:** Execute the drop action (drop an object being carried).
- **GoNextTo:** Go next to an object matching a given (type, color) description or next to a cell at a given position.
- **Explore:** Uncover previously unseen parts of the environment.

All of the Baby Language instructions are decomposed into these internal subgoals which the bot knows how to solve. Many of these subgoals, during their execution, can also push new subgoals on the stack. A central part of the design of the bot is that it keeps track of the grid cells of the environment which it has and has not seen. This is crucial to ensure that the bot can only use information which it could realistically have access to by exploring the

environment. Exploration is implemented as part of the Explore subgoal, which is recursive. For instance, exploring the environment may require opening doors, or moving objects that are in the way. Opening locked doors may in turn require finding a key, which may itself require exploration and moving obstructing objects. Another key component of the bot's design is a shortest path search routine. This is used to navigate to objects, to locate the closest door, or to navigate to the closest unexplored cell.

6.4. Experiments

We assess the difficulty of BabyAI levels by training a behavioral cloning baseline for each level. Furthermore, we estimate how much data is required to solve some of the simpler levels and study to which extent the data demands can be reduced by using basic curriculum learning and interactive teaching methods. All the code that we use for the experiments, as well as containerized pretrained models, is available online.

6.4.1. Setup

The BabyAI platform provides by default a 7x7x3 symbolic observation x_t (a partial and local egocentric view of the state of the environment) and a variable length instruction c as inputs at each time step. We use a basic model consisting of standard components to predict the next action a based on x and c. In particular, we use a GRU (Cho et al., 2014) to encode the instruction and a convolutional network with two batch-normalized (Ioffe and Szegedy, 2015) FiLM (Perez et al., 2017) layers to jointly process the observation and the instruction. An LSTM (Hochreiter and Schmidhuber, 1997) memory is used to integrate representations produced by the FiLM module at each step. Our model is thus similar to the gated-attention model used by Chaplot et al. (2018), inasmuch as gated attention is equivalent to using FiLM without biases and only at the output layer.

We have used two versions of our model, to which we will refer as the Large model and the Small model. In the Large model, the memory LSTM has 2048 units and the instruction GRU is bidirectional and has 256 units. Furthermore, an attention mechanism (Bahdanau et al., 2015) is used to focus on the relevant states of the GRU. The Small model uses a smaller memory of 128 units and encodes the instruction with a unidirectional GRU and no attention mechanism.

In all our experiments, we used the Adam optimizer (Kingma and Ba, 2015) with the hyperparameters $\alpha = 10^{-4}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-5}$. In our imitation learning (IL) experiments, we truncated the backpropagation through time at 20 steps for the Small model and at 80 steps for the Large model. For our reinforcement learning experiments, we used the Proximal Policy Optimization (PPO, Schulman et al., 2017) algorithm with parallelized data collection. Namely, we performed 4 epochs of PPO using 64 rollouts of length 40 collected with multiple processes. We gave a non-zero reward to the agent only

TABLE 6.2. Baseline imitation learning results for all BabyAI levels. Each model was trained with 1M demonstrations from the respective level. For reference, we also list the mean and standard deviation of demonstration length for each level.

Model	Success Rate (%)	Demo Length (Mean ± Std)
GoToObj	100	5.18±2.38
GoToRedBallGrey	100	5.81±3.29
GoToRedBall	100	5.38±3.13
GoToLocal	99.8	5.04±2.76
PutNextLocal	99.2	12.4±4.54
PickupLoc	99.4	6.13±2.97
GoToObjMaze	99.9	70.8±48.9
GoTo	99.4	56.8±46.7
Pickup	99	57.8±46.7
UnblockPickup	99	57.2±50
Open	100	31.5±30.5
Unlock	98.4	81.6±61.1
PutNext	98.8	89.9±49.6
Synth	97.3	50.4±49.3
SynthLoc	97.9	47.9±47.9
GoToSeq	95.4	72.7±52.2
SynthSeq	87.7	81.8±61.3
GoToImpUnlock	87.2	110±81.9
BossLevel	77	84.3±64.5

when it fully completed the mission, and the magnitude of the reward was $1 - 0.9n/n_{max}$, where n is the length of the successful episode and n_{max} is the maximum number of steps that we allowed for completing the episode, different for each mission. The future returns were discounted with a factor $\gamma = 0.99$. For generalized advantage estimation (Schulman et al., 2015) in PPO we used $\lambda = 0.99$.

In all our experiments we reported the success rate, defined as the ratio of missions of the level that the agent was able to accomplish within n_{max} steps.

Running the experiments outlined in this section required between 20 and 50 GPUs over two weeks. At least as much computing was required for preliminary investigations.

6.4.2. Baseline Results

To obtain baseline results for all BabyAI levels, we have trained the Large model (see Section 6.4.1) with imitation learning using one million demonstration episodes for each level. The demonstrations were generated using the bot described in Section 6.3.4. The models were trained for 40 epochs on levels with a single room and for 20 epochs on levels with a 3x3 maze of rooms. Table 6.2 reports the maximum success rate on a validation set of 512

episodes. All of the single-room levels are solved with a success rate of 100.0%. As a general rule, levels for which demonstrations are longer tend to be more difficult to solve.

Using 1M demonstrations for levels as simple as GoToRedBall is very inefficient and hardly ever compatible with the long-term goal of enabling human teaching. The BabyAI platform is meant to support studies of how neural agents can learn with less data. To bootstrap such studies, we have computed baseline sample efficiencies for imitation learning and reinforcement learning approaches to solving BabyAI levels. We say an agent solves a level if it reaches a success rate of at least 99%. We define the sample efficiency as the minimum number of demonstrations or RL episodes required to train an agent to solve a given level. To estimate the thus defined sample efficiency for imitation learning while staying within a reasonable computing budget, we adopt the following procedure. For a given level, we first run three experiments with 10^6 demonstrations. In the remaining M experiments we use $k_1 = 2^{l_0}, k_2 = 2^{l_0+d}, \ldots, k_M = 2^{l_0+(M-1)d}$ demonstrations respectively. We use different values of l_0, M for each level to ensure that we run experiments with not enough, just enough and more than enough demonstrations. Same value of $d = 0.2$ is used in all imitation learning experiments. For each experiment i, we measure the best smoothed online validation performance s_i that is achieved during the first $2T$ training steps, where $T = (T_1 + T_2 + T_3)/3$ is the average number of training steps required to solve the level in the three runs with 10^6 demonstrations. We then fit a Gaussian Process (GP) model (Rasmussen and Williams, 2005) with noisy observations using (k_i, s_i) as training data in order to interpolate between these data points. The GP posterior is fully tractable, which allows us to compute analytically the posterior distribution of the expected success rate, as well as the posterior over the minimum number of samples k_{min} that is sufficient to solve the level. We report the 99% credible interval for k_{min}. We refer the reader to Section 6.6 for a more detailed explanation of this procedure.

We estimate sample efficiency of imitation learning on 6 chosen levels. The results are shown in Table 6.3 (see "IL from Bot" column). In the same table (column "RL") we report the 99% confidence interval for the number of episodes that were required to solve each of these levels with RL, and as expected, the sample efficiency of RL is substantially worse than that of IL (anywhere between 2 to 10 times in these experiments).

To analyze how much the sample efficiency of IL depends on the source of demonstrations, we try generating demonstrations from agents that were trained with RL in the previous experiments. The results for the 3 easiest levels are reported in the "IL from RL Expert" column in Table 6.5. Interestingly, we found that the demonstrations produced by the RL agent are easier for the learner to imitate. The difference is most significant for GoToRedBallGrey, where less than 2K and more than 8K RL and bot demonstrations respectively are required to solve the level. For GoToRedBall and GoToLocal, using RL demonstrations

TABLE 6.3. The sample efficiency of imitation learning (IL) and reinforcement learning (RL) as the number of demonstrations (episodes) required to solve each level. All numbers are thousands. For the imitation learning results we report a 99% credible interval. For RL experiments we report the 99% confidence interval. See Section 6.4 for details.

Level	IL from Bot	RL
GoToRedBallGrey	8.431 - 12.43	15.9 - 17.4
GoToRedBall	49.67 - 62.01	261.1 - 333.6
GoToLocal	148.5 - 193.2	903 - 1114
PickupLoc	204.3 - 241.2	1447 - 1643
PutNextLocal	244.6 - 322.7	2186 - 2727
GoTo	341.1 - 408.5	816 - 1964

TABLE 6.4. The sample efficiency results for pretraining experiments. For each pair of base levels and target levels that we have tried, we report how many demonstrations (in thousands) were required, as well as the baseline number of demonstrations required for training from scratch. In both cases we report a 99% credible interval, see Section 6.4 for details. Note how choosing the right base levels (e.g. GoToLocal instead of GoToObjMaze) is crucial for pretraining to be helpful.

Base Levels	Target Level	Without Pretraining	With Pretraining
GoToLocal	GoTo	341 - 409	183 - 216
GoToObjMaze	GoTo	341 - 409	444 - 602
GoToLocal-GoToObjMaze	GoTo	341 - 409	173 - 216
GoToLocal	PickupLoc	204 - 241	71.2 - 88.9
GoToLocal	PutNextLocal	245 - 323	188 - 231

results in 1.5-2 times better sample efficiency. This can be explained by the fact that the RL expert has the same neural network architecture as the learner.

6.4.3. Curriculum Learning

To demonstrate how curriculum learning research can be done using the BabyAI platform, we perform a number of basic pretraining experiments. In particular, we select 5 combinations of base levels and a target level and study whether pretraining on base levels can help the agent master the target level with fewer demonstrations. The results are reported in Table 6.4. In four cases, using GoToLocal as one of the base levels reduces the number of demonstrations required to solve the target level. However, when only GoToObjMaze was used as the base level, we have not found pretraining to be beneficial. We find this counter-intuitive result interesting, as it shows how current deep learning methods often can not take the full advantage of available curriculums.

TABLE 6.5. The sample efficiency of imitation learning (IL) from an RL-pretrained expert and interactive imitation learning defined as the number of demonstrations required to solve each level. All numbers are in thousands. 99% credible intervals are reported in all experiments, see Section 6.4 for details.

Level	IL from Bot	IL from RL Expert	Interactive IL from Bot
GoToRedBallGrey	8.43 - 12.4	1.53 - 2.11	1.71 - 1.88
GoToRedBall	49.7 - 62	36.6 - 44.5	31.8 - 36
GoToLocal	148 - 193	74.2 - 81.8	93 - 107

6.4.4. Interactive Learning

Lastly, we perform a simple case study of how sample efficiency can be improved by interactively providing more informative examples to the agent based on what it has already learned. We experiment with an iterative algorithm for adaptively growing the agent's training set. In particular, we start with 2^{10} base demonstrations, and at each iteration we increase the dataset size by a factor of $2^{1/4}$ by providing bot demonstrations for missions on which the agent failed. After each dataset increase we train a new agent from scratch. We perform such dataset increases until the dataset reaches the final size is clearly sufficient to achieve 99% success rate. We repeat the experiment 3 times for levels GoToRedBallGrey, GoToRedBall and GoToLocal and then estimate how many interactively provided demonstrations would be required for the agent be 99% successful for each of these levels. To this end, we use the same GP posterior analysis as for regular imitation learning experiments.

The results for the interactive imitation learning protocol are reported in Table 6.5. For all 3 levels that we experimented with, we have observed substantial improvement over the vanilla IL, which is most significant (4 times less demonstrations) for GoToRedBallGrey and smaller (1.5-2 times less demonstrations) for the other two levels.

6.5. Conclusion & Future Work

We present the BabyAI research platform to study language learning with a human in the loop. The platform includes 19 levels of increasing difficulty, based on a decomposition of tasks into a set of basic competencies. Solving the levels requires understanding the Baby Language, a subset of English with a formally defined grammar which exhibits compositional properties. The language is minimalistic and the levels seem simple, but empirically we have found them quite challenging to solve. The platform is open source and extensible, meaning new levels and language concepts can be integrated easily.

The results in Section 6.4 suggest that current imitation learning and reinforcement learning methods scale and generalize poorly when it comes to learning tasks with a compositional structure. Hundreds of thousands of demonstrations are needed to learn tasks which seem

trivial by human standards. Methods such as curriculum learning and interactive learning can provide measurable improvements in terms of sample efficiency, but, in order for learning with an actual human in the loop to become realistic, an improvement of at least three orders of magnitude is required.

An obvious direction of future research to find strategies to improve sample efficiency of language learning. Tackling this challenge will likely require new models and new teaching methods. Approaches that involve an explicit notion of modularity and subroutines, such as Neural Module Networks (Andreas et al., 2016) or Neural Programmer-Interpreters (Reed and de Freitas, 2015), seem like a promising direction. It is our hope that the BabyAI platform can serve as a challenge and a benchmark for the sample efficiency of language learning for years to come.

6.6. Sample Efficiency Estimation

6.6.1. Reinforcement Learning

To estimate the number of episodes required for an RL agent to solve a BabyAI level, we monitored the agent's smoothed online success rate. We recorded the number of training episodes after which the smoothed performance crossed the 99% success rate threshold. Each experiment was repeated 10 times and the 99% t-test confidence interval is reported in Table 6.3.

6.6.2. Imitation Learning

Estimating how many demonstrations is required for imitation learning to achieve a given performance level is challenging. In principle, one can sample a dense grid of dataset sizes, train the model until full convergence on each of the resulting datasets, and find the smallest dataset size for which on average the model's best performance exceeds the target level. In practice, such a procedure would be prohibitively computationally expensive.

To make sample efficiency estimation practical, we designed a relatively cheap semi-automatic approximate protocol. We minimize computational resources by using early-stopping and non-parametric interpolation between different data points.

Early Stopping Using Normal Time

Understanding if a training run has converged and if the model's performance will not improve any further is non-trivial. To early-stop models in a consistent automatic way, we estimate the "normal" time T that training a model on a given level would take if an unlimited (in our case 10^6) number of demonstrations was available. To this end, we train 3 models with 10^6 demonstrations. We evaluate the online success rate after every 100 or 400 (depending on the model size) batches, each time using 512 different episodes.

The online success rate is smoothed using a sliding window of length 10. Let $s(k, j, t)$ denote the smoothed online performance for the j-th run with k demonstrations at time t. Using this notation, we compute the normal time T as $T = (T_1 + T_2 + T_3)/3$, where $T_i = \min_t \{t : s_j(10^6, j, t) > 99\}$. Once T is computed, it is used to early stop the remaining M runs that use different numbers of demonstrations k_i. Namely the result s_i of the i-th of these runs is computed as $s_i = \max_{t<2T} s(k_i, 1, t)$.

Interpolation Using Gaussian Processes

Given the success rate measurements $D = \{(k_i, s_i)\}_{i=1}^{M}$, $k_1 < k_2 < \ldots < k_M$, we estimate the minimum number of samples k_{min} that is required for the model to reach 99% average success rate. To this end, we a Gaussian Process (GP) model to interpolate between the available (k_i, s_i) data points (Rasmussen and Williams, 2005). GP is a popular model for non-linear regression, whose main advantage is principled modelling of predictions' uncertainty.

Specifically, we model the dependency between the success rate s and the number of examples k as follows:

$$f \sim GP_{RBF}(l), \tag{6.6.1}$$

$$\tilde{s}(k) = 99 + \sigma_f f(\log_2 k), \tag{6.6.2}$$

$$\epsilon(k) \sim N(0, 1), \tag{6.6.3}$$

$$s(k) = \tilde{s}(k) + \sigma_\epsilon \epsilon(k), \tag{6.6.4}$$

where RBF reflects the fact that we use the Radial Basis Function kernel, l is the kernel's length-scale parameter, $\epsilon(k)$ is white noise, σ_f and σ_ϵ add scaling to the GP f and the noise ϵ. Note the distinction between the average and the observed performances $\tilde{s}(k)$ and $s(k)$. Using the introduced notation, k_{min} can be formally defined as $k_{min} = \min_{k \in [k_1; k_M]} \tilde{s}(k) = 99$.

To focus on the interpolation in the region of interest, we drop all (k_i, s_i) data points for which $s_i < 95$. We then fit the model's hyperparameters l, σ_f and σ_ϵ by maximizing the likelihood of the remaining data points. To this end, we use the implementation from scikit-learn (Pedregosa et al., 2011). Once the model is fit, it defines a Gaussian posterior density $p(\tilde{s}(k'_1), \ldots, \tilde{s}(k'_{M'})|D)$ for any M' data points $k'_1, k'_2, \ldots, k'_{M'}$. It also defines a probability distribution $p(k_{min}|D)$. We are not aware of an analytic expression for $p(k_{min}|D)$, and hence we compute a numerical approximation as follows. We sample a dense log-scale grid of M' points $k'_1, k'_2, \ldots, k'_{M'}$ in the range $[k_1; k_M]$. For each number of demonstrations k'_i we approximate the probability $p(k'_{i-1} < k_{min} < k'_i|D)$ that $\tilde{s}(k)$ crosses the 99% threshold somewhere between k'_{i-1} and k'_i as follows:

$$p(k'_{i-1} < k_{min} < k'_i|D) \approx p'_i = p(\tilde{s}(k'_1) < 99, \ldots, \tilde{s}(k'_{i-1}) < 99, \tilde{s}(k'_i) > 99|D) \tag{6.6.5}$$

Equation 6.6.5 is an approximation because the posterior $\tilde{s}$ is not necessarily monotonic. In practice, we observed that the monotonic nature of the observed data D shapes the posterior accordingly. We use the probabilities p'_i to construct the following discrete approximation of the posterior $p(k_{min}|D)$:

$$p(k_{min}|D) \approx \sum_{i=1}^{M} p'_i \delta(k'_i) \tag{6.6.6}$$

where $\delta(k'_i)$ are Dirac delta-functions. Such a discrete approximation is sufficient for the purpose of computing 99% credible intervals for k_{min} that we report in the paper.

Chapter 7

PROLOGUE TO THIRD ARTICLE

7.1. Article Details

Systematic Generalization: What Is Required and Can It Be Learned? Dzmitry Bahdanau*, Shikhar Murty*, Michael Noukhovitch, Thien Huu Nguyen, Harm de Vries, and Aaron Courville (* denotes equal contribution). International Conference on Learning Representations 2019.

Personal Contribution. The initiative to perform a minimalistic study of systematic generalization of language grounding was mine. I designed the first version of the task that was strongly influenced by CLEVR-CoGenT and evaluated several baseline models on it. Thien Huu Ngueyn and Michael Noukhovitch reimplemented several baseline models for that stage of the project, which unfortunately did not yield interesting insights.

Second iteration of the project started from an insightful suggestion by Harm de Vries to think about object frequencies, rather than color-shape biases. This idea has gradually developed into the object-pair split that made it to the paper. Shikhar Murty, who at that point was starting his internship at our lab, implemented the new dataset based on my old code. Shikhar and I collaborated closely to design and implement the NMN models for the paper. Michael Noukhovitch implemented an additional baseline. All project participants participated in paper writing. After submission, I reran all the experiments, and then Shikhar and I performed the final round of editing. Aaron Courville supervised the project throughout its duration.

7.2. Context

Numerous studies showed that deep learning systems often achieve high test set performance by adapting to peculiarities of the data distribution. At roughly the same time, Lake and Baroni (2018) brought the concepts of systematicity and systematic generalization back to mainstream machine learning discourse. Lastly, NMNs, that combine the ideas of compositionality and deep learning were proposed by Andreas et al. (2016) in the context

of grounded language understanding. These three developments inspired us to do a study of systematic generalization that would (a) use a train-test split that is inspired by real world considerations (b) focus on the differences between usual deep learning models and NMNs.

7.3. Contributions

The key contribution is showcasing how NMNs that are constructed from generic modules can generalize much better than the conventional non-modular models. We furthermore analyze what is required for such a strong generalization, and find that choosing a correct layout of the modules is crucial for this purpose. Lastly, the paper highlights the challenges arising when one tries to learn the structural aspects of an NMN from a biased dataset in an end-to-end way.

Chapter 8

SYSTEMATIC GENERALIZATION: WHAT IS REQUIRED AND CAN IT BE LEARNED?

8.1. Introduction

In recent years, neural network based models have become the workhorse of natural language understanding and generation. They empower industrial machine translation (Wu et al., 2016) and text generation (Kannan et al., 2016) systems and show state-of-the-art performance on numerous benchmarks including Recognizing Textual Entailment (Gong et al., 2018), Visual Question Answering (Jiang et al., 2018), and Reading Comprehension (Wang et al., 2018). Despite these successes, a growing body of literature suggests that these approaches do not generalize outside of the specific distributions on which they are trained, something that is necessary for a language understanding system to be widely deployed in the real world. Investigations on the three aforementioned tasks have shown that neural models easily latch onto statistical regularities which are omnipresent in existing datasets (Agrawal et al., 2016; Gururangan et al., 2018; Jia and Liang, 2017) and extremely hard to avoid in large scale data collection. Having learned such dataset-specific solutions, neural networks fail to make correct predictions for examples that are even slightly out of domain, yet are trivial for humans. These findings have been corroborated by a recent investigation on a synthetic instruction-following task (Lake and Baroni, 2018), in which seq2seq models (Sutskever et al., 2014; Bahdanau et al., 2015) have shown little systematicity (Fodor and Pylyshyn, 1988) in how they generalize, that is they do not learn general rules on how to compose words and fail spectacularly when for example asked to interpret "jump twice" after training on "jump", "run twice" and "walk twice".

An appealing direction to improve the generalization capabilities of neural models is to add modularity and structure to their design to make them structurally resemble the kind of rules they are supposed to learn (Andreas et al., 2016; Gaunt et al., 2016). For example, in the Neural Module Network paradigm (NMN, Andreas et al. (2016)), a neural network is assembled from several *neural modules*, where each module is meant to perform

a particular subtask of the input processing, much like a computer program composed of functions. The NMN approach is intuitively appealing but its widespread adoption has been hindered by the large amount of domain knowledge that is required to decide (Andreas et al., 2016) or predict (Johnson et al., 2017; Hu et al., 2017) how the modules should be created (*parametrization*) and how they should be connected (*layout*) based on a natural language utterance. Besides, their performance has often been matched by more traditional neural models, such as FiLM (Perez et al., 2017), Relations Networks (Santoro et al., 2017), and MAC networks (Hudson and Manning, 2018). Lastly, generalization properties of NMNs, to the best of our knowledge, have not been rigorously studied prior to this work.

Here, we investigate the impact of explicit modularity and structure on systematic generalization of NMNs and contrast their generalization abilities to those of generic models. For this case study, we focus on the task of visual question answering (VQA), in particular its simplest binary form, when the answer is either "yes" or "no". Such a binary VQA task can be seen as a fundamental task of language understanding, as it requires one to evaluate the truth value of the utterance with respect to the state of the world. Among many systematic generalization requirements that are desirable for a VQA model, we choose the following basic one: *a good model should be able to reason about all possible object combinations despite being trained on a very small subset of them.* We believe that this is a key prerequisite to using VQA models in the real world, because they should be robust at handling unlikely combinations of objects. We implement our generalization demands in the form of a new synthetic dataset, called **S**patial **Q**ueries **O**n **O**bject **P**airs (SQOOP), in which a model has to perform spatial relational reasoning about pairs of randomly scattered letters and digits in the image (e.g. answering the question "*Is there a letter A left of a letter B?*"). The main challenge in SQOOP is that models are evaluated on all possible object pairs, but trained on only a subset of them.

Our first finding is that NMNs do generalize better than other neural models when layout and parametrization are chosen appropriately. We then investigate which factors contribute to improved generalization performance and find that using a layout that matches the task (i.e. a tree layout, as opposed to a chain layout), is crucial for solving the hardest version of our dataset. Lastly, and perhaps most importantly, we experiment with existing methods for making NMNs more end-to-end by inducing the module layout (Johnson et al., 2017) or learning module parametrization through soft-attention over the question (Hu et al., 2017). Our experiments show that such end-to-end approaches often fail by not converging to tree layouts or by learning a blurred parameterization for modules, which results in poor generalization on the hardest version of our dataset. We believe that our findings challenge the intuition of researchers in the field and provide a foundation for improving systematic generalization of neural approaches to language understanding.

8.2. The SQOOP Dataset For Testing Systematic Generalization

We perform all experiments of this study on the SQOOP dataset. SQOOP is a minimalistic VQA task that is designed to test the model's ability to interpret unseen combinations of known relation and object words. Clearly, given known objects X, Y and a known relation R, a human can easily verify whether or not the objects X and Y are in relation R. Some instances of such queries are common in daily life (*is there a cup on the table*), some are extremely rare (*is there a violin under the car*), and some are unlikely but have similar, more likely counter-parts (*is there grass on the frisbee* vs *is there a frisbee on the grass*). Still, a person can easily answer these questions by understanding them as just the composition of the three separate concepts. Such compositional reasoning skills are clearly required for language understanding models, and SQOOP is explicitly designed to test for them.

Concretely speaking, SQOOP requires observing a 64×64 RGB image x and answering a yes-no question $q = \mathrm{X\,R\,Y}$ about whether objects X and Y are in a spatial relation R. The questions are represented in a redundancy-free X R Y form; we did not aim to make the questions look like natural language. Each image contains 5 randomly chosen and randomly positioned objects. There are 36 objects: the latin letters A-Z and digits 0-9, and there are 4 relations: LEFT_OF, RIGHT_OF, ABOVE, and BELOW. This results in $36 \cdot 35 \cdot 4 = 5040$ possible unique questions (we do not allow questions about identical objects). To make negative examples challenging, we ensure that both X and Y of a question are always present in the associated image and that there are distractor objects $\mathrm{Y}' \neq \mathrm{Y}$ and $\mathrm{X}' \neq \mathrm{X}$ such that $\mathrm{X\,R\,Y}'$ and $\mathrm{X}'\mathrm{R\,Y}$ are both true for the image. These extra precautions guarantee that answering a question requires the model to locate all possible X and Y then check if any pair of them are in the relation R. Two SQOOP examples are shown in Figure 8.3.

Our goal is to discover which models can correctly answer questions about all $36 \cdot 35$ possible object pairs in SQOOP after having been trained on only a subset. For this purpose we build training sets containing $36 \cdot 4 \cdot k$ unique questions by sampling k different right-hand-side (RHS) objects Y_1, Y_2, ..., Y_k for each left-hand-side (LHS) object X. We use this procedure instead of just uniformly sampling object pairs in order to ensure that each object appears in at least one training question, thereby keeping the all versions of the dataset solvable. We will refer to k as the *#rhs/lhs parameter* of the dataset. Our test set is composed from the remaining $36 \cdot 4 \cdot (35 - k)$ questions. We generate training and test sets for rhs/lhs values of 1,2,4,8 and 18, as well as a control version of the dataset, #rhs/lhs=35, in which both the training and the test set contain all the questions (with different images). Note that lower #rhs/lhs versions are harder for generalization due to the presence of spurious dependencies between the words X and Y to which the models may adapt. In order to exclude a possible compounding factor of overfitting on the training

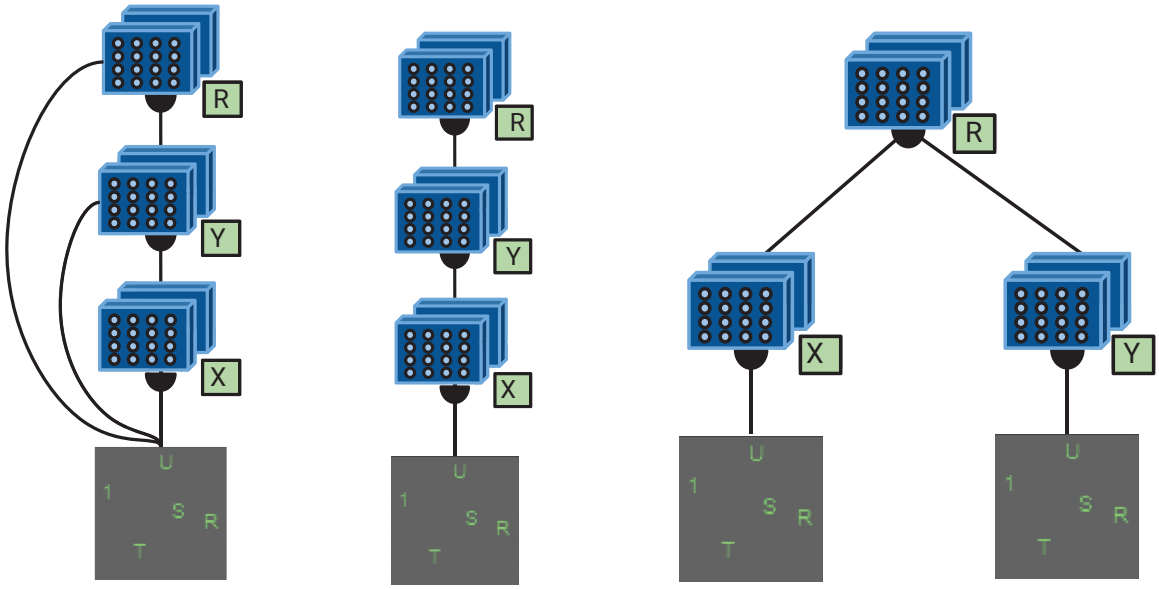


FIGURE 8.1. Different NMN layouts: *NMN-Chain-Shortcut* (**left**), *NMN-Chain* (**center**), *NMN-Tree* (**right**). See Section 8.3.2 for details.

images, all our training sets contain 1 million examples, so for a dataset with #rhs/lhs $= k$ we generate approximately $10^6/(36 \cdot 4 \cdot k)$ different images per unique question.

8.3. MODELS

A great variety of VQA models have been recently proposed in the literature, among which we can distinguish two trends. Some of the recently proposed models, such as FiLM (Perez et al., 2017) and Relation Networks (RelNet, Santoro et al. (2017)) are highly *generic* and do not require any task-specific knowledge to be applied on a new dataset. On the opposite end of the spectrum are *modular* and *structured* models, typically flavours of Neural Module Networks (Andreas et al., 2016), that do require some knowledge about the task at hand to be instantiated. Here, we evaluate systematic generalization of several state-of-the-art models in both families. In all models, the image x is first fed through a CNN based network, that we refer to as the *stem*, to produce a feature-level 3D tensor h_{x}. This is passed through a model-specific computation conditioned on the question q, to produce a joint representation $h_{q\,\mathrm{x}}$. Lastly, this representation is fed into a fully-connected *classifier* network to produce logits for prediction. Therefore, the main difference between the models we consider is how the computation $h_{q\,\mathrm{x}} = model(h_{\mathrm{x}}, q)$ is performed.

8.3.1. Generic Models

We consider four generic models in this paper: CNN+LSTM, FiLM, Relation Network (RelNet), and Memory-Attention-Control (MAC) network. For CNN+LSTM, FiLM, and

(A) S above T? **Yes**

(B) W left of A? **No**

FIGURE 8.3. A positive (**left**) and negative (**right**) example from the SQOOP dataset.

RelNet models, the question q is first encoded into a fixed-size representation h_q using a unidirectional LSTM network. **CNN+LSTM** flattens the 3D tensor h_{x} to a vector and concatenates it with h_q to produce $h_{q\mathrm{x}}$:

$$h_{q\mathrm{x}} = [flatten(h_{\mathrm{x}}); h_q]. \tag{8.3.1}$$

RelNet (Santoro et al., 2017) uses a network g which is applied to all pairs of feature columns of h_{x} concatenated with the question representation h_q, all of which is then pooled to obtain $h_{q\mathrm{x}}$:

$$h_{q\mathrm{x}} = \sum_{i,j} g(h_{\mathrm{x}}(i), h_{\mathrm{x}}(j), h_q) \tag{8.3.2}$$

where $h_x(i)$ is the i-th feature column of h_x. **FiLM** networks (Perez et al., 2017) use N convolutional FiLM blocks applied to h_{x}. A FiLM block is a residual block (He et al., 2016) in which a feature-wise affine transformation (FiLM layer) is inserted after the 2nd convolutional layer. The FiLM layer is conditioned on the question at hand via prediction of the scaling and shifting parameters γ_n and β_n:

$$[\gamma_n; \beta_n] = W_q^n h_q + b_q^n \tag{8.3.3}$$

$$\tilde{h}_{q\mathrm{x}}^n = BN(W_2^n * ReLU(W_1^n * h_{q\mathrm{x}}^{n-1} + b_n)) \tag{8.3.4}$$

$$h_{q\mathrm{x}}^n = h_{q\mathrm{x}}^{n-1} + ReLU(\gamma_n \odot \tilde{h}_{q\mathrm{x}}^n \oplus \beta_n) \tag{8.3.5}$$

where BN stands for batch normalization (Ioffe and Szegedy, 2015), $*$ stands for convolution and $\odot$ stands for element-wise multiplications. $h_{q\mathrm{x}}^n$ is the output of the n-th FiLM block and $h_{q\mathrm{x}}^0 = h_{\mathrm{x}}$. The output of the last FiLM block $h_{q\mathrm{x}}^N$ undergoes an extra 1×1 convolution and max-pooling to produce $h_{q\mathrm{x}}$. **MAC** network of Hudson and Manning (2018) produces $h_{q\mathrm{x}}$ by repeatedly applying a Memory-Attention-Composition (MAC) cell that is conditioned on the question through an attention mechanism. The MAC model is too complex to be fully described here and we refer the reader to the original paper for details.

8.3.2. Neural Module Networks

Neural Module Networks (NMN) (Andreas et al., 2016) are an elegant approach to question answering that constructs a question-specific network by composing together trainable

neural modules, drawing inspiration from symbolic approaches to question answering (Malinowski and Fritz, 2014). To answer a question with an NMN, one first constructs the computation graph by making the following decisions: (a) how many modules and of which types will be used, (b) how will the modules be connected to each other, and (c) how are these modules parametrized based on the question. We refer to the aspects (a) and (b) of the computation graph as the *layout* and the aspect (c) as the *parametrization*. In the original NMN and in many follow-up works, different module types are used to perform very different computations, e.g. the `Find` module from Hu et al. (2017) performs trainable convolutions on the input attention map, whereas the `And` module from the same paper computes an element-wise maximum for two input attention maps. In this work, we follow the trend of using more homogeneous modules started by Johnson et al. (2017), who use only two types of modules: unary and binary, both performing similar computations. We restrict our study to NMNs with homogeneous modules because they require less prior knowledge to be instantiated and because they performed well in our preliminary experiments despite their relative simplicity. We go one step further than Johnson et al. (2017) and retain a single binary module type, using a zero tensor for the second input when only one input is available. Additionally, we choose to use exactly three modules, which simplifies the layout decision to just determining how the modules are connected. Our preliminary experiments have shown that, even after these simplifications, NMNs are far ahead of other models in terms of generalization.

In the original NMN, the layout and parametrization were set in an ad-hoc manner for each question by analyzing a dependency parse. In the follow-up works (Johnson et al., 2017; Hu et al., 2017), these aspects of the computation are predicted by learnable mechanisms with the goal of reducing the amount of background knowledge required to apply the NMN approach to a new task. We experiment with the End-to-End NMN (N2NMN) (Hu et al., 2017) paradigm from this family, which predicts the layout with a seq2seq model (Sutskever et al., 2014) and computes the parametrization of the modules using a soft attention mechanism. Since all the questions in SQOOP have the same structure, we do not employ a seq2seq model but instead have a trainable layout variable and trainable attention variables for each module.

Formally, our NMN is constructed by repeatedly applying a *generic neural module* $f(\theta, \gamma, s^0, s^1)$, which takes as inputs the shared parameters θ, the question-specific parametrization γ and the left-hand side and right-hand side inputs s^0 and s^1. Three such modules are connected and conditioned on a question $q = (q_1, q_2, q_3)$ as follows:

$$\gamma_k = \sum_{i=1}^{3} \alpha^{k,i} e(q_i) \tag{8.3.6}$$

$$s_k^m = \sum_{j=-1}^{k-1} \tau_m^{k,j} s_j \tag{8.3.7}$$

$$s_k = f(\theta, \gamma_k, s_k^0, s_k^1) \tag{8.3.8}$$

$$h_{qx} = s_3 \tag{8.3.9}$$

In the equations above, $s_{-1} = 0$ is the zero tensor input, $s_0 = h_x$ are the image features outputted by the stem, e is the embedding table for question words. $k \in \{1, 2, 3\}$ is the module number, s_k is the output of the k-th module and s_k^m are its left ($m = 0$) and right ($m = 1$) inputs. We refer to $A = (\alpha^{k,i})$ and $T = (\tau_m^{k,j})$ as the *parametrization attention matrix* and the *layout tensor* respectively.

We experiment with two choices for the NMN's generic neural module: the Find module from Hu et al. (2017) and the Residual module from Johnson et al. (2017). The equations for the Residual module are as follows:

$$[W_1^k; b_1^k; W_2^k; b_2^k; W_3^k; b_3^k] = \gamma_k \tag{8.3.10}$$

$$\tilde{s_k} = ReLU(W_3^k * [s_k^0; s_k^1] + b_3^k), \tag{8.3.11}$$

$$f_{Residual}(\gamma_k, s_k^0, s_k^1) = ReLU(\tilde{s_k} + W_1^k * ReLU(W_2^k * \tilde{s_k} + b_2^k)) + b_1^k), \tag{8.3.12}$$

and for Find module as follows:

$$[W_1; b_1; W_2; b_2] = \theta, \tag{8.3.13}$$

$$f_{Find}(\theta, \gamma_k, s_k^0, s_k^1) = ReLU(W_1 * \gamma_k \odot ReLU(W_2 * \left[s_k^0; s_k^1\right] + b_2) + b_1). \tag{8.3.14}$$

In the formulas above all W's stand for convolution weights, and all b's are biases. Equations 8.3.10 and 8.3.13 should be understood as taking vectors γ_k and θ respectively and chunking them into weights and biases. The main difference between Residual and Find is that in Residual all parameters depend on the questions words (hence θ is omitted from the signature of $f_{Residual}$), where as in Find convolutional weights are the same for all questions, and only the element-wise multipliers γ_k vary based on the question. We note that the specific Find module we use in this work is slightly different from the one used in (Hu et al., 2017) in that it outputs a feature tensor, not just an attention map. This change was required in order to connect multiple Find modules in the same way as we connect multiple residual ones.

Based on the generic NMN model described above, we experiment with several specific architectures that differ in the way the modules are connected and parametrized (see Figure 8.1). In **NMN-Chain** the modules form a sequential chain. Modules 1, 2 and 3 are parametrized based on the first object word, second object word and the relation word respectively, which is achieved by setting the attention maps α_1, α_2, α_3 to the corresponding one-hot vectors. We also experiment with giving the image features h_x as the right-hand side input to all 3 modules and call the resulting model **NMN-Chain-Shortcut**. **NMN-Tree** is

similar to NMN-Chain in that the attention vectors are similarly hard-coded, but we change the connectivity between the modules to be tree-like. **Stochastic N2NMN** follows the N2NMN approach by Hu et al. (2017) for inducing layout. We treat the layout T as a stochastic latent variable. T is allowed to take two values: T_{tree} as in NMN-Tree, and T_{chain} as in NMN-Chain. We calculate the output probabilities by marginalizing out the layout i.e. probability of answer being "yes" is computed as $p(\text{yes}|x,q) = \sum_{T\in\{T_{tree},T_{chain}\}} p(\text{yes}|T,x,q)p(T)$. Lastly, **Attention N2NMN** uses the N2NMN method for learning parametrization (Hu et al., 2017). It is structured just like NMN-Tree but has α^k computed as $\text{softmax}(\tilde{\alpha}^k)$, where $\tilde{\alpha}^k$ is a trainable vector. We use Attention N2NMN only with the Find module because using it with the Residual module would involve a highly non-standard interpolation between convolutional weights.

8.4. Experiments

In our experiments we aimed to: (a) understand which models are capable of exhibiting systematic generalization as required by SQOOP, and (b) understand whether it is possible to induce, in an end-to-end way, the successful architectural decisions that lead to systematic generalization.

All models share the same stem architecture which consists of 6 layers of convolution (8 for Relation Networks), batch normalization and max pooling. The input to the stem is a 64 $\times$ 64 $\times$ 3 image, and the feature dimension used throughout the stem is 64. Further details can be found in Appendix C.1. The code for all experiments is available online[1].

8.4.1. Which Models Generalize Better?

We report the performance for all models on datasets of varying difficulty in Figure 8.4. Our first observation is that the modular and tree-structured NMN-Tree model exhibits strong systematic generalization. Both versions of this model, with Residual and Find modules, robustly solve all versions of our dataset, including the most challenging #rhs/lhs=1 split.

The results of NMN-Tree should be contrasted with those of generic models. 2 out of 4 models (Conv+LSTM and RelNet) are not able to learn to answer all SQOOP questions, no matter how easy the split was (for high #rhs/lhs Conv+LSTM overfitted and RelNet did not train). The results of other two models, MAC and FiLM, are similar. Both models are clearly able to solve the SQOOP task, as suggested by their almost perfect $< 1\%$ error rate on the control #rhs/lhs=35 split, yet they struggle to generalize on splits with lower #rhs/lhs. In particular, we observe $13.67 \pm 9.97\%$ errors for MAC and a $34.73 \pm 4.61\%$ errors for FiLM on the hardest #rhs/lhs=1 split. For the splits of intermediate difficulty we saw

[1] https://github.com/rizar/systematic-generalization-sqoop

the error rates of both models decreasing as we increased the #rhs/lhs ratio from 2 to 18. Interestingly, even with 18 #rhs/lhs some MAC and FiLM runs result in a test error rate of $\sim 2\%$. Given the simplicity and minimalism of SQOOP questions, we believe that these results should be considered a failure to pass the SQOOP test for both MAC and FiLM. That said, we note a difference in how exactly FiLM and MAC fail on #rhs/lhs=1: in several runs (3 out of 15) MAC exhibits a strong generalization performance ($\sim 0.5\%$ error rate), whereas in all runs of FiLM the error rate is about 30%. We examine the successful MAC models and find that they converge to a successful setting of the control attention weights, where specific MAC units consistently attend to the right questions words. In particular, MAC models that generalize strongly for each question seem to have a unit focusing strongly on X and a unit focusing strongly on Y (see Appendix C.2 for more details). As MAC was the strongest competitor of NMN-Tree across generic models, we perform an ablation study for this model, in which we vary the number of modules and hidden units, as well as experiment with weight decay. These modifications do not result in any significant reduction of the gap between MAC and NMN-Tree. Interestingly, we find that using the default high number of MAC units, namely 12, is helpful, possibly because it increases the likelihood that at least one unit converges to focus on X and Y words (see Appendix C.2 for details).

8.4.2. What is Essential to Strong Generalization of NMN?

The superior generalization of NMN-Tree raises the following question: what is the key architectural difference between NMN-Tree and generic models that explains the performance gap between them? We consider two candidate explanations. First, the NMN-Tree model differs from the generic models in that it does not use a language encoder and is instead built from modules that are parametrized by question words directly. Second, NMN-Tree is structured in a particular way, with the idea that modules 1 and 2 may learn to locate objects and module 3 can learn to reason about object locations independently of their identities. To understand which of the two differences is responsible for the superior generalization, we compare the performance of the NMN-Tree, NMN-Chain and NMN-Chain-Shortcut models (see Figure 8.1). These 3 versions of NMN are similar in that none of them are using a language encoder, but they differ in how the modules are connected. The results in Figure 8.4 show that for both Find and Residual module architectures, using a tree layout is absolutely crucial (and sufficient) for generalization, meaning that the generalization gap between NMN-Tree and generic models can not be explained merely by the language encoding step in the latter. In particular, NMN-Chain models perform barely above random chance, doing even worse than generic models on the #rhs/lhs=1 version of the dataset and dramatically failing even on the easiest #rhs/lhs=18 split. This is in stark contrast with NMN-Tree models that exhibits nearly perfect performance on the hardest #rhs/lhs=1 split. As a sanity

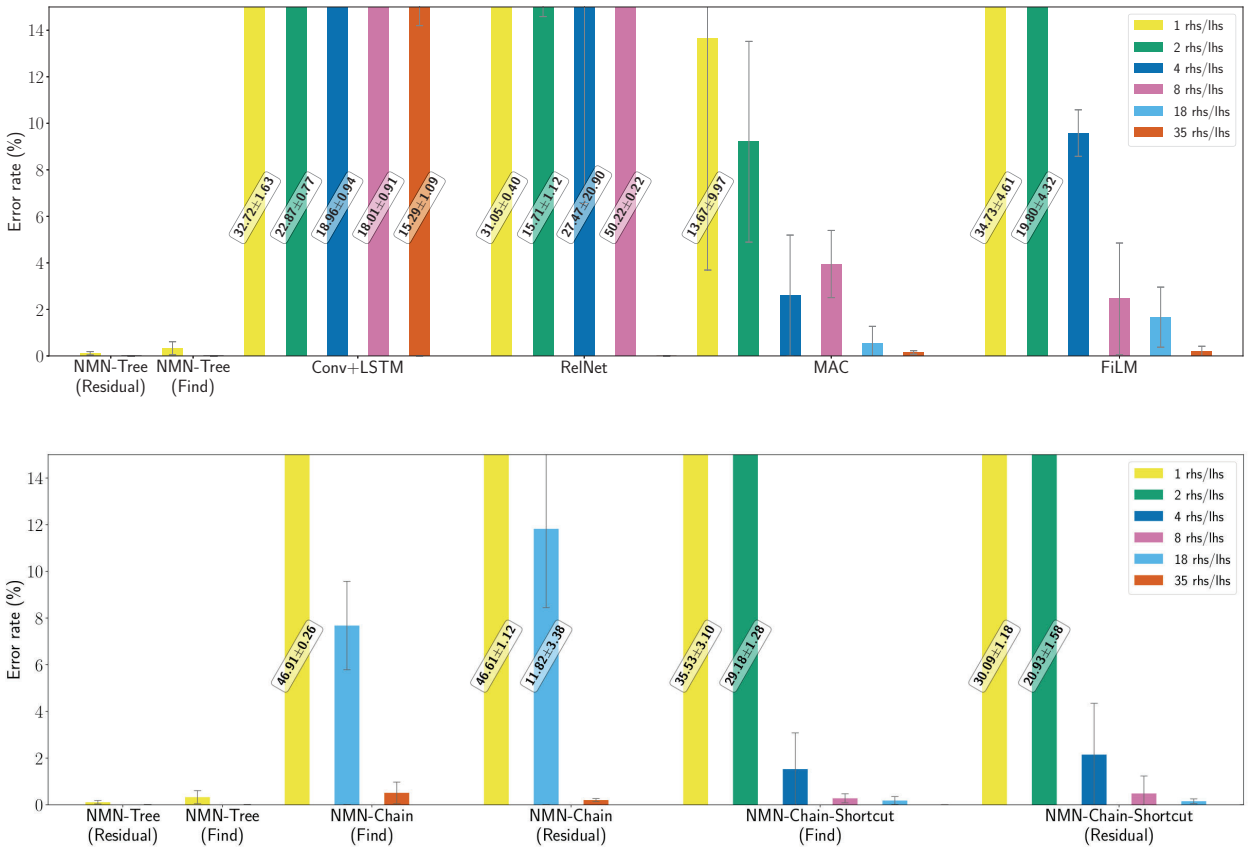


FIGURE 8.4. **Top:** Comparing the performance of generic models on datasets of varying difficulty (lower #rhs/lhs is more difficult). Note that NMN-Tree generalizes perfectly on the hardest #rhs/lhs=1 version of SQOOP, whereas MAC and FiLM fail to solve completely even the easiest #rhs/lhs=18 version. **Bottom:** Comparing NMNs with different layouts and modules. We can clearly observe the superior generalization of NMN-Tree, poor generalization of NMN-Chain and mediocre generalization of NMN-Chain-Shortcut. Means and standard deviations after at least 5 runs are reported.

check we train NMN-Chain models on the vanilla #rhs/lhs=35 split. We find that NMN-Chain has little difficulty learning to answer SQOOP questions when it sees all of them at training time, even though it previously shows poor generalization when testing on unseen examples. Interestingly, NMN-Chain-Shortcut performs much better than NMN-Chain and quite similarly to generic models. We find it remarkable that such a slight change in the model layout as adding shortcut connections from image features h_x to the modules results in a drastic change in generalization performance. In an attempt to understand why NMN-Chain generalizes so poorly we compare the test set responses of the 5 NMN-Chain models trained on #rhs/lhs=1 split. Notably, there was very little agreement between predictions of these 5 runs (Fleiss $\kappa = 0.05$), suggesting that NMN-Chain performs rather randomly outside of the training set.

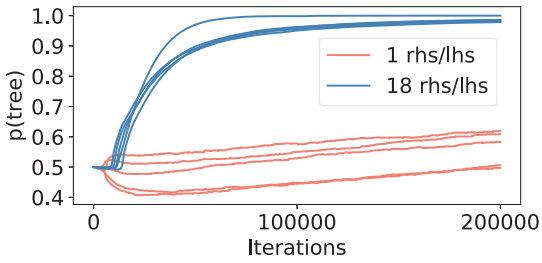


FIGURE 8.5. Learning dynamics of layout induction on 1 rhs/lhs and 18 rhs/lhs datasets using the Residual module with $p_0(tree) = 0.5$. All 5 runs do not learn to use the tree layout for 1 rhs/lhs, the very setting where the tree layout is necessary for generalization.

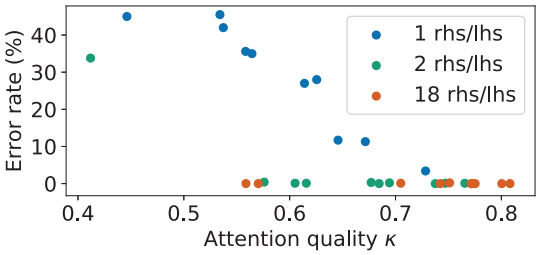


FIGURE 8.6. Attention quality κ vs accuracy for Attention N2NMN models trained on different #rhs/lhs splits. We can observe that generalization is strongly associated with high κ for #rhs/lhs=1, while for splits with 2 and 18 rhs/lhs blurry attention may be sufficient.

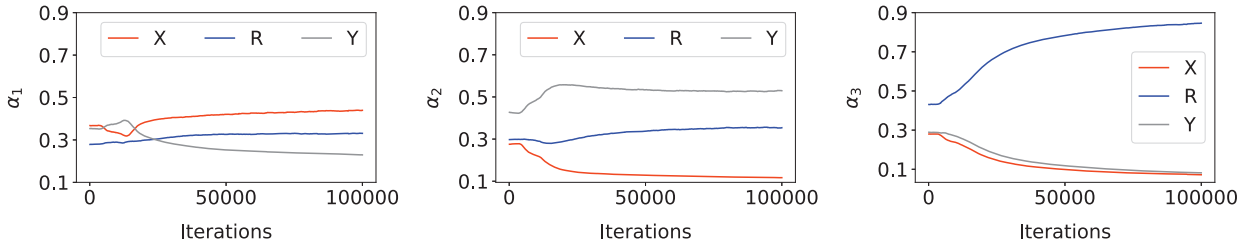


FIGURE 8.7. An example of how attention weights of modules 1 (**left**), 2 (**middle**), and 3 (**right**) evolve during training of an Attention N2NMN model on the 18 rhs/lhs version of SQOOP. Modules 1 and 2 learn to focus on different objects words, X and Y respectively in this example, but they also assign high weight to the relation word R. Module 3 learns to focus exclusively on R.

8.4.3. Can the Right Kind of NMN Be Induced?

The strong generalization of the NMN-Tree is impressive, but a significant amount of prior knowledge about the task was required to come up with the successful *layout* and *parametrization* used in this model. We therefore investigate whether the amount of such prior knowledge can be reduced by fixing one of these structural aspects and inducing the other.

8.4.3.1. *Layout Induction*

In our layout induction experiments, we use the Stochastic N2NMN model which treats the layout as a stochastic latent variable with two values (T_{tree} and T_{chain}, see Section 8.3.2 for details). We experiment with N2NMNs using both Find and Residual modules and

TABLE 8.1. Tree layout induction results for Stochastic N2NMNs using Residual and Find modules on 1 rhs/lhs and 18 rhs/lhs datasets. For each setting of $p_0(tree)$ we report results after 5 runs. $p_{200K}(tree)$ is the probability of using a tree layout after 200K training iterations.

module	#rhs/lhs	$p_0(tree)$	Test error rate (%)	Test loss	$p_{200K}(tree)$
Residual	1	0.1	31.89 ± 0.75	0.64 ± 0.03	0.08 ± 0.01
		0.5	1.64 ± 1.79	0.27 ± 0.04	0.56 ± 0.06
		0.9	0.16 ± 0.11	0.03 ± 0.01	0.96 ± 0.00
	18	0.1	3.99 ± 5.33	0.15 ± 0.06	0.59 ± 0.34
		0.5	0.19 ± 0.11	0.06 ± 0.02	0.99 ± 0.01
		0.9	0.12 ± 0.12	0.01 ± 0.00	1.00 ± 0.00
Find	1	0.1	47.54 ± 0.95	1.78 ± 0.47	0.00 ± 0.00
		0.5	0.78 ± 0.52	0.05 ± 0.04	0.94 ± 0.07
		0.9	0.41 ± 0.07	0.02 ± 0.00	1.00 ± 0.00
	18	0.1	5.11 ± 1.19	0.14 ± 0.03	0.02 ± 0.04
		0.5	0.17 ± 0.16	0.01 ± 0.01	1.00 ± 0.00
		0.9	0.11 ± 0.03	0.00 ± 0.00	1.00 ± 0.00

report results with different initial conditions, $p_0(tree) \in 0.1, 0.5, 0.9$. We believe that the initial probability $p_0(tree) = 0.1$ should not be considered small, since in more challenging datasets the space of layouts would be exponentially large, and sampling the right layout in 10% of all cases should be considered a very lucky initialization. We repeat all experiments on #rhs/lhs=1 and on #rhs/lhs=18 splits, the former to study generalization, and the latter to control whether the failures on #rhs/lhs=1 are caused specifically by the difficulty of this split. The results (see Table 8.1) show that the success of layout induction (i.e. converging to a $p(tree)$ close to 0.9) depends in a complex way on all the factors that we considered in our experiments. The initialization has the most influence: models initialized with $p_0(tree) = 0.1$ typically do not converge to a tree (exception being experiments with Residual module on #rhs/lhs=18, in which 3 out of 5 runs converged to a solution with a high $p(tree)$). Likewise, models initialized with $p_0(tree) = 0.9$ always stay in a regime with a high $p(tree)$. In the intermediate setting of $p_0(tree) = 0.5$ we observe differences in behaviors for Residual and Find modules. In particular, N2NMN based on Residual modules stays spurious with $p(tree) = 0.5 \pm 0.08$ when #rhs/lhs=1, whereas N2NMN based on Find modules always converges to a tree.

One counterintuitive result in Table 8.1 is that for the Stochastic N2NMNs with Residual modules, trained with $p_0(tree) = 0.5$ and #rhs/lhs=1, make just $1.64 \pm 1.79\%$ test error despite never resolving the layout uncertainty through training ($p_{200K}(tree) = 0.56 \pm 0.06$). We offer an investigation of this result in Appendix C.3.

8.4.3.2. *Parametrization Induction*

Next, we experiment with the Attention N2NMN model (see Section 8.3.2) in which the parametrization is learned for each module as an attention-weighted average of word embeddings. In these experiments, we fix the layout to be tree-like and sample the pre-softmax attention weights $\tilde{\alpha}$ from a uniform distribution $U[0; 1]$. As in the layout induction investigations, we experiment with several SQOOP splits, namely we try #rhs/lhs $\in \{1, 2, 18\}$. The results (reported in Table 8.2) show that Attention N2NMN fails dramatically on #rhs/lhs=1 but quickly catches up as soon as #rhs/lhs is increased to 2. Notably, 9 out of 10 runs on #rhs/lhs=2 result in almost perfect performance, and 1 run completely fails to generalize (26% error rate), resulting in a high 8.18% variance of the mean error rate. All 10 runs on the split with 18 rhs/lhs generalize flawlessly. Furthermore, we inspect the learned attention weights and find that for typical successful runs, module 3 focuses on the relation word, whereas modules 1 and 2 focus on different object words (see Figure 8.7) while still focusing on the relation word. To better understand the relationship between successful layout induction and generalization, we define an attention quality metric $\kappa = \min_{w \in \{X,Y\}} \max_{k \in 1,2} \alpha_{k,w}/(1 - \alpha_{k,R})$. Intuitively, κ is large when for each word $w \in X, Y$ there is a module i that focuses mostly on this word. The renormalization by $1/(1 - \alpha_{k,R})$ is necessary to factor out the amount of attention that modules 1 and 2 assign to the relation word. For the ground-truth parametrization that we use for NMN-Tree κ takes a value of 1, and if both modules 1 and 2 focus on X, completely ignoring Y, κ equals 0. The scatterplot of the test error rate versus κ (Figure 8.6) shows that for #rhs/lhs=1 high generalization is strongly associated with higher κ, meaning that it is indeed necessary to have different modules strongly focusing on different object words in order to generalize in this most challenging setting. Interestingly, for #rhs/lhs=2 we see a lot of cases where N2NMN generalizes well despite attention being rather spurious ($\kappa \approx 0.6$).

In order to put Attention N2NMN results in context we compare them to those of MAC (see Table 8.2). Such a comparison can be of interest because both models perform attention over the question. For 1 rhs/lhs MAC seems to be better on average, but as we increase #rhs/lhs to 2 we note that Attention N2NMN succeeds in 9 out of 10 cases on the #rhs/lhs=2 split, much more often than 1 success out of 10 observed for MAC[2]. This result suggests that Attention N2NMNs retains some of the strong generalization potential of NMNs with hard-coded parametrization.

[2]If we judge a run successful when the error rate is lower than $\tau = 1\%$, these success rates are different with a p-value of 0.001 according to the Fisher exact test. Same holds for any other threshold $\tau \in [1\%; 5\%]$.

TABLE 8.2. Parameterization induction results for 1,2,18 rhs/lhs datasets for Attention N2NMN. The model does not generalize well in the difficult 1 rhs/lhs setting. Results for MAC are presented for comparison. Means and standard deviations were estimated based on at least 10 runs.

Model	#rhs/lhs	Test error rate (%)	Test loss (%)
Attention N2NMN	1	27.19 ± 16.02	1.22 ± 0.71
Attention N2NMN	2	2.82 ± 8.18	0.14 ± 0.41
Attention N2NMN	18	0.16 ± 0.12	0.00 ± 0.00
MAC	1	13.67 ± 9.97	0.41 ± 0.32
MAC	2	9.21 ± 4.31	0.28 ± 0.15
MAC	18	0.53 ± 0.74	0.01 ± 0.02

8.5. Related Work

The notion of systematicity was originally introduced by (Fodor and Pylyshyn, 1988) as the property of human cognition whereby "the ability to entertain a given thought implies the ability to entertain thoughts with semantically related contents". They illustrate this with an example that no English speaker can understand the phrase "John loves the girl" without being also able to understand the phrase "the girl loves John". The question of whether or not connectionist models of cognition can account for the systematicity phenomenon has been a subject of a long debate in cognitive science (Fodor and Pylyshyn, 1988; Smolensky, 1987; Marcus, 1998, 2003; Calvo and Colunga, 2003). Recent research has shown that lack of systematicity in the generalization is still a concern for the modern seq2seq models (Lake and Baroni, 2018; Bastings et al., 2018; Loula et al., 2018). Our findings about the weak systematic generalization of generic VQA models corroborate the aforementioned seq2seq results. We also go beyond merely stating negative generalization results and showcase the high systematicity potential of adding explicit modularity and structure to modern deep learning models.

Besides the theoretical appeal of systematicity, our study is inspired by highly related prior evidence that when trained on downstream language understanding tasks, neural networks often generalize poorly and latch on to dataset-specific regularities. Agrawal et al. (2016) report how neural models exploit biases in a VQA dataset, e.g. responding "snow" to the question "what covers the ground" regardless of the image because "snow" is the most common answer to this question. Gururangan et al. (2018) report that many successes in natural language entailment are actually due to exploiting statistical biases as opposed to solving entailment, and that state-of-the-art systems are much less performant when tested on unbiased data. Jia and Liang (2017) demonstrate that seemingly state-of-the-art reading comprehension system can be misled by simply appending an unrelated sentence that resembles the question to the document.

Using synthetic VQA datasets to study grounded language understanding is a recent trend started by the CLEVR dataset (Johnson et al., 2016). CLEVR images are 3D-rendered and CLEVR questions are longer and more complex than ours, but in the associated generalization split CLEVR-CoGenT the training and test distributions of images are different. In our design of SQOOP we aimed instead to minimize the difference between training and test images to make sure that we test a model's ability to interpret unknown combinations of known words. The ShapeWorld family of datasets by Kuhnle and Copestake (2017) is another synthetic VQA platform with a number of generalization tests, but none of them tests SQOOP-style generalization of relational reasoning to unseen object pairs. Most closely related to our work is the recent study of generalization to long-tail questions about rare objects done by Bingham et al. (2017). They do not, however, consider as many models as we do and do not study the question of whether the best-performing models can be made end-to-end.

The key paradigm that we test in our experiments is Neural Module Networks (NMN). Andreas et al. (2016) introduced NMNs as a modular, structured VQA model where a fixed number of hand-crafted neural modules (such as `Find`, or `Compare`) are chosen and composed together in a layout determined by the dependency parse of the question. Andreas et al. (2016) show that the modular structure allows answering questions that are longer than the training ones, a kind of generalization that is complementary to the one we study here. Hu et al. (2017) and Johnson et al. (2017) followed up by making NMNs end-to-end, removing the non-differentiable parser. Both Hu et al. (2017) and Johnson et al. (2017) reported that several thousands of ground-truth layouts are required to pretrain the layout predictor in order for their approaches to work. In a recent work, Hu et al. (2018) attempt to soften the layout decisions, but training their models end-to-end from scratch performed substantially lower than best models on the CLEVR task. Gupta and Lewis (2018) report successful layout induction on CLEVR for a carefully engineered heterogeneous NMN that takes a scene graph as opposed to a raw image as the input.

8.6. Conclusion and Discussion

We have conducted a rigorous investigation of an important form of systematic generalization required for grounded language understanding: the ability to reason about all possible pairs of objects despite being trained on a small subset of such pairs. Our results allow one to draw two important conclusions. For one, the intuitive appeal of modularity and structure in designing neural architectures for language understanding is now supported by our results, which show how a modular model consisting of general purpose residual blocks generalizes much better than a number of baselines, including architectures such as MAC, FiLM and RelNet that were designed specifically for visual reasoning. While this may seem unsurprising, to the best of our knowledge, the literature has lacked such a clear empirical

evidence in favor of modular and structured networks before this work. Importantly, we have also shown how sensitive the high performance of the modular models is to the layout of modules, and how a tree-like structure generalizes much stronger than a typical chain of layers.

Our second key conclusion is that coming up with an end-to-end and/or soft version of modular models may be not sufficient for strong generalization. In the very setting where strong generalization is required, end-to-end methods often converge to a different, less compositional solution (e.g. a chain layout or blurred attention). This can be observed especially clearly in our NMN layout and parametrization induction experiments on the #rhs/lhs=1 version of SQOOP, but notably, strong initialization sensitivity of layout induction remains an issue even on the #rhs/lhs=18 split. This conclusion is relevant in the view of recent work in the direction of making NMNs more end-to-end (Suarez et al., 2018; Hu et al., 2018; Hudson and Manning, 2018; Gupta and Lewis, 2018). Our findings suggest that merely replacing hard-coded components with learnable counterparts can be insufficient, and that research on regularizers or priors that steer the learning towards more systematic solutions can be required. That said, our parametrization induction results on the #rhs/lhs=2 split are encouraging, as they show that compared to generic models, a weaker nudge (in the form of a richer training signal or a prior) towards systematicity may suffice for end-to-end NMNs.

While our investigation has been performed on a synthetic dataset, we believe that it is the real-world language understanding where our findings may be most relevant. It is possible to construct a synthetic dataset that is bias-free and that can only be solved if the model has understood the entirety of the dataset's language. It is, on the contrary, much harder to collect real-world datasets that do not permit highly dataset-specific solutions, as numerous dataset analysis papers of recent years have shown (see Section 8.5 for a review). We believe that approaches that can generalize strongly from imperfect and biased data will likely be required, and our experiments can be seen as a simulation of such a scenario. We hope, therefore, that our findings will inform researchers working on language understanding and provide them with a useful intuition about what facilitates strong generalization and what is likely to inhibit it.

Chapter 9

PROLOGUE TO FOURTH ARTICLE

9.1. Article Details

CLOSURE: Assessing Systematic Generalization of CLEVR Models. Dzmitry Bahdanau, Harm de Vries, Timothy J. O'Donnell, Shikhar Murty, Philippe Beaudoin, Yoshua Bengio, and Aaron Courville. In preparation.

Personal Contribution. I did most of the work, including building the dataset, implementing all models, running all experiments, and writing most of the paper. Other authors mostly advised and helped with writing. In particular, Harm de Vries's and Timothy O'Donnell's help with writing was very substantial.

9.2. Context

The project had two goals. One was to complement the study of systematic generalization that is presented in Chapter 8 by an investigation that uses a more diverse synthetic dataset and focuses on different linguistic phenomena. Another goal was to contextualize the impressive results on the CLEVR dataset that had been reported for a number of models. We suspected that a key aspect of these results is the fact that the same templates are used at training and test time in the usual CLEVR setup and designed our experiments to test this hypothesis.

9.3. Contributions

We analyze and compare generalization abilities of modern VQA models and show that all of them experience generalization issues when tested on CLEVR-like questions that are out of the original data distribution. We propose a new Vector-NMN module with which the NMNs generalize better. Lastly, our few-shot transfer learning studies highlight different adaptation behaviors of models with and without internal programs.

Chapter 10

CLOSURE: ASSESSING SYSTEMATIC GENERALIZATION OF CLEVR MODELS

10.1. Introduction

The ability to communicate in natural language and ground it effectively into our rich unstructured 3D reality is a crucial skill that we expect from artificial agents of the future. A popular task to benchmark progress towards this goal is *Visual Question Answering* (VQA), in which one must give a (typically short) answer to a question about the content of an image. The release of the relatively large VQA 1.0 dataset by Antol et al. (2015) ignited the interest for the VQA setup, but researchers soon found that the biases of natural data (such as the heavily skewed answer distribution for certain question types) make it hard to interpret the VQA 1.0 results (Agrawal et al., 2016). To complement biased natural data, Johnson et al. (2016) constructed the CLEVR dataset of complex synthetic questions about 3D-rendered scenes to be free of such biases (see Q1 and Q2 in Figure 10.1 for examples of CLEVR questions). The CLEVR dataset has spurred VQA modeling research, and many models were designed for it and showcased using it (Santoro et al., 2017; Perez et al., 2017; Johnson et al., 2017; Hudson and Manning, 2018; Mascharka et al., 2018).

The high complexity and diversity of CLEVR questions and the reported 97-99% accuracies may lead to the impression that these high-performing models are capable of answering any possible question that uses the same linguistic constructs as in CLEVR. Such an intuitive expectation corresponds to the concept of *systematicity* (Fodor and Pylyshyn, 1988), which characterizes the ability of humans to interpret arbitrary combinations of known primitives. One can argue that systematicity is also a highly desirable property for AI systems. For example, suppose you refer to an object by relating its appearance to another object, as in "the object that is the same size as the red cube". If a CLEVR-trained model understands such a referring expression, it is likely that you will expect this model to understand it in other, more complex contexts. This includes cases in which such an expression is embedded in a more complex referring expression, e.g. "the cylinder to the left of the object that is

Q1 (CLEVR): There is another cube that is the same size as the brown cube; what is its color?
Q2 (CLEVR): There is a thing that is in front of the yellow thing; does it have the same color as cylinder?
Q3 (CLOSURE): There is another rubber object that is the same size as the gray cylinder; does it have the same color as the tiny shiny block?

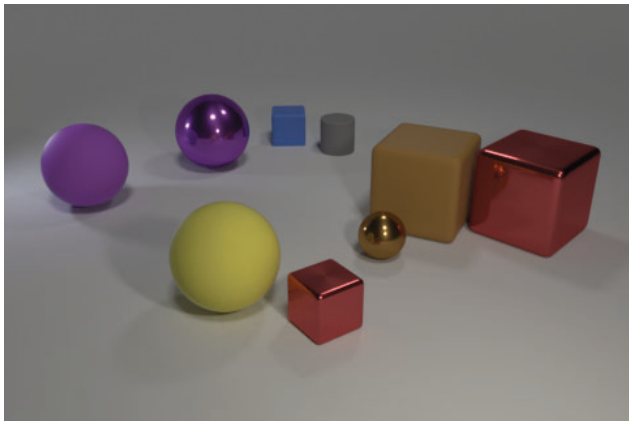


FIGURE 10.1. CLEVR questions (Q1 and Q2) require complex multi-step reasoning about the contents of 3D-rendered images. We construct CLOSURE questions (Q3) by using the referring expressions that rely on matching object properties (e.g. the red fragment in Q1) in novel contexts, such as e.g. comparison questions with two referring expressions (Q2).

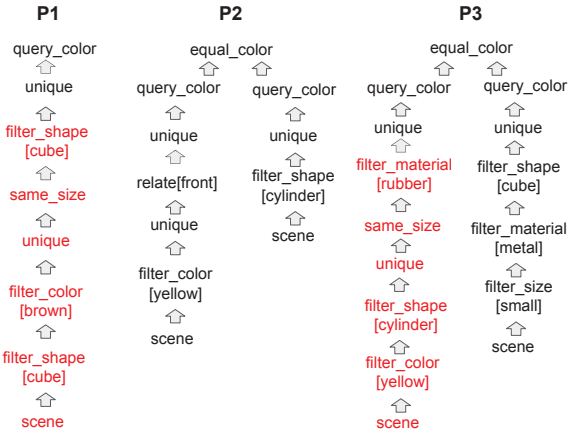


FIGURE 10.2. Programs P1, P2, P3 that define the ground-truth meaning for the questions Q1, Q2 and Q3 in Figure 10.1. The fragments in red correspond to the matching REs in the respective questions.

the same size as the red cube", or is logically combined with another expression, e.g. "either cubes or objects that are the same size as the red cube". For learning-based systems, unless the training distribution uniformly covers all sensible compositions of interest (which is nearly impossible to achieve for natural data), systematicity requires a particular kind of out-of-domain generalization, whereby the test distribution is different from the training one but follows the same rules of semantic and syntactic composition. Such *systematic generalization* of modern neural models has been recently studied in the context of artificial sequence transduction and VQA tasks (Lake and Baroni, 2018; Bahdanau et al., 2019b), the latter done in a setup that is much simpler and less diverse than CLEVR.

In this work, we perform a case-study of how systematic CLEVR-trained models are in their generalization capabilities. In doing so, we seek to provide important context to the near-perfect CLEVR accuracies that are measured by the usual methodology, as well as to contribute to the literature on systematic generalization. The specific aspect of systematicity that we analyze is the one exemplified in the previous paragraph: the ability to interpret known ways of referring to objects in arbitrary contexts. We focus on the *matching referring expressions* (see e.g. "another cube that it is the same size as the brown cube" in Figure 10.1) that require the object (or objects) to match another object in terms of a property, such as the size, the color, the material or the shape. We construct 7 CLOSURE tests with questions that highly overlap with the CLEVR ones and yet are different (see the Q3 in Figure 10.1 for an example from one such test), aiming to cover the contexts in which property matching is not used to refer to objects in the original CLEVR. We call the resulting benchmark CLOSURE referring to the underlying idea of taking a closure (in the mathematical sense) of CLEVR questions under the operation of referring expression substitution and keeping those questions that are similar enough to the original ones.

We evaluate a number of different CLEVR-trained models on CLOSURE, including end-to-end differentiable ones, like FiLM (Perez et al., 2017) and MAC (Hudson and Manning, 2018), and models using intermediate symbolic programs, like NS-VQA (Yi et al., 2018) and the variety of Neural Module Networks (NMN, Andreas et al. (2016)) proposed by Johnson et al. (2017). We show that all aforementioned models often struggle on CLOSURE questions. For models using symbolic programs, such as NS-VQA and NMN, we observe a generalization gap in the performance of their neural sequence-to-sequence program generators (Sutskever et al., 2014; Bahdanau et al., 2015). Furthermore, NMNs often exhibit poor generalization even when the ground-truth programs are provided. This result is remarkable given that the original motivation for NMNs is to decompose the model into components that can be recombined arbitrarily. To improve the NMN's generalization, we develop a new Vector-NMN module with vector-valued (as opposed to tensor-valued) inputs and outputs. We show that the Vector-NMN modules perform much better than prior work when assembled in configurations that are different from the training ones. Lastly, we complement our 0-shot systematic generalization analysis with a few-shot transfer learning study and contrast the few-shot adaptation behavior of models with and without symbolic programs.

10.2. CLOSURE: A Systematic Generalization Benchmark for CLEVR

Analysis of CLEVR

The key source of diversity and complexity in CLEVR questions is how objects of interest are referred to. We call a noun phrase a *referring expression* (RE) when it refers to an object

or a set of objects, themselves called *referents*. We distinguish three kinds of REs that occur in CLEVR: simple REs, complex REs and logical REs. A *simple RE* is a noun that is (optionally) modified by one or more adjective, e.g.:

$$\text{the big red cube} \tag{10.2.1}$$

$$\text{yellow shiny spheres.} \tag{10.2.2}$$

In *Complex REs*, a relative clause (in square brackets in the examples below) is used to modify the noun (possibly in addition to adjectives):

$$\text{the cube [that is left of } \langle RE \rangle\text{]}, \tag{10.2.3}$$

$$\text{big spheres [that are the same color as } \langle RE \rangle\text{]}. \tag{10.2.4}$$

In examples above, $\langle RE \rangle$ is the *embedded RE*, which can be either simple or also complex. Complex REs in CLEVR can be *spatial* (Example 10.2.3) or rely on matching objects' properties (Example 10.2.4). We will call the latter *matching REs*. The RE's type is determined by whether a *spatial predicate* ("is left of $\langle RE \rangle$", "is right of $\langle RE \rangle$", ...) or a *matching predicate* ("is the same size as $\langle RE \rangle$", "is the same color as $\langle RE \rangle$", ...) is used to construct the relative clause.[1] In *Logical REs*, two REs (Example 10.2.5, square brackets) or two prepositional phrases (Example 10.2.6, square brackets) are combined using "and" or "or":

$$\text{[tiny balls] or [brown blocks behind the matte object]}, \tag{10.2.5}$$

$$\text{a metallic object that is [left of the brown ball]}$$
$$\text{and [in front of the tiny block]} \tag{10.2.6}$$

The second most important axis of variation in CLEVR is what kind of question is asked about the referents. CLEVR includes existence, counting, attribute and object comparison questions, see examples below:

- (existence) Is there a big cyan object?
- (counting) How many purple things are behind the cylinder?
- (attribute) What material is the big purple ball?
- (comparison) Do the red thing and the big thing have the same shape?

In existence, counting and attribute questions there is one top-level RE, whereas comparison questions contain two top-level REs.

CLOSURE questions

We have constructed the CLOSURE dataset by generating new CLEVR-like questions with matching predicates. To this end, we analyzed the composition of CLEVR and found

[1]Note that the original paper by (Johnson et al., 2016) these are called "spatial relationships" and "same-attribute relationships"

a number of question templates in which a spatial predicate could be seamlessly substituted for a matching one. We focused on 7 cases where such substitution was possible and where it yielded questions that were not possible under CLEVR's original data distribution. Below, we describe and give examples of each of the resulting 7 CLOSURE tests. For a more technical explanation of the question generation procedure we refer the reader to Section 10.7.

The **embed_spa_mat** test contains existence questions with a matching RE that has an embedded spatial RE, e.g:

- Is there a cylinder that is the same material as the object to the left of the blue thing?

Here, a spatial RE "the object to the left of the blue thing" is embedded in a matching RE "a cylinder that is the same material ...". Note, that in original CLEVR matching REs can only contain simple embedded REs. A closely related test is **embed_mat_spa**, in which the top-level RE is spatial and the embedded one uses property matching:

- Is there a thing behind the cube that is the same color as the ball?

In **compare_mat** and **compare_mat_spa** tests we focus on models' ability to understand matching REs in comparison questions:

- There is another small cylinder that is the same material as the small cyan cylinder; does it have the same color as the block?
- There is another cube that is the same material as the gray cube; does it have the same size as the metal thing to the right of the tiny gray cube?

The comparison questions in CLEVR only use spatial REs, hence `compare_mat` and `compare_mat_spa` require models to recombine known constructs (that is the matching REs and the comparison questions) in a novel way. The two tests differ in whether the second RE is simple (`compare_mat`) or spatial (`compare_mat_spa`).

The remaining three CLOSURE tests assess models' understanding of matching predicates in logical REs. The **or_mat** and **or_mat_spa** questions require counting referents for a logical "or" of two REs, one of which uses property matching:

- How many things are cubes or cylinders that are the same size as the red object?
- How many things are objects that are in front of the blue thing or small metallic things that are the same color as the rubber block?

The `or_mat_spa` test differs from the `or_mat` one in that the second RE is also a complex one. The **and_mat_spa** test contains attribute questions in which the RE involves a logical "and" of a spatial and a matching predicate:

- What is the color of the thing that is to the left of the red cylinder and is the same size as the red block?

All the three tests presented above contain questions that are impossible under CLEVR's original data distribution, as logical REs in CLEVR only employ spatial predicates.

CLOSURE templates and programs

To construct CLOSURE questions and to compute the ground-truth answers we used the template-based question generator that comes with CLEVR. The question generator randomly fills the slots of a given template to produce new questions as well as symbolic programs in a functional domain-specific language (DSL) that represent questions' meanings. See Figure 10.2 for example programs P1 and P2. The programs are executed against a symbolic scene graph to produce the ground-truth answers for all questions. Another usecase for groundtruth programs is to bootstrap learning in models that internally use programs as representations of questions' meanings, such e.g. Neural Module Networks (NMN). Here, we explain how differences between CLOSURE and CLEVR questions manifest themselves in their respective ground-truth programs.

The DSL functions that implement meanings of referring expressions operate on sets of objects, where each object is represented by the values of its four properties (shape, color, size and material) and its spatial coordinates. *Filter* functions filter the input set of objects by the value of a property (e.g. "filter_color[brown]", "filter_shape[cube]"), and *relations* return a set of all other objects that are related to the given object. The two kinds of relations in the DSL correspond to the spatial and matching predicates that we discussed above. Namely, there are spatial relations: ("relate[left]", "relate[right]", ...) and matching relations ("same_shape", "same_color", ...). The subprograms that correspond to the REs consist of chained filters and relations, as well as set union and set intersection functions in the case of logical REs[2]. The wider use of matching predicates that distinguishes CLOSURE questions from CLEVR ones translates into matching relations appearing in more diverse kinds of programs than in CLEVR. For example, in program P3 in Figure 10.2 the relation "same_size" appears in the same program with the function "equal_color", a combination that would not be possible in CLEVR. Hence, to succeed on CLOSURE, the NMN models have to learn modules which can be arbitrarily recombined with each other.

Dataset statistics

Our public dataset release[3] includes:

- a validation test of 3600 questions each CLOSURE test,
- a test set of the same size,
- a small training set of 36 questions per CLOSURE test for few-shot learning investigations.

The validation and test sets contain questions about different validation images from CLEVR (we could not generate new questions for CLEVR test images as the corresponding scene

[2]The "unique" function also frequently appears in the subprograms that correspond to REs. Its only role is to raise an exception if its input is not a singleton set

[3]https://github.com/rizar/CLOSURE

graphs are not available). The training set questions are asked about training images in CLEVR.

10.3. Models

A large number of models for the CLEVR task have been recently proposed, and it would be impossible for us to evaluate all of them. We therefore choose several models that vary in how CLEVR-specific their design is, aiming to cover the whole spectrum of "CLEVR-awareness" that such models possess. In addition to existing models, we experiment with a novel Vector-NMN neural module that we employ in the context of the Neural Module Network paradigm.

Throughout this section we use capital letters for matrix- or tensor-shaped parameters of all models and small letters for the vector-valued ones. We use $*$ to denote convolution as well as to inform the reader that the symbols on the left and right sides of the operator are a 4D and a 3D tensor respectively. $\odot$ and $\oplus$ are used to denote feature-wise multiplication and addition for the case where one argument is a 3D tensor and another is a vector. The respective operation is applied independently to all sub-vectors of the tensor-valued argument obtained by fixing its first two indices (the approach known as "broadcasting"). We will use square brackets $[x; y]$ to denote tensor concatenation performed along the last dimension.

10.3.1. Generic Models

The most generic method that we consider is Feature-wise Linear Modulation (**FiLM**) by Perez et al. (2017). In this approach, an LSTM recurrent network transforms the question q into biases β and element-wise multipliers α that are then applied in the blocks of a deep residual convolutional network (He et al., 2016). A FiLM-ed residual block takes a tensor-valued input h_{in} and performs the following computation upon it:

$$[\gamma; \beta] = W \cdot LSTM(q) + b, \tag{10.3.1}$$

$$\tilde{h} = BN(W_2 * R(W_1 * h_{in} \oplus b_1)), \tag{10.3.2}$$

$$h_{out} = h_{in} + R(\gamma \odot \tilde{h} \oplus \beta), \tag{10.3.3}$$

where R stands for the Rectified Linear Unit, BN denotes batch normalization (Ioffe and Szegedy, 2015). Several such blocks are stacked together and applied to a 3D feature tensor h_x that is produced by several layers of convolutions, some of them pretrained. The FiLM-ed network thus processes the input image x in a manner that is modulated by the question q. Despite its simplicity, FiLM achieves a remarkably high reported accuracy of 97.7% on the CLEVR task.

A more advanced model that we include in our evaluation is Memory-Attention-Composition (**MAC**) by Hudson and Manning (2018). In the MAC approach, the input

and control components of the model first produce a sequence of control vectors c_i from the question q. A visual attention component (called the read unit in the original paper) is then recurrently applied to a preprocessed version h_x of the image x. The i-th application of the read unit is conditioned on the respective control vector c_i and on a memory m_i of the unit's outputs at the previous steps:

$$r_i = read\_unit(h_x, c_i, m_{i-1}), \tag{10.3.4}$$

$$m_i = memory\_unit(r_i, m_{i-1}). \tag{10.3.5}$$

Such read operations and memory updates are performed for T steps, after which the last memory vector m_T and a question representation q are concatenated and passed to the classifier. Different versions of the MAC model reach near-perfect 98.9-99.4% performance on CLEVR.

10.3.2. Modular and Symbolic Approaches

In addition to the end-to-end differentiable models, we experiment with methods that rely on intermediate structured symbolic meaning representations. We adhere to the common practice of using programs expressed in the CLEVR DSL as such representations, although in principle logical formulae or other formalisms from the field of formal semantics could be used for this purpose. Under the assumption that a symbolic *execution engine* for the programs is available, the task of VQA can be reduced to parsing the question and the image into a program and a symbolic scene representation respectively. Such an approach has been proposed by Yi et al. (2018) under the name Neural-Symbolic VQA (**NS-VQA**) with a reported CLEVR accuracy of 99.8%. This excellent performance, however, is achieved by relying heavily on the prior knowledge about the task, meaning that applying NS-VQA in conditions other than CLEVR could require significantly more adaptation and data collection than needed for the more generic methods, such as FiLM and MAC.

Intermediate symbolic programs can also be used without apriori knowledge of the semantics of the symbols, in which case the execution engine for the programs is either fully or partially learned. In the Neural Module Network (**NMN**) paradigm, proposed by Andreas et al. (2016), the meanings of symbols are represented in the form of trainable neural modules. Given a program, the modules that correspond to the program's symbols are retrieved and composed following the program's structure. Formally, a program in CLEVR DSL can be represented as a (P, L, R) triple, where $P = (p_1, p_2, \ldots, p_T)$ is the sequence of function tokens[4], $L = (l_1, \ldots, l_T)$ and $R = (r_1, \ldots, r_T)$ are the indices of the left and right arguments for each function call respectively (some DSL functions only take one or zero arguments, in which case the respective r_i and l_i are undefined). Using this formalism, a step of the NMN

[4]In this work we treat composite functions like e.g. "filter_color[brown]" as standalone ones, not as "filter_color" parameterized by "brown"

computation can be expressed as follows:

$$h_i = \begin{cases} M_{p_i}(h_x), & arity(p_i) = 0, \\ M_{p_i}(h_x, h_{l_i}), & arity(p_i) = 1, \\ M_{p_i}(h_x, h_{l_i}, h_{r_i}), & arity(p_i) = 2. \end{cases} \tag{10.3.6}$$

Here, M_{p_i} is the neural module corresponding to the function token p_i and h_i is its output, while h_x is a tensor representing the image. Similar to MAC, the output h_T of the last module is fed to the classifier, after which the modules are jointly trained by backpropagating the classifier's loss.

A number of NMN-based approaches have been proposed for the CLEVR task, including those where different modules perform different operations (e.g. the module corresponding to logical "and" might compute an element-wise maximum of two vectors (Hu et al., 2017; Mascharka et al., 2018)), and those where all modules perform similar computations but use different parameters (Johnson et al., 2017). We focus on the latter variety of NMNs, since such models rely less on the domain knowledge and thus complement well the NS-VQA approach in our evaluation. In both cases, a *program generator* can be pretrained with a small seed set of (question, program)-pairs and then fine-tuned, e.g. with REINFORCE (Hu et al., 2017; Johnson et al., 2017), on the rest of the dataset, using only (image, question, answer)-triplets as supervision. The programs produced by such a program generator can then be used at test time, meaning that after training, the complete model takes the same inputs as end-to-end continuous models, such as FiLM and MAC.

The first NMN model that we consider is the one proposed by Johnson et al. (2017), in which residual blocks (He et al., 2016) are used as neural modules M_{p_i}. For example, modules corresponding to functions of arity 2, (such as e.g. "and", "equal_color", etc.), perform the following computation in their approach:

$$h_{proj} = R(W_1 * [h_{l_i}; h_{r_i}]), \tag{10.3.7}$$

$$\tilde{h} = R(W_2 * h_{proj} \oplus b_2), \tag{10.3.8}$$

$$h_i = R(W_3 * \tilde{h} \oplus b_3) + h_{proj}. \tag{10.3.9}$$

Note that the module described above does not use the image representation h_x as an input; only the M_{scene} module—the root node in all CLEVR programs—does so.

Our preliminary experiments showed that such modules often perform much worse when assembled in novel combinations. We hypothesized that this could be due to the fact that high-capacity 3D tensors h_i are used in this model as the interface between modules. In order to test this hypothesis, we have designed a new module with a lower-dimensional vector output. We will henceforth refer to the module by Johnson et al. (2017) and our new module as **Tensor-NMN** and **Vector-NMN** respectively. The computation of our

Vector-NMN is inspired by the FiLM approach to conditioning residual blocks on external inputs:

$$\tilde{h}_1 = R(U_1 * (\gamma_1 \odot h_x \oplus \beta_1)), \tag{10.3.10}$$

$$\tilde{h}_2 = R(U_2 * (\gamma_2 \odot \tilde{h}_1 \oplus \beta_2) + h_x), \tag{10.3.11}$$

$$h_i = \text{maxpool}(\tilde{h}_2), \tag{10.3.12}$$

where "maxpool" denotes max pooling of the 3D-tensor across all locations. Note that each Vector-NMN module also takes the image feature tensor h_x as the input, unlike Tensor-NMN. The above equations describe a 1-block version of Vector-NMN, but in general several FiLM-ed residual blocks described by Equations 10.3.10 and 10.3.11 can be stacked prior to the max-pooling. The FiLM coefficients β_1, β_2, γ_1 and γ_2 are computed with 1-hidden-layer MLPs from the concatenation $h_{cond} = [e(p_i); h_{l_i}; h_{r_i}]$ of the embedding $e(p_i)$ of the function token p_i and the module inputs h_{l_i} and h_{r_i}

$$[\beta_k, \tilde{\gamma_k}] = W_2^k(R(W_1^k h_{cond} + b_1^k) + b_2^k), \tag{10.3.13}$$

$$\gamma_k = 2\tanh(\tilde{\gamma}_k) + 1. \tag{10.3.14}$$

The extra tanh nonlinearity in Equation 10.3.14 was required to achieve stable training. Note that unlike in Tensor-NMN, the convolutional filters U_1 and U_2 are reused among all modules. To make this possible, we feed zero vectors instead of h_{r_i} or h_{l_i} when the function p_i takes less than two inputs.

10.4. Experiments

We use the original implementation for FiLM and Tensor-NMN and train these models with the hyperparameter settings suggested by the authors. For the MAC model, we use a PyTorch reimplementation by Bahdanau et al. (2019b) that is close to the original one. We report results for a 2-block version of Vector-NMN, as our preliminary experiments on the original CLEVR dataset showed that it performs better than the 1-block version.

For all models that rely on symbolic programs, i.e. NS-VQA and the NMNs, we use a standard seq2seq model with an attention mechanism (Bahdanau et al., 2015) as the program generator. Our preliminary investigations showed that this model generalizes better than the seq2seq models without attention (Sutskever et al., 2014; Cho et al., 2014), as used in (Johnson et al., 2017), and better than the seq2seq model used in the reference implementation of NS-VQA, in which the decoder does not take the attention outputs as inputs. We report results for program generators trained with supervised learning on the

(question, program)-pairs from all 700K CLEVR examples[5]. In addition to evaluating the NMNs with the predicted programs, we also measure their performance when the ground-truth programs are given at test time. For the latter setting, we prepend **GT** to the model's name (as in GT-Vector-NMN), as opposed to prepending **PG** (as in PG-Vector-NMN).

All numbers that we report are averages over 5 or 10 runs. Where relevant, we also report the standard deviation σ in the form of $\pm\sigma$ or as a vertical black bar in figures.

10.4.1. Zero-shot Generalization

In our first set of experiments, we assess zero-shot systematic generalization of models trained on CLEVR by measuring their performance on the CLOSURE tests. To put these results in context, for each model and test we measure the model's performance on validation questions from CLEVR that are most similar to the given test's questions[6]. For example, consider the `embed_spa_mat` questions in which a spatial RE is embedded in a matching RE, such as "Is there a big blue metal thing that is the same shape as the rubber object behind the blue shiny object?". To establish a baseline for this test, we used existence questions where a spatial RE was embedded in another spatial RE, such as "Is there a big blue metal thing that is behind the rubber object behind the blue shiny object?" (notably, all models were accurate on $> 98\%$ of such questions). The gap between the model's performance on baseline questions and the model's performance on a CLOSURE test is indicative of how systematic the model's generalization behavior is. Indeed, the test- and model- dependent baseline scheme described above allows us to focus on the impact of combining known constructs in novel ways while controlling for other factors that influence the models' performance. For example, CLEVR-trained models typically perform worse on counting questions than on other question types, and hence we should expect lower accuracies on the `or_mat` and `or_mat_spa` tests that require counting.

The results are reported in Figure 10.3. One can clearly see that most models perform significantly worse on most CLOSURE tests compared to their accuracies on the respective baseline questions. A notable exception is `embed_spa_mat`, on which all models perform quite well. Among generic models, MAC consistently fares better than FiLM, albeit the former still loses 15% to 35% of its baseline accuracy in 6 out of 7 tests. Surprisingly, the NS-VQA model, whose only learnable component is a program generator, generalizes outright badly on tests that involved logical references, and also shows significant deterioration compared to baseline questions on other tests. This lack of systematic generalization in program generation also strongly affects performances of the two NMN models that we considered (PG-Vector-NMN

[5]Similarly to prior work we found that pretraining on as few as 300-1000 ground-truth programs, followed by REINFORCE finetuning, is sufficient to achieve near-perfect program generation performance. We chose, however, to use all available data to keep the study focused on systematic generalization.

[6]For `or_mat` and or_mat_loc we had to generate new CLEVR questions to compute baseline performance, see Section 10.7 for details.

and PG-Tensor-NMN). Interestingly, even given the ground-truth programs at test-time, Tensor-NMN modules perform worse on CLOSURE questions than on the respective baseline questions in 6 tests out of 7 (see GT-Tensor-NMN results). In contrast, our proposed Vector-NMN module generalizes much better, matching its baseline performance almost always, with a notable exception of the `and_mat_spa` test.

10.4.2. Few-shot transfer learning

The results above show that inductive biases of existing models are often insufficient for 0-shot systematic generalization measured by CLOSURE. A natural question to ask in these circumstances is whether just a few examples would be sufficient to correct the models' extrapolation behavior. To answer this question, we finetune MAC, PG-Vector-NMN and NS-VQA models that are pretrained on CLEVR using 36 examples from each CLOSURE family, for a total of 252 new examples. For PG-Vector-NMN and NS-VQA we consider two fine-tuning scenarios: one where the programs are provided for new examples and one where they are not given and must be inferred. To infer programs, we use a basic REINFORCE-based program induction approach (Johnson et al., 2017; Hu et al., 2017). We will refer to the two said scenarios as strong and weak supervision respectively. To get the best finetuning performance, we oversample 300 times the 252 training CLOSURE examples, add them to the CLEVR training set and train on the resulting mixed dataset. Just like in 0-shot experiments, we consider the model's performance on the closest CLEVR questions as the systematic generalization target.

The few-shot results, reported in Figure 10.4, show that as few as 36 examples from each family can significantly improve CLOSURE performance for all models. A notable exception from this general observation is the and_sim question family, on which weakly supervised program induction for NS-VQA and PG-Vector-NMN most often did not work. We analyzed this case in detail and found that the appropriate programs for and_sim would typically have a very low probability, and hence were never sampled in our REINFORCE-based program search.

A closer analysis reveals that the impact of 36 examples varies widely depending on the model and on the test. The models using symbolic programs reached the target performance in 6 tests out of 7. On the contrary, for MAC a gap of 5% to 20% between its CLOSURE and target accuracies remained on all tests, except for the `embed_spa_mat` test that MAC handled well even without fine-tuning. Notably, unlike the models relying on weakly-supervised program induction, MAC benefited greatly from fine-tuning on the challenging `and_mat_spa` test. Besides, MAC's absolute performance on `or_mat` and `or_mat_spa` is comparable to that of PG-Vector-NMN.

10.5. Related Work

Several related generalization tests that were proposed for CLEVR and other VQA datasets differ from CLOSURE in what they aim to measure and/or how they were constructed. The Compositional Generalization Test (CoGenT) from the original paper by Johnson et al. (2016) restricts the colors that cubes and cylinders can have in the training set images and inverts this restriction during the test time. By its design, CoGenT evaluates how robust a model is to a shift in the image distribution. On the contrary, in CLOSURE the image distribution remains the same at test time, but the question distribution changes to contain novel combinations of linguistic constructs from CLEVR. The generalization splits from the ShapeWorld platform (Kuhnle and Copestake, 2017) also focus on the difference in the distribution of images, not questions. The CLEVR-Humans dataset was collected by having crowd workers ask questions about CLEVR images (Johnson et al., 2017). Some questions from this dataset require reasoning that is outside of the scope of CLEVR, such as e.g. quantification ("Are all the balls small?"). In contrast, CLOSURE requires models to recombine only the well-known reasoning primitives. A compositional C-VQA split was proposed for the VQA 1.0 dataset (Agrawal et al., 2017). In C-VQA similar questions must have different answers when they appear in the training and test sets, yet the distributions of questions at training and testing remain similar, unlike CLOSURE.

Perhaps the closest to our work is the SQOOP dataset and the study conducted on it by Bahdanau et al. (2019b). SQOOP features questions of the form "Is there an X R of Y", where X and Y are object words and R a spatial relation. The authors test whether models can answer all possible SQOOP questions after training on a subset that is defined by holding out most of the (X, Y) pairs. The methodology of that study is thus very similar to ours, however the specific nature of the generalization split is different. Similarly to our results, Bahdanau et al. (2019b) report significant generalization gaps for a number of VQA models, with a notable exception of the Tensor-NMN, that generalized perfectly in their study when a tree-like layout was used to connect the modules. We believe our CLOSURE results are an important addition to the SQOOP ones. The specific cause of the aforementioned discrepancies between the two studies is, however, an intriguing question for future work.

As can be clearly seen from the performance of the NS-VQA model, much of the performance drop that we reported can be explained by insufficient systematicity of seq2seq models that we use for program generation. The SCAN dataset (Lake and Baroni, 2018) and the follow-up works (Loula et al., 2018; Bastings et al., 2018) have recently brought much-needed attention to this important issue. Compared to SCAN, CLOSURE features richer and more natural-looking language, and hence can serve to validate the conclusions drawn in recent SCAN-based studies, e.g. (Russin et al., 2019).

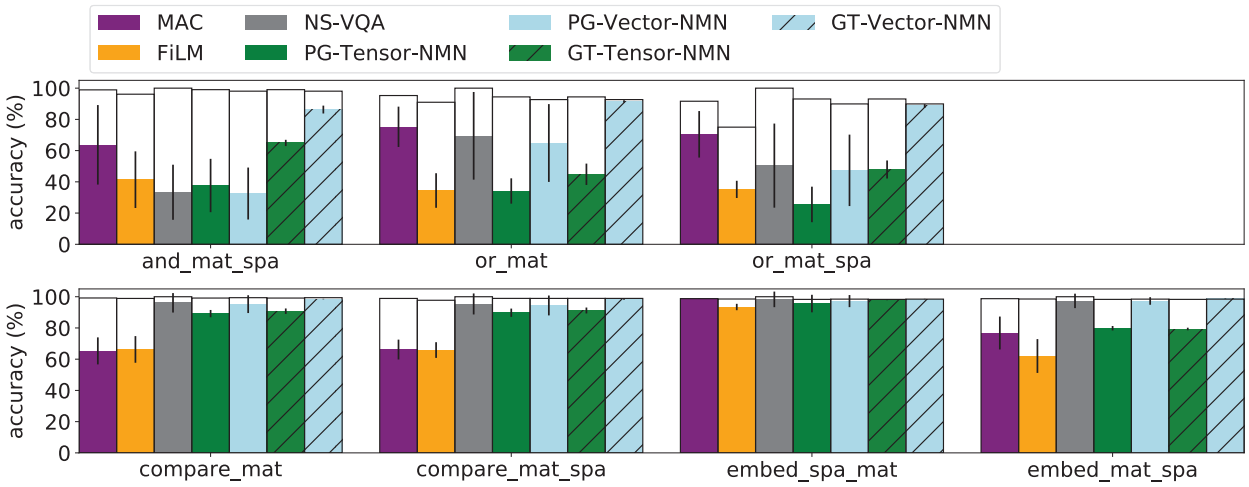


FIGURE 10.3. 0-shot accuracy of all models on the 7 CLOSURE tests. For each model and test, the white bar in the background is the model's accuracy on the closest CLEVR questions. The hatching used for "GT-..." models indicates that we used the ground-truth programs at test time.

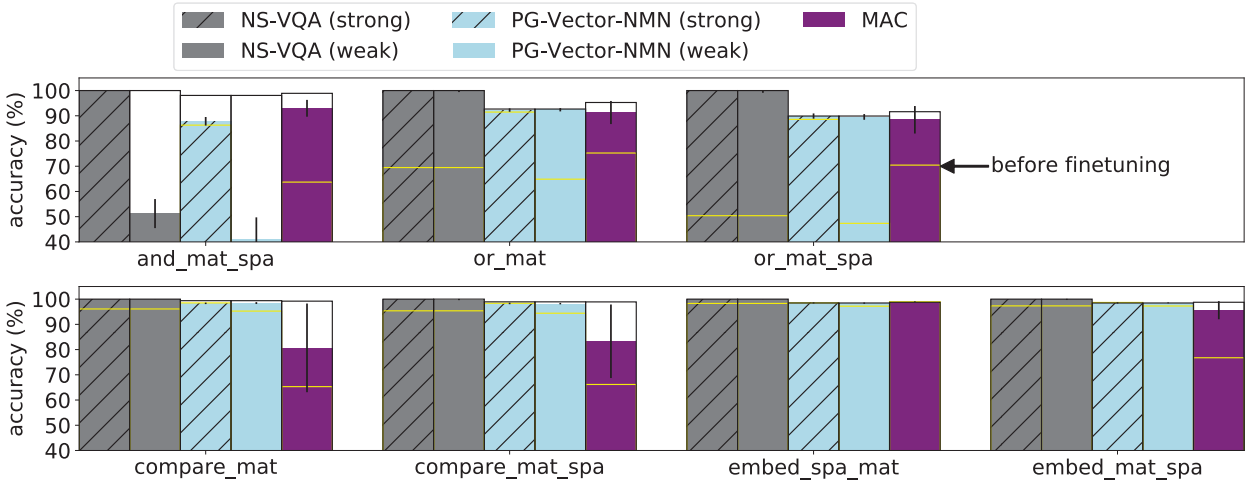


FIGURE 10.4. The accuracies for NS-VQA, PG-Vector-NMN and MAC after finetuning on 36 examples from each CLOSURE family. The background white bar is the model's accuracy on the closest CLEVR questions. The yellow horizontal line denotese the model's accuracy before fine-tuning. The hatcing indicates the use of ground-truth programs at the fine-tuning stage.

Prior work on Neural Module Networks features modules that output either attention maps (Andreas et al., 2016; Hu et al., 2017, 2018) or feature tensors (Johnson et al., 2017). The model by (Mascharka et al., 2018) combines modules with both attention- and tensor-valued outputs. Our Vector-NMN generalizes more systematically than its tensor-based predecessor by Johnson et al. (2017), while inheriting that model's simplicity, generality and good CLEVR performance.

10.6. Discussion

Our study shows that while models trained on CLEVR are very good at answering questions from CLEVR, their high performance quickly deteriorates when the question distribution features unfamiliar combinations of well-known primitives. We believe that this is an interesting finding, given that CLEVR puts VQA models in very favorable conditions: the training set is large and well-balanced and complex questions are well represented. One could say that we took advantage of a gap in the CLEVR question distribution to make our point, yet we believe that natural language datasets collected under naturalistic conditions will only have more gaps like this.

While in our 0-shot test of systematic generalization all models fare similarly badly, our few-shot learning study highlights important differences in their behavior. Given few examples, the program-based models either almost perfectly adapt to the target task or completely fail, depending on whether the right programs are found. For end-to-end continuous models back-propagation is always effective in adapting them to the target examples, but the systematic generalization gap is often not fully bridged with a number of examples that is sufficient for the program-based models. An important context for this comparison is that program-based models require seed programs to jump-start the training and are later on constrained to the seed lexicon. It would be highly desirable to combine the strengths of these two types of systems in one model without inheriting any of their limitations, a direction that we would like to explore in our future work.

We hope that the CLOSURE benchmark will facilitate future work in a number of directions. First, our results suggest that parsing (program generation in our case) can be the bottleneck for systematic generalization of grounded language learning. CLOSURE can thus be used for testing systematic generalization of neural parsers, complementing the SCAN benchmark and its variants. Besides parsing, further work on interchangeability of neural modules can be done using CLOSURE. While Vector-NMN improves upon prior work, it still generalizes suboptimally on the `and_mat_spa` test. Lastly, our test set construction methods can be adapted to natural data, yielding more insights and helping researchers make measurable progress towards learning-based models for grounded language understanding that generalize systematically.

10.7. Further details on CLOSURE and baseline questions

The exact CLOSURE templates can be found in Figure 10.5. The templates contain three kinds of slots:

- the slots ⟨A⟩ and ⟨Q⟩ will be filled by names of CLEVR properties, such as “shape”, “size”, “color” and “material”

- the slots ⟨Z⟩, ⟨C⟩, ⟨M⟩ and ⟨S⟩ will be either left blank or filled by words that characterize objects' properties, such as "big", "small", "yellow", "rubber", "cube", etc.
- the ⟨R⟩ slots will be filled with spatial words, such as "left of", "right of", "behind" or "in front of".

In some templates fragments of text are in square brackets; those are optional and will be discarded with 50% probability.

We generate the question from the templates in two stages. At the first stage, we fill the ⟨A⟩ and ⟨Q⟩ slots, and at the second stage we use CLEVR question generation code to fill the rest. The two-stage procedure is required because the original generation engine does not support ⟨A⟩ and ⟨Q⟩ slots; instead, CLEVR authors wrote unique templates for each property that is queried (the purpose of ⟨Q⟩) or used to refer to objects (the purpose of ⟨A⟩).

We used CLEVR generation code mostly as is, with the exception of two important modifications. First, we restricted the answers of counting questions to be either 1, 2 or 3, and also applied more aggressive rejection sampling to make these answers equally likely. We did so because the distribution of answers to counting questions in CLEVR is skewed, and answers of 4 and more are very unlikely. Instead of trying to replicate the original skewed answer distribution, we chose to enforce uniformity among those answers (that is 1, 2 and 3) that do have a significant probability in CLEVR. We did not allow 0 as the answer because due to implementation details, CLEVR questions that contain logical "or" and a complex spatial RE never have 0 as the answer. Overall, with our modifications to generation of counting questions we tried to put side the irrelevant confounding factors and focus on the impact of replacing a spatial RE with a matching one. To compute the appropriate target performance for `or_mat` and `or_mat_spa`, we generated new questions from the closest original CLEVR templates but using the modified version of the generation code.

The second modification concerns the question degeneracy check that is described in the appendix of (Johnson et al., 2016). In the reference question generation code it is only applied to programs with spatial relations. We modified the code to also apply the degeneracy check to matching relations.

A minor issue with compare_sim and compare_sim_loc questions is that the word "another" can be used in cases where it is not required, e.g. "... another cube that is the same size as the sphere". CLEVR generation code removes "another" in such cases, but we found it hard to extend this feature to CLOSURE questions in a maintainable way. In our preliminary experiments we found the proper handling of "another" does not change results of zero-shot experiments. We also experimented with removing the word "is" from "... and is the same ⟨A⟩ ..." in the `and_mat_spa` template to make it more similar to the closest

- **embed_spa_mat** Is there a ⟨Z⟩ ⟨C⟩ ⟨M⟩ ⟨S⟩ that is the same ⟨A⟩ as the ⟨Z2⟩ ⟨C2⟩ ⟨M2⟩ ⟨S2⟩ ⟨R⟩ the ⟨Z3⟩ ⟨C3⟩ ⟨M3⟩ ⟨S3⟩?
- **embed_mat_spa** Is there a ⟨Z⟩ ⟨C⟩ ⟨M⟩ ⟨S⟩ ⟨R⟩ the ⟨Z3⟩ ⟨C3⟩ ⟨M3⟩ ⟨S3⟩ that is the same ⟨A⟩ as ⟨Z2⟩ ⟨C2⟩ ⟨M2⟩ ⟨S2⟩?
- **compare_mat** There is another ⟨Z2⟩ ⟨C2⟩ ⟨M2⟩ ⟨S2⟩ that is the same ⟨A⟩ as the ⟨Z⟩ ⟨C⟩ ⟨M⟩ ⟨S⟩; does it have the same ⟨Q⟩ as the ⟨Z3⟩ ⟨C3⟩ ⟨M3⟩ ⟨S3⟩?
- **compare_mat_spa** There is another ⟨Z2⟩ ⟨C2⟩ ⟨M2⟩ ⟨S2⟩ that is the same ⟨A⟩ as the ⟨Z⟩ ⟨C⟩ ⟨M⟩ ⟨S⟩; does it have the same ⟨Q⟩ as the ⟨Z4⟩ ⟨C4⟩ ⟨M4⟩ ⟨S4⟩ [that is] ⟨R2⟩ the ⟨Z3⟩ ⟨C3⟩ ⟨M3⟩ ⟨S3⟩?
- **and_mat_spa** What is the ⟨Q⟩ of the ⟨Z3⟩ ⟨C3⟩ ⟨M3⟩ ⟨S3⟩ that is ⟨R2⟩ the ⟨Z2⟩ ⟨C2⟩ ⟨M2⟩ ⟨S2⟩ and is the same ⟨A⟩ as the ⟨Z⟩ ⟨C⟩ ⟨M⟩ ⟨S⟩?
- **or_mat** How many things are [either] ⟨Z⟩ ⟨C⟩ ⟨M⟩ ⟨S⟩s or ⟨Z3⟩ ⟨C3⟩ ⟨M3⟩ ⟨S3⟩s that are the same ⟨A⟩ as the ⟨Z2⟩ ⟨C2⟩ ⟨M2⟩ ⟨S2⟩?
- **or_mat_spa** How many things are [either] ⟨Z2⟩ ⟨C2⟩ ⟨M2⟩ ⟨S2⟩s [that are] ⟨R⟩ the ⟨Z⟩ ⟨C⟩ ⟨M⟩ ⟨S⟩ or ⟨Z4⟩ ⟨C4⟩ ⟨M4⟩ ⟨S4⟩s that are the same ⟨A⟩ as the ⟨Z3⟩ ⟨C3⟩ ⟨M3⟩ ⟨S3⟩?

FIGURE 10.5. CLOSURE templates.

CLEVR questions, in which "and" combines prepositional phrases (i.e. "... that is left of the cube and right of the sphere"). Likewise, we saw no influence on the zero-shot results.

Chapter 11

GENERAL CONCLUSION

Studies of deep learning systems that understand grounded language are still in their infancy. In the articles that are presented in this thesis, we attempted to lay a solid foundation for this field. The BabyAI platform that we presented in Chapter 6 challenges researchers to achieve near-perfect performance using a smaller amount of supervision. We hope that the 19 levels of BabyAI will facilitate studies on how to make deep language grounding methods benefit from curriculum learning. In Chapter 4 we show that such near-perfect performance can also be achieved using restricted supervision in the form of instructions and their corresponding goal states. The AGILE algorithm that we developed for this purpose relies on adversarial imitation learning. An important finding of the AGILE project is that the challenge of false negative examples needs to be tackled to approach convergence. A straightforward next step in this line of research can be to study the sample efficiency of AGILE-like algorithms more rigorously using the BabyAI platform.

Our visual question answering studies presented in Chapters 8 and 10 contribute much-needed clarity on the systematic generalization abilities of modern models. The SQOOP and CLOSURE evaluations highlight the undesirable tendency of end-to-end continuous models to adapt to particularities of the current data distribution. While explicitly modular models generally fared better in both studies, a challenge of making these models as domain-agnostic and flexible as the end-to-end continuous models remains open. The modular models rely on layouts or programs that specify which modules should be used and how these are connected, and results in Chapter 10 suggest that predicting such programs with a neural program generator can on its own be a systematic generalization bottleneck. Furthermore, ground-truth programs are required for the currently available methods for training the program generator, and as a consequence, using modular models on a new domain requires a significant amount of domain knowledge. Future work could focus on addressing these issues, as well as on the module coadaptation problem that we have alleviated but not yet fully solved with the new Vector-NMN module.

11.1. Towards Better Methodology for Systematic Generalization Studies

As research on systematic (or compositional) generalization gains popularity, future work should put it on a more solid theoretical foundation. Most recently proposed studies, including the ones presented in Chapters 8 and 10, start from informal considerations and turn them into concrete tasks with generalization splits of rather arbitrary character. These studies provide useful insights, but it is unfortunate that the implied bespoke definitions are sometimes inconsistent with each other. For example, in CLEVR-CoGenT (Johnson et al., 2016) and ShapeWorld (Kuhnle and Copestake, 2017) benchmarks the image distributions at training and test time are deliberately made different. This does not align with our view that systematic generalization in language understanding should be about recombining perfectly familiar individual words in novel ways, and that the meanings of the individual words should be the same in training and testing. The lack of a theoretical foundation makes it hard to precisely point out the difference and discuss its implications. This problem is not new, for the original article by Fodor and Pylyshyn (1988) that contributed the notion of systematicity, as well as later contributions (e.g. (Marcus, 2003)), were very informal in the first place. In general, the notions of systematicity and compositionality resist easy formalization (see e.g. (Zadrozny, 1994)). A promising formulation of compositional generalization has recently been proposed in a concurrent work by Keysers et al. (2020), where it is defined as generalization across distributions in which the atoms (i.e. the grammar rules) occur with the same frequencies, whereas compounds (i.e. the combinations of the aforementioned rules) have different frequencies. We find this intuition relevant, although it seems unfortunate that this definition depends on the particular grammatical formalism used. In this regard, drawing connections between systematic generalization and the formal semantics subfield of linguistics would be desirable. For perceptual grounded settings, such as the ones we study here, it would be interesting to establish closer connections to cognitive science, whose scholars also often use the term compositionality (Lake et al., 2016).

Furthermore, to make systematic generalization studies more relevant for downstream applications, future work could use models that are pretrained on large corpora in self-supervised way, such as e.g. BERT (Devlin et al., 2018). The parameter settings of these models are shaped by enormous amount of training, and several studies argued that these models internally know a lot about language (Tenney et al., 2019; Hewitt and Manning, 2019). It is conceivable that fine-tuning these models the right way could yield much stronger generalization than training randomly initialized models from scratch.

11.2. Towards Better Data Efficiency and More Systematic Generalization

Looking forward, a promising research direction for the future work is to bridge the gap between the deep learning approaches that we study here and the largely symbolic approaches that roboticists rely on (see Section 2.3.1). In particular, there are two assumptions in these models that seem to be particularly relevant to our goals. First, the object-centric representations that are characteristic of classical systems can facilitate generalization through explicit disentangling of different objects' properties. That said, there is no universally correct way to partition the world into discrete things that can be referred to. It would therefore be interesting to build systems that identify and represent the set of relevant entities in a way that depends on what the instruction or the question refers to. Second, while grammar-based approaches to constructing meanings in a compositional way appear overly restrictive, the limited context sensitivity with which decisions are made in such models might be the missing ingredient that we need to add to neural models to improve their generalization and data efficiency.

A complementary line of research could study how better learning objectives could be used to improve generalization. The conventional objectives do not encourage neural networks to find solutions that are in some sense more general than others; in fact defining what it means for the solution to be general is arguably the main challenge in this context. Data-driven definitions of generality, such as e.g. the CLOSURE tests that we presented in Chapter 10, lose some of their validity once they are used to provide any form of training signal. One can attempt to provide automatic generalization signal by using the domain-specific considerations regarding the compositional nature of the world and language (see (Andreas, 2019) and (Lake, 2019) for recent work in this direction). Tests like CLOSURE can then be used to verify the success of such optimization for generalization.

To sum up, we believe that integrating what we know or even intuit about world, humans and natural language into the learning-centric approach of deep learning is the way forward. No matter if this is achieved by explicitly altering models or by formulating more complex training objectives, it is important that researchers carefully assess the achieved progress. We hope that the contributions of this thesis will either directly (as benchmarks) or indirectly (as food for thought) have a positive impact on the way research on grounded language understanding is carried out.

Bibliography

Abbeel, P. and Ng, A. Y. (2004). Apprenticeship Learning via Inverse Reinforcement Learning. In *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML '04.

Agrawal, A., Batra, D., and Parikh, D. (2016). Analyzing the Behavior of Visual Question Answering Models. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*.

Agrawal, A., Batra, D., Parikh, D., and Kembhavi, A. (2018). Don't Just Assume; Look and Answer: Overcoming Priors for Visual Question Answering. In *Proceedings of the 2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. arXiv: 1712.00377.

Agrawal, A., Kembhavi, A., Batra, D., and Parikh, D. (2017). C-VQA: A Compositional Split of the Visual Question Answering (VQA) v1.0 Dataset. *arXiv:1704.08243 [cs]*. arXiv: 1704.08243.

Anderson, P., He, X., Buehler, C., Teney, D., Johnson, M., Gould, S., and Zhang, L. (2018a). Bottom-Up and Top-Down Attention for Image Captioning and Visual Question Answering. *Proceedings of the 2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. arXiv: 1707.07998.

Anderson, P., Wu, Q., Teney, D., Bruce, J., Johnson, M., Sünderhauf, N., Reid, I., Gould, S., and Hengel, A. v. d. (2018b). Vision-and-Language Navigation: Interpreting visually-grounded navigation instructions in real environments. In *Proceedings of the 2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Andreas, J. (2019). Good-Enough Compositional Data Augmentation. *arXiv:1904.09545 [cs]*. arXiv: 1904.09545.

Andreas, J. and Klein, D. (2015). Alignment-Based Compositional Semantics for Instruction Following. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*.

Andreas, J., Rohrbach, M., Darrell, T., and Klein, D. (2016). Neural Module Networks. In *Proceedings of 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Antol, S., Agrawal, A., Lu, J., Mitchell, M., Batra, D., Lawrence Zitnick, C., and Parikh, D. (2015). Vqa: Visual Question Answering. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (CVPR)*, pages 2425–2433.

Artzi, Y. and Zettlemoyer, L. (2013). Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association for Computational Linguistics*, 1:49–62.

Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural Machine Translation by Jointly Learning to Align and Translate. In *International Conference on Learning Representations, ICLR 2015*.

Bahdanau, D., Hill, F., Leike, J., Hughes, E., Hosseini, A., Kohli, P., and Grefenstette, E. (2019a). Learning to Understand Goal Specifications by Modelling Reward. In *International Conference on Learning Representations, ICLR 2019*.

Bahdanau, D., Murty, S., Noukhovitch, M., Nguyen, T. H., Vries, H. d., and Courville, A. (2019b). Systematic Generalization: What Is Required and Can It Be Learned? In *International Conference on Learning Representations, ICLR 2019*.

Bastings, J., Baroni, M., Weston, J., Cho, K., and Kiela, D. (2018). Jump to better conclusions: SCAN both left and right. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*. Association for Computational Linguistics.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. (2013). The Arcade Learning Environment: An Evaluation Platform for General Agents. *Journal of Artificial Intelligence Research*, 47:253–279. arXiv: 1207.4708.

Bender, E. M. (2013). Linguistic Fundamentals for Natural Language Processing: 100 Essentials from Morphology and Syntax. *Synthesis Lectures on Human Language Technologies*, 6(3):1–184.

Bengio, Y. (2009). Learning Deep Architectures for AI. *Foundations and Trends® in Machine Learning*, 2(1):1–127.

Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A Neural Probabilistic Language Model. *Journal of Machine Learning Research*, 3(Feb):1137–1155.

Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009). Curriculum learning. In *Proceedings of the 26th International Conference on Machine Learning (ICML)*, pages 41–48.

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning Long-term Dependencies With Gradient Descent is Difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166.

Bingham, E., Molino, P., Szerlip, P., Fritz, O., and Noah, G. (2017). Characterizing how Visual Question Answering scales with the world. In *NIPS 2017 Visually-Grounded Interaction and Language Workshop*.

Bowman, S. R., Angeli, G., Potts, C., and Manning, C. D. (2015). A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics.

Brodeur, S., Perez, E., Anand, A., Golemo, F., Celotti, L., Strub, F., Rouat, J., Larochelle, H., and Courville, A. (2017). HoME: a Household Multimodal Environment. *arXiv:1711.11017 [cs, eess]*. arXiv: 1711.11017.

Calvo, F. and Colunga, E. (2003). The statistical brain: Reply to Marcus' The algebraic mind. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, volume 25.

Chaplot, D. S., Sathyendra, K. M., Pasumarthi, R. K., Rajagopal, D., and Salakhutdinov, R. (2018). Gated-Attention Architectures for Task-Oriented Language Grounding. In *Proceedings of 32nd AAAI Conference on Artificial Intelligence*.

Chen, D. L. and Mooney, R. J. (2011). Learning to Interpret Natural Language Navigation Instructions from Observations. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, pages 859–865.

Cho, K., van Merrienboer, B., Gulcehre, C., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Christiano, P., Leike, J., Brown, T. B., Martic, M., Legg, S., and Amodei, D. (2017). Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems 30*. arXiv: 1706.03741.

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. (2011). Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314.

Côté, M.-A., Kádár, A., Yuan, X., Kybartas, B., Barnes, T., Fine, E., Moore, J., Hausknecht, M., Asri, L. E., Adada, M., Tay, W., and Trischler, A. (2018). TextWorld: A Learning Environment for Text-based Games. *arXiv:1806.11532 [cs, stat]*. arXiv: 1806.11532.

Daumé III, H., Langford, J., and Marcu, D. (2009). Search-based structured prediction. *Machine learning*, 75(3):297–325.

DeepMind (2017). PycoLab.

Denil, M., Colmenarejo, S. G., Cabi, S., Saxton, D., and de Freitas, N. (2017). Programmable Agents. *arXiv:1706.06383 [cs, stat]*.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics,*

NAACL 2019.

Dong, L. and Lapata, M. (2016). Language to Logical Form with Neural Attention. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016*. Association for Computational Linguistics.

Duvallet, F., Kollar, T., and Stentz, A. (2013). Imitation learning for natural language direction following through unknown environments. In *2013 IEEE International Conference on Robotics and Automation*, pages 1047–1053.

Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, 14(2):179–211.

Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., Legg, S., and Kavukcuoglu, K. (2018). IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures. In *Proceedings of the 22nd International Conference on Machine Learning, ICML 2018*. arXiv: 1802.01561.

Feldman, J., Lakoff, G., Bailey, D., Narayanan, S., Regier, T., and Stolcke, A. (1996). L 0—the first five years of an automated language acquisition project. In *Integration of natural language and vision processing*, pages 205–231. Springer.

Fodor, J. A. (1975). *The language of thought*, volume 5. Harvard University Press.

Fodor, J. A. and Pylyshyn, Z. W. (1988). Connectionism and cognitive architecture: A critical analysis. *Cognition*, 28(1):3–71.

Fried, D., Hu, R., Cirik, V., Rohrbach, A., Andreas, J., Morency, L.-P., Berg-Kirkpatrick, T., Saenko, K., Klein, D., and Darrell, T. (2018). Speaker-Follower Models for Vision-and-Language Navigation. In *Advances in Neural Information Processing Systems 31, NeurIPS 2018*. arXiv: 1806.02724.

Fu, J., Korattikara, A., Levine, S., and Guadarrama, S. (2018). From Language to Goals: Inverse Reinforcement Learning for Vision-Based Instruction Following. In *International Conference on Learning Representations*.

Ganin, Y., Kulkarni, T., Babuschkin, I., Eslami, S. M. A., and Vinyals, O. (2018). Synthesizing Programs for Images using Reinforced Adversarial Learning. *Proceedings of the 35th International Conference on Machine Learning, ICML 2018*. arXiv: 1804.01118.

Gaunt, A. L., Brockschmidt, M., Kushman, N., and Tarlow, D. (2016). Differentiable Programs with Neural Libraries. In *Proceedings of the 34th International Conference on Machine Learning*. arXiv: 1611.02109.

Gehring, J., Auli, M., Grangier, D., Yarats, D., and Dauphin, Y. N. (2017). Convolutional Sequence to Sequence Learning. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017*. arXiv: 1705.03122.

Geman, D., Geman, S., Hallonquist, N., and Younes, L. (2015). Visual Turing test for computer vision systems. *Proceedings of the National Academy of Sciences*, 112(12):3618–3623.

Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep Sparse Rectifier Neural Networks. In *AISTATS*, volume 15, page 275.

Gong, Y., Luo, H., and Zhang, J. (2018). Natural Language Inference over Interaction Space. In *Proceedings of the 2018 International Conference on Learning Representations, ICLR 2018*. arXiv: 1709.04348.

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

Gopalan, N., Arumugam, D., Wong, L., and Tellex, S. (2018). Sequence-to-Sequence Language Grounding of Non-Markovian Task Specifications. In *Robotics: Science and Systems XIV*. Robotics: Science and Systems Foundation.

Gordon, J. and Van Durme, B. (2013). Reporting Bias and Knowledge Acquisition. In *Proceedings of the 2013 Workshop on Automated Knowledge Base Construction*. ACM. event-place: San Francisco, California, USA.

Goyal, Y., Khot, T., Summers-Stay, D., Batra, D., and Parikh, D. (2016). Making the V in VQA Matter: Elevating the Role of Image Understanding in Visual Question Answering. *arXiv:1612.00837 [cs]*. arXiv: 1612.00837.

Graves, A. (2011). Practical Variational Inference for Neural Networks. In Shawe-Taylor, J., Zemel, R. S., Bartlett, P. L., Pereira, F., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 24*, pages 2348–2356. Curran Associates, Inc.

Graves, A., Wayne, G., Reynolds, M., Harley, T., Danihelka, I., Grabska-Barwińska, A., Colmenarejo, S. G., Grefenstette, E., Ramalho, T., Agapiou, J., Badia, A. P., Hermann, K. M., Zwols, Y., Ostrovski, G., Cain, A., King, H., Summerfield, C., Blunsom, P., Kavukcuoglu, K., and Hassabis, D. (2016). Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476.

Gupta, N. and Lewis, M. (2018). Neural Compositional Denotational Semantics for Question Answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, EMNLP 2018*.

Gururangan, S., Swayamdipta, S., Levy, O., Schwartz, R., Bowman, S. R., and Smith, N. A. (2018). Annotation Artifacts in Natural Language Inference Data. In *Proceedings of NAACL-HLT 2018*. arXiv: 1803.02324.

Halevy, A., Norvig, P., and Pereira, F. (2009). The Unreasonable Effectiveness of Data. *IEEE Intelligent Systems*, 24:8–12.

Harnad, S. (1990). The symbol grounding problem. *Physica D: Nonlinear Phenomena*, 42(1):335–346.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Hebb, D. O. (1949). *The organization of behavior; a neuropsychological theory*. Wiley, Oxford, England.

Heim, I. and Kratzer, A. (1998). *Semantics in Generative Grammar*. Blackwell.

Hermann, K. M., Hill, F., Green, S., Wang, F., Faulkner, R., Soyer, H., Szepesvari, D., Czarnecki, W. M., Jaderberg, M., Teplyashin, D., Wainwright, M., Apps, C., Hassabis, D., and Blunsom, P. (2017). Grounded Language Learning in a Simulated 3d World. *arXiv:1706.06551 [cs, stat]*.

Hermann, K. M., Kocisky, T., Grefenstette, E., Espeholt, L., Kay, W., Suleyman, M., and Blunsom, P. (2015). Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems*, pages 1693–1701.

Hewitt, J. and Manning, C. D. (2019). A Structural Probe for Finding Syntax in Word Representations. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4129–4138, Minneapolis, Minnesota. Association for Computational Linguistics.

Ho, J. and Ermon, S. (2016). Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems*, pages 4565–4573.

Hochreiter, S. and Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780.

Hu, R., Andreas, J., Darrell, T., and Saenko, K. (2018). Explainable Neural Computation via Stack Neural Module Networks. In *arXiv:1807.08556 [cs]*. arXiv: 1807.08556.

Hu, R., Andreas, J., Rohrbach, M., Darrell, T., and Saenko, K. (2017). Learning to Reason: End-to-End Module Networks for Visual Question Answering. In *Proceedings of 2017 IEEE International Conference on Computer Vision*. arXiv: 1704.05526.

Hudson, D. A. and Manning, C. D. (2018). Compositional Attention Networks for Machine Reasoning. In *Proceedings of the 2018 International Conference on Learning Representations*.

Hudson, D. A. and Manning, C. D. (2019). GQA: A New Dataset for Real-World Visual Reasoning and Compositional Question Answering. *Proceedings of the 2019 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019*. arXiv: 1902.09506.

Ioffe, S. and Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015*.

Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., and Kavukcuoglu, K. (2016). Reinforcement Learning with Unsupervised Auxiliary Tasks. In *International Conference on Learning Representations, ICLR 2016*.

Janner, M., Narasimhan, K., and Barzilay, R. (2017). Representation Learning for Grounded Spatial Reasoning. *Transactions of the Association for Computational Linguistics, TACL*.

Jia, R. and Liang, P. (2017). Adversarial Examples for Evaluating Reading Comprehension Systems. *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017*.

Jiang, Y., Natarajan, V., Chen, X., Rohrbach, M., Batra, D., and Parikh, D. (2018). Pythia v0.1: the Winning Entry to the VQA Challenge 2018. *arXiv:1807.09956 [cs]*. arXiv: 1807.09956.

Johnson, J., Hariharan, B., van der Maaten, L., Fei-Fei, L., Zitnick, C. L., and Girshick, R. (2016). CLEVR: A Diagnostic Dataset for Compositional Language and Elementary Visual Reasoning. In *Proceedings of 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. arXiv: 1612.06890.

Johnson, J., Hariharan, B., van der Maaten, L., Hoffman, J., Fei-Fei, L., Zitnick, C. L., and Girshick, R. (2017). Inferring and Executing Programs for Visual Reasoning. In *Proceedings of 2017 IEEE International Conference on Computer Vision, ICCV 2017*.

Kalchbrenner, N. and Blunsom, P. (2013). Recurrent Continuous Translation Models. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017*, volume 3, page 413.

Kannan, A., Kurach, K., Ravi, S., Kaufmann, T., Tomkins, A., Miklos, B., Corrado, G., Lukacs, L., Ganea, M., Young, P., and Ramavajjala, V. (2016). Smart Reply: Automated Response Suggestion for Email. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16. ACM.

Kempka, M., Wydmuch, M., Runc, G., Toczek, J., and Jaskowski, W. (2016). ViZDoom: A Doom-based AI research platform for visual reinforcement learning. In *IEEE Conference on Computational Intelligence and Games, CIG 2016*, pages 1–8. IEEE.

Keysers, D., Schärli, N., Scales, N., Buisman, H., Furrer, D., Kashubin, S., Momchev, N., Sinopalnikov, D., Stafiniak, L., Tihon, T., Tsarkov, D., Wang, X., van Zee, M., and Bousquet, O. (2020). Measuring Compositional Generalization: A Comprehensive Method on Realistic Data. In *International Conference on Learning Representations*. arXiv: 1912.09713.

Kingma, D. P. and Ba, J. (2015). Adam: A Method for Stochastic Optimization. In *Proceedings of the 2015 International Conference on Learning Representations, ICLR 2015*. arXiv: 1412.6980.

Kiros, R., Salakhutdinov, R., and Zemel, R. S. (2014). Unifying visual-semantic embeddings with multimodal neural language models. *arXiv preprint arXiv:1411.2539*.

Knox, W. B. and Stone, P. (2009). Interactively Shaping Agents via Human Reinforcement: The TAMER Framework. In *Proceedings of the Fifth International Conference on Knowledge Capture*, K-CAP '09, pages 9–16, New York, NY, USA. ACM. event-place: Redondo Beach, California, USA.

Koenig, N. and Howard, A. (2004). Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2149–2154, Sendai, Japan.

Kollar, T., Tellex, S., Roy, D., and Roy, N. (2010). Toward understanding natural language directions. In *2010 5th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 259–266.

Kollar, T., Tellex, S., Walter, M., Huang, A., Bachrach, A., Hemachandra, S., Brunskill, E., Banerjee, A., Roy, D., Teller, S., and Roy, N. (2017). Generalized Grounding Graphs: A Probabilistic Framework for Understanding Grounded Commands. *arXiv:1712.01097 [cs]*. arXiv: 1712.01097.

Kolve, E., Mottaghi, R., Gordon, D., Zhu, Y., Gupta, A., and Farhadi, A. (2017). AI2-THOR: An Interactive 3d Environment for Visual AI. *CoRR*, abs/1712.05474.

Krishnamurthy, J. and Kollar, T. (2013). Jointly Learning to Parse and Perceive: Connecting Natural Language to the Physical World. *Transactions of the Association for Computational Linguistics*, 1:193–206.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25*, pages 1106–1114.

Kuhnle, A. and Copestake, A. (2017). ShapeWorld - A new test methodology for multimodal language understanding. *arXiv:1704.04517 [cs]*. arXiv: 1704.04517.

Kumar, M. P., Packer, B., and Koller, D. (2010). Self-Paced Learning for Latent Variable Models. In *Advances in Neural Information Processing Systems 23*, pages 1189–1197.

Lake, B. M. (2019). Compositional generalization through meta sequence-to-sequence learning. arXiv: 1906.05381.

Lake, B. M. and Baroni, M. (2018). Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *Proceedings of the 36th International Conference on Machine Learning*. arXiv: 1711.00350.

Lake, B. M., Ullman, T. D., Tenenbaum, J. B., and Gershman, S. J. (2016). Building Machines That Learn and Think Like People. *arXiv:1604.00289 [cs, stat]*. arXiv: 1604.00289.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551.

Lenat, D. B. (1995). CYC: A Large-scale Investment in Knowledge Infrastructure. *Commun. ACM*, 38(11):33–38.

Levesque, H., Davis, E., and Morgenstern, L. (2012). The Winograd Schema challenge. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Thirteenth International Conference, {KR} 2012*.

Liang, P., Jordan, M. I., and Klein, D. (2013). Learning Dependency-Based Compositional Semantics. *Computational Linguistics*, 39(2):389–446.

Liu, C.-W., Lowe, R., Serban, I. V., Noseworthy, M., Charlin, L., and Pineau, J. (2016). How NOT To Evaluate Your Dialogue System: An Empirical Study of Unsupervised Evaluation

Metrics for Dialogue Response Generation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, {EMNLP} 2016*.

Loula, J., Baroni, M., and Lake, B. M. (2018). Rearranging the Familiar: Testing Compositional Generalization in Recurrent Networks. In *Proceedings of the 2018 BlackboxNLP EMNLP Workshop*.

Luong, M.-T., Pham, H., and Manning, C. D. (2015). Effective Approaches to Attention-based Neural Machine Translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, {EMNLP} 2015*. arXiv: 1508.04025.

MacGlashan, J., Babes-Vroman, M., desJardins, M., Littman, M. L., Muresan, S., Squire, S., Tellex, S., Arumugam, D., and Yang, L. (2015). Grounding English Commands to Reward Functions. In *Robotics: Science and Systems*.

Macmahon, M., Stankiewicz, B., and Kuipers, B. (2006). Walk the Talk: Connecting Language, Knowledge, Action in Route Instructions. In *In Proc. of the Nat. Conf. on Artificial Intelligence (AAAI*, pages 1475–1482.

Malinowski, M. and Fritz, M. (2014). A Multi-world Approach to Question Answering About Real-world Scenes Based on Uncertain Input. In *Proceedings of the 27th International Conference on Neural Information Processing Systems*, NIPS'14, pages 1682–1690, Cambridge, MA, USA. MIT Press.

Manning, C. D. and Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA, USA.

Mao, J., Gan, C., Kohli, P., Tenenbaum, J. B., and Wu, J. (2018). The Neuro-Symbolic Concept Learner: Interpreting Scenes, Words, and Sentences From Natural Supervision. In *International Conference on Learning Representations, ICLR 2019*.

Marcus, G. F. (1998). Rethinking Eliminative Connectionism. *Cognitive Psychology*, 37(3):243–282.

Marcus, G. F. (2003). *The algebraic mind: Integrating connectionism and cognitive science*. MIT press.

Marcus, M. P., Santorini, B., and Marcinkiewicz, M. A. (1993). Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Mascharka, D., Tran, P., Soklaski, R., and Majumdar, A. (2018). Transparency by Design: Closing the Gap Between Performance and Interpretability in Visual Reasoning. In *2018 IEEE Conference on Computer Vision and Pattern Recognition*. arXiv: 1803.05268.

Matuszek, C., FitzGerald, N., Zettlemoyer, L., Bo, L., and Fox, D. (2012). A Joint Model of Language and Perception for Grounded Attribute Learning. In *Proceedings of the 29th International Conference on Machine Learning, ICML 2012*. arXiv: 1206.6423.

Matuszek, C., Herbst, E., Zettlemoyer, L., and Fox, D. (2013). Learning to parse natural language commands to a robot control system. In *Experimental Robotics*, pages 403–415. Springer.

McCarthy, J., Minsky, M., Rochester, N., and Shannon, C. (1956). A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence. Technical report.

McCoy, R. T., Pavlick, E., and Linzen, T. (2019). Right for the Wrong Reasons: Diagnosing Syntactic Heuristics in Natural Language Inference. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, {ACL} 2019*. arXiv: 1902.01007.

Mei, H., Bansal, M., and Walter, M. R. (2016). Listen, Attend, and Walk: Neural Mapping of Navigational Instructions to Action Sequences. In *Proceedings of the 2016 AAAI Conference on Artificial Intelligence*.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Mikolov, T., Karafiát, M., Burget, L., Cernocky, J., and Khudanpur, S. (2010). Recurrent neural network based language model. In *Interspeech*, volume 2, page 3.

Minsky, M. and Papert, S. (1969). *Perceptrons*. Perceptrons. M.I.T. Press, Oxford, England.

Misra, D., Langford, J., and Artzi, Y. (2017). Mapping Instructions and Visual Observations to Actions with Reinforcement Learning. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, {EMNLP} 2017*.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937.

Monahan, G. E. (1982). State of the art—a survey of partially observable Markov decision processes: theory, models, and algorithms. *Management Science*, 28(1):1–16.

Newell, A. and Simon, H. A. (1976). Computer Science As Empirical Inquiry: Symbols and Search. *Commun. ACM*, 19(3):113–126.

Ng, A. Y., Harada, D., and Russell, S. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the 16th International Conference on Machine Learning, ICML 1999*, volume 99, pages 278–287.

Ng, A. Y. and Russell, S. (2000). Algorithms for Inverse Reinforcement Learning. In *in Proc. 17th International Conf. on Machine Learning*, pages 663–670.

Oh, J., Singh, S., Lee, H., and Kohli, P. (2017). Zero-Shot Task Generalization with Multi-Task Deep Reinforcement Learning. In *Proceedings of The 34st International Conference on Machine Learning*.

Pathak, D., Mahmoudieh, P., Luo, G., Agrawal, P., Chen, D., Shentu, Y., Shelhamer, E., Malik, J., Efros, A. A., and Darrell, T. (2018). Zero-shot visual imitation. In *International Conference on Learning Representations, ICLR 2018*.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau,

D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, volume 14, pages 1532–1543.

Perez, E., Strub, F., de Vries, H., Dumoulin, V., and Courville, A. (2017). FiLM: Visual Reasoning with a General Conditioning Layer. In *In Proceedings of the 2017 AAAI Conference on Artificial Intelligence*.

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep Contextualized Word Representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics, NAACL-HLT 2018*. arXiv: 1802.05365.

Pinker, S. (1994). *The language instinct: How the mind creates language*. Penguin UK.

Pomerleau, D. A. (1991). Efficient Training of Artificial Neural Networks for Autonomous Navigation. *Neural Computation*, 3(1):88–97.

Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. (2016). SQuAD: 100, 000+ Questions for Machine Comprehension of Text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, pages 2383–2392.

Rasmussen, C. E. and Williams, C. K. I. (2005). Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning). MIT Press.

Reed, S. and de Freitas, N. (2015). Neural Programmer-Interpreters. In *International Conference on Learning Representations, ICLR 2016*. arXiv: 1511.06279.

Rosenblatt, F. (1958). The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain. *Psychological Review*, pages 65–386.

Ross, S., Gordon, G., and Bagnell, D. (2011). A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. In *PMLR*, pages 627–635.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986a). Learning representations by back-propagating errors. *Nature*, 323(6088):533–536.

Rumelhart, D. E., McClelland, J. L., and University of California, S. D. P. R. G. (1986b). *Parallel distributed processing : explorations in the microstructure of cognition*. MIT Press.

Rush, A. M., Chopra, S., and Weston, J. (2015). A neural attention model for abstractive sentence summarization. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, {EMNLP} 2015*.

Russin, J., Jo, J., and O'Reilly, R. C. (2019). Compositional generalization in a deep seq2seq model by separating syntax and semantics. *arXiv:1904.09708 [cs, stat]*. arXiv: 1904.09708.

Santoro, A., Raposo, D., Barrett, D. G. T., Malinowski, M., Pascanu, R., Battaglia, P., and Lillicrap, T. (2017). A simple neural network module for relational reasoning. In *Advances*

in Neural Information Processing Systems 31. arXiv: 1706.01427.

Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. (2015). High-Dimensional Continuous Control Using Generalized Advantage Estimation. In *Advances in Neural Information Processing Systems 30*.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal Policy Optimization Algorithms. *arXiv:1707.06347 [cs]*. arXiv: 1707.06347.

Schuster, M. and Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *Signal Processing, IEEE Transactions on*, 45(11):2673–2681.

Sercu, T., Puhrsch, C., Kingsbury, B., and LeCun, Y. (2016). Very deep multilingual convolutional neural networks for LVCSR. pages 4955–4959. IEEE.

Smolensky, P. (1987). The constituent structure of connectionist mental states: A reply to Fodor and Pylyshyn. *Southern Journal of Philosophy*, 26(Supplement):137–161.

Socher, R., Perelygin, A., Wu, J. Y., Chuang, J., Manning, C. D., Ng, A. Y., Potts, C., and others (2013). Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, volume 1631, page 1642.

Speer, R. and Havasi, C. (2013). ConceptNet 5: A large semantic network for relational knowledge. In *The People's Web Meets NLP*, pages 161–176. Springer.

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958.

Stadie, B. C., Abbeel, P., and Sutskever, I. (2017). Third-Person Imitation Learning. In *International Conference on Learning Representations, ICLR 2017*.

Steedman, M. (1996). *Surface structure and interpretation*. MIT press.

Suarez, J., Johnson, J., and Li, F.-F. (2018). DDRprog: A CLEVR Differentiable Dynamic Reasoning Programmer. *arXiv:1803.11361 [cs]*. arXiv: 1803.11361.

Suhr, A., Lewis, M., Yeh, J., and Artzi, Y. (2017). A corpus of natural language for visual reasoning. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 217–223.

Sukhbaatar, S., Szlam, A., Synnaeve, G., Chintala, S., and Fergus, R. (2015). MazeBase: A Sandbox for Learning from Games. *arXiv:1511.07401 [cs]*. arXiv: 1511.07401.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to Sequence Learning with Neural Networks. In *Advances in Neural Information Processing Systems 27*, pages 3104–3112.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. (2000). Policy Gradient Methods for Reinforcement Learning with Function Approximation. In Solla, S. A., Leen,

T. K., and Müller, K., editors, *Advances in Neural Information Processing Systems 12*, pages 1057–1063. MIT Press.

Tellex, S., Kollar, T., Dickerson, S., Walter, M. R., Banerjee, A. G., Teller, S., and Roy, N. (2011). Understanding Natural Language Commands for Robotic Navigation and Mobile Manipulation. In *Twenty-Fifth AAAI Conference on Artificial Intelligence*.

Tellex, S., Thaker, P., Joseph, J., and Roy, N. (2014). Learning perceptually grounded word meanings from unaligned parallel data. *Machine Learning*, 94(2):151–167.

Tenney, I., Das, D., and Pavlick, E. (2019). BERT Rediscovers the Classical NLP Pipeline. In *Association for Computational Linguistics*.

Thomas, P. (2014). Bias in natural actor-critic algorithms. In *International conference on machine learning*, pages 441–448.

Tibshirani, R. (1994). Regression Shrinkage and Selection Via the Lasso. *Journal of the Royal Statistical Society, Series B*, 58:267–288.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention Is All You Need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017*. arXiv: 1706.03762.

Vinyals, O., Kaiser, L., Koo, T., Petrov, S., Sutskever, I., and Hinton, G. (2015). Grammar as a foreign language. In *Advances in Neural Information Processing Systems*, pages 2773–2781.

Vinyals, O. and Le, Q. (2015). A neural conversational model. *arXiv preprint arXiv:1506.05869*.

Vogel, A. and Jurafsky, D. (2010). Learning to Follow Navigational Directions. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 806–814. Association for Computational Linguistics.

Waibel, A., Hanazawa, T., Hinton, G., Shikano, K., and Lang, K. J. (1989). Phoneme recognition using time-delay neural networks. *IEEE transactions on acoustics, speech, and signal processing*, 37(3):328–339.

Wang, S. I., Liang, P., and Manning, C. D. (2016). Learning Language Games through Interaction. In *Proceedings Of the 54th Annual Meeting of the Association for Computational Linguistics*. arXiv: 1606.02447.

Wang, W., Yan, M., and Wu, C. (2018). Multi-Granularity Hierarchical Attention Fusion Networks for Reading Comprehension and Question Answering. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, pages 1705–1714. Association for Computational Linguistics.

Warnell, G., Waytowich, N., Lawhern, V., and Stone, P. (2017). Deep TAMER: Interactive Agent Shaping in High-Dimensional State Spaces. In *Proceedings of 32nd AAAI Conference on Artificial Intelligence*. arXiv: 1709.10163.

Werbos, P. (1974). *Beyond regression : new tools for prediction and analysis in the behavioral sciences*. PhD thesis.

Weston, J., Bordes, A., Chopra, S., Rush, A. M., van Merriënboer, B., Joulin, A., and Mikolov, T. (2016). Towards AI-Complete Question Answering: A Set of Prerequisite Toy Tasks. In *International Conference on Learning Representations, {ICLR} 2016*.

Williams, A., Nangia, N., and Bowman, S. R. (2017). A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics, {NAACL-HLT} 2018*.

Williams, E. C., Gopalan, N., Rhee, M., and Tellex, S. (2018). Learning to Parse Natural Language to Grounded Reward Functions with Weak Supervision. In *2018 IEEE International Conference on Robotics and Automation, ICRA 2018, Brisbane, Australia, May 21-25, 2018*, pages 1–7.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256.

Wilson, A., Fern, A., and Tadepalli, P. (2012). A Bayesian Approach for Policy Learning from Trajectory Preference Queries. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 25*, pages 1133–1141.

Winograd, T. (1972). Understanding natural language. *Cognitive Psychology*, 3(1):1–191.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., and others (2016). Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *arXiv preprint arXiv:1609.08144*.

Wu, Y., Wu, Y., Gkioxari, G., and Tian, Y. (2018). Building Generalizable Agents with a Realistic and Rich 3d Environment. *arXiv:1801.02209 [cs]*. arXiv: 1801.02209.

Yang, Z., He, X., Gao, J., Deng, L., and Smola, A. (2016). Stacked Attention Networks for Image Question Answering. *the 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016*. arXiv: 1511.02274.

Yi, K., Wu, J., Gan, C., Torralba, A., Kohli, P., and Tenenbaum, J. B. (2018). Neural-Symbolic VQA: Disentangling Reasoning from Vision and Language Understanding. *Advances in Neural Information Processing Systems 31: NeurIPS 2018*. arXiv: 1810.02338.

Yu, H., Zhang, H., and Xu, W. (2018). Interactive Grounded Language Acquisition and Generalization in 2d Environment. In *International Conference on Learning Representations, ICLR 2018*.

Zadrozny, W. (1994). From compositional to systematic semantics. *Linguistics and Philosophy*, 17:329–342.

Zaremba, W. and Sutskever, I. (2015). Learning to Execute. In *2015 International Conference on Learning Representations*. arXiv: 1410.4615.

Zettlemoyer, L. S. and Collins, M. (2005). Learning to Map Sentences to Logical Form: Structured Classification with Probabilistic Categorial Grammars. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, UAI'05, pages 658–666.

Ziebart, B. D., Maas, A., Bagnell, J. A., and Dey, A. K. (2008). Maximum Entropy Inverse Reinforcement Learning. In *Proceedings of the 23rd {AAAI} Conference on Artificial Intelligence, {AAAI} 2008*, pages 1433–1438.

Appendix A

SUPPLEMENTARY MATERIAL FOR THE FIRST ARTICLE

A.1. Training Details

We trained the policy π_θ and the discriminator D_ϕ concurrently using RMSProp as the optimizer and Asynchronous Advantage Actor-Critic (A3C) (Mnih et al., 2016) as the RL method. A baseline predictor (see Appendix A.6 for details) was trained to predict the discounted return by minimizing the mean square error. The RMSProp hyperparameters were different for π_θ and D_ϕ, see Table A.1. A designated worker was used to train the discriminator (see Algorithm 1). Other workers trained only the policy (see Algorithm 2). We tried having all workers write to the replay buffer B that was used for the discriminator training and found that this gave the same performance as using (c, s) pairs produced by the discriminator worker only. We found it crucial to regularize the discriminator by clipping columns of all weights matrices to have the L2 norm of at most 1. In particular, we multiply incoming weights w_u of each unit u by $\min(1, 1/||w_u||_2)$ after each gradient update as proposed by Srivastava et al. (2014). We linearly rescaled the policy's rewards to the $[0; 0.1]$ interval for both RL and AGILE. When using RL with reward prediction we fetch a batch from the replay buffer and compute the extra gradient for every rollout.

For the exact values of hyperparameters for the GridLU-Relations task we refer the reader to Table A.1. The hyperparameters for GridLU-Arrangements were mostly the same, with the exception of the episode length and the rollout length, which were 45 and 30 respectively. For training the RL baseline for GridLU-Relations we used the same hyperparameter settings as for the AGILE policy.

A.2. GridLU Environment

The GridLU world is a 5×5 gridworld surrounded by walls. The cells of the grid can be occupied by blocks of 3 possible shapes (circle, triangle, and square) and 3 possible colors (red, blue, and green). The grid also contains an agent sprite. The agent may carry a block;

TABLE A.1. Hyperparameters for the policy and the discriminator for the GridLU-Relations task.

Group	Hyperparameter	Policy π_θ	Discriminator D_ϕ
RMSProp	learning rate	0.0003	0.0005
	decay	0.99	0.9
	ϵ	0.1	10^{-10}
	grad. norm threshold	40	25
	batch size	1	256
RL	rollout length	15	—
	episode length	30	—
	discount	0.99	—
	reward scale	0.1	—
	baseline cost	1.0	—
	reward prediction cost (when used)	1.0	—
	reward prediction batch size	4	—
	num. workers training π_θ	15	1
AGILE	size of replay buffer B	—	100000
	num. workers training D_ϕ	—	1
Regularization	entropy weight α	0.01	—
	max. column norm	—	1

when it does so, the agent sprite changes color[1]. When the agent is free, i.e. when it does not carry anything, it is able to enter cells with blocks. A free agent can pick a block in the cell where both are situated. An agent that carries a block cannot enter non-empty cells, but it can instead drop the block that it carries in any non-empty cell. Both picking up and dropping are realized by the INTERACT action. Other available actions are LEFT, RIGHT, UP and DOWN and NOOP. The GridLU agent can be seen as a cursor (and this is also how it is rendered) that can be moved to select a block or a position where the block should be released. Figure A.1 illustrates the GridLU world and its dynamics. We render the state of the world as a color image by displaying each cell as an 8×8 patch[2] and stitching these patches in a 56×56 image[3]. All neural networks take this image as an input.

A.3. Experiment Details

Every experiment was repeated 5 times and the average result is reported.

[1]We wanted to make sure the that world state is fully observable, hence the agent's carrying state is explicitly color-coded.
[2]The relatively high 8×8 resolution was necessary to let the network discern the shapes.
[3]The image size is 56×56 because the walls surrounding the GridLU world are also displayed.

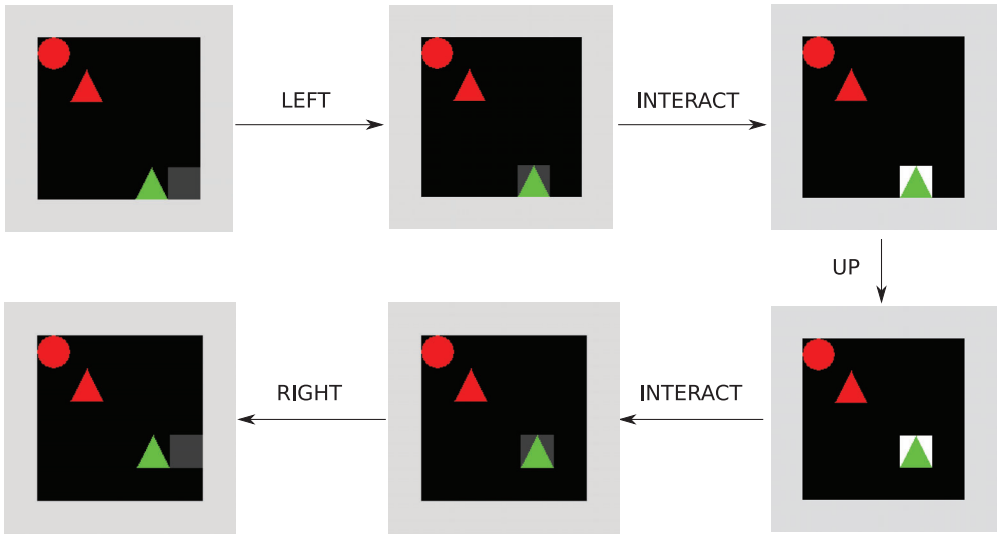


FIGURE A.1. The dynamics of the GridLU world illustrated by a 6-step trajectory. The order of the states is indicated by arrows. The agent's actions are written above arrows.

RL vs. AGILE

All agents were trained for $5 \cdot 10^8$ steps.

Data Efficiency

We trained AGILE policies with datasets $\mathcal{D}$ of different sizes for $5 \cdot 10^8$ steps. For each policy we report the maximum success rate that it showed in the course of training.

GridLU-Arrangements

We trained the agent for 100M time steps, saving checkpoints periodically, and selected the checkpoint that best fooled the discriminator according to the agent's internal reward.

Data Efficiency

We measure how many examples of instructions and goal-states are required by AGILE in order to understand the semantics of the GridLU-Relations instruction language. The results are reported in Figure A.2. The AGILE-trained agent succeeds in more than 50% of cases starting from 8000 examples, but as many as 130000 is required for the best performance.

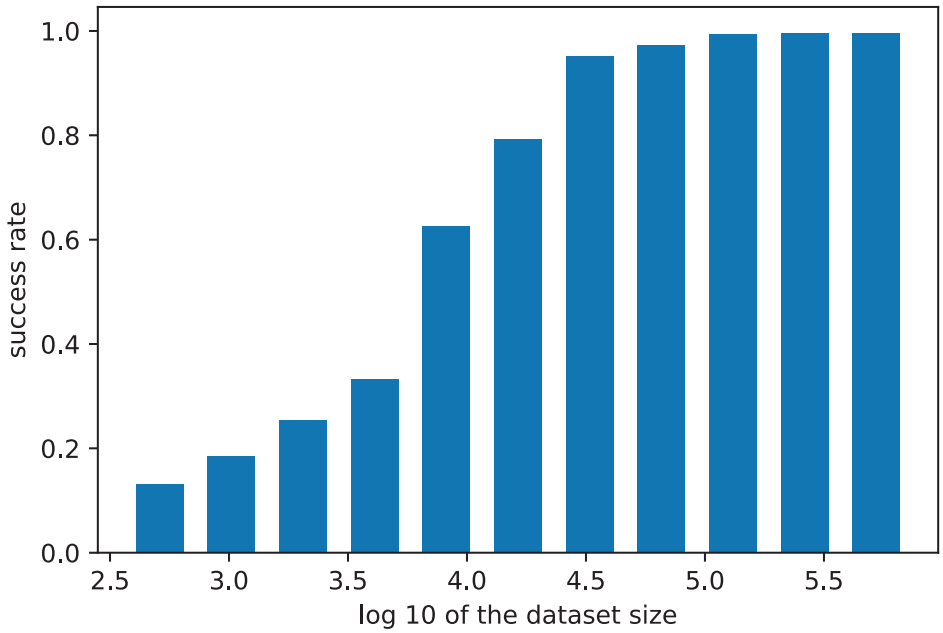


FIGURE A.2. Performance of AGILE for different sizes of the dataset of instructions and goal-states. For each dataset size of we report is the best average success rate over the course of training.

A.4. Analysis of the GridLU-Relations Task

GridLU Relations Instance Generator

All GridLU instructions can be generated from `<instruction>` using the following Backus-Naur form, with one exception: The first expansion of `<obj>` must not be identical to the second expansion of `<obj>` in `<bring_to_instruction>`.

```
<shape> ::= circle | rect | triangle
<color> ::= red | green | blue

<relation1> ::= NorthFrom | SouthFrom | EastFrom | WestFrom
<relation2> ::= <relation1> | SameLocation

<obj> ::= Color(<color>, <obj_part2>) | Shape(<shape>, SCENE)
<obj_part2> ::= Shape(<shape>, SCENE) | SCENE

<go_to_instruction> ::= <relation2>(AGENT, <obj>) | <relation2>(<obj>, AGENT)
<bring_to_instruction> ::= <relation1>(<obj>, <obj>)
<instruction> ::= <go_to_instruction> | <bring_to_instruction>
```

There are 15 unique possibilities to expand the nonterminal `<obj>`, so there are 150 unique possibilities to expand `<go_to_instruction>` and 840 unique possibilities to expand `<bring_to_instruction>` (not counting the exceptions mentioned above). Hence there are 990 unique instructions in total. However, several syntactically different instructions can be semantically equivalent, such as `EastFrom(AGENT, Shape(rect, SCENE))` and `WestFrom(Shape(rect, SCENE), AGENT)`.

Every instruction partially specifies what kind of objects need to be available in the environment. For go-to-instructions we generate one object and for bring-to-instructions we generate two objects according to this partial specification (unspecified shapes or colors are picked uniformly at random). Additionally, we generate one "distractor object". This distractor object is drawn uniformly at random from the 9 possible objects. All of these objects and the agent are each placed uniformly at random into one of 25 cells in the 5x5 grid.

The instance generator does not sample an instruction uniformly at random from a list of all possible instructions. Instead, it generates the environment at the same time as the instruction according to the procedure above. Afterwards we impose two 'sanity checks': are any two objects in the same location or are they all identical? If any of these two checks fail, the instance is discarded and we start over with a new instance.

Because of this rejection sampling technique, go-to-instructions are ultimately generated with approximately 25% probability even though they only represent $\approx 15\%$ of all possible instructions.

The number of different initial arrangements of three objects can be lower-bounded by $\binom{9}{3} = 2300$ if we disregard their permutation. Hence every bring-to-instruction has at least $K = 2300 \cdot 9 \approx 2 \cdot 10^4$ associated initial arrangements. Therefore the total number of task instances can be lower-bounded with $840 \cdot K \approx 1.7 \cdot 10^7$, disregarding the initial position of the agent.

Discriminator Evaluation

During the training on GridLU-Relations we compared the predictions of the discriminator with those of the ground-truth reward checker. This allowed us to monitor several performance indicators of the discriminator, see Figure A.3.

A.5. Analysis of the GridLU-Arrangements Task

Instruction Syntax

We used two types of instructions in the GridLU-Arrangements task, those referring only to the arrangement and others that also specified the color of the blocks. Examples

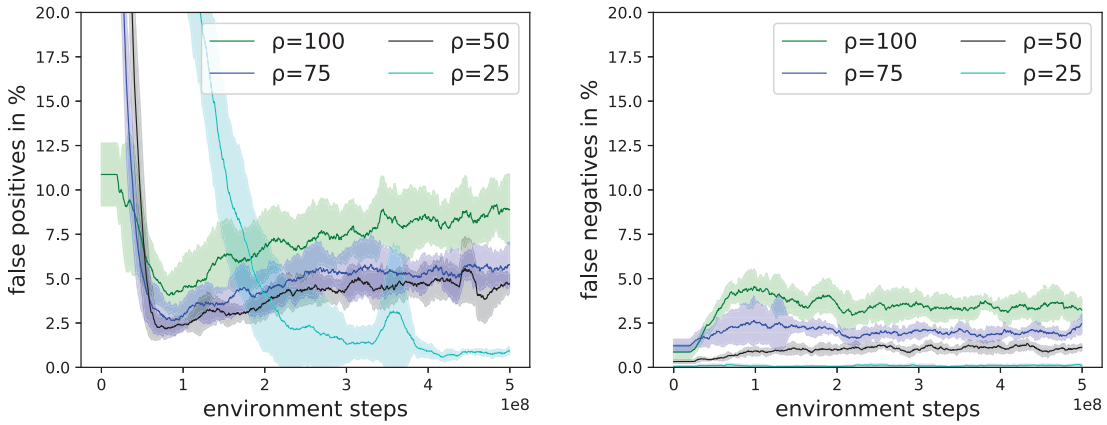


FIGURE A.3. The discriminator's errors in the course of training. **Left:** percentage of false positives. **Right:** percentage of false negatives.

Connected(AGENT, SCENE) and Snake(AGENT, Color('yellow', SCENE)) illustrate the syntax that we used for both instruction types.

Number of Distinct Goal-States

Table A.2 presents our computation of the number of distinct goal-states in the GridLU-Arrangements Task.

TABLE A.2. Number of unique goal-states in GridLU-Arrangements task.

Arrangement	**Possible arrangement positions**	**Possible colors**	**Possible agent positions**	**Possible distractor positions**	**Possible distractor colors**	**Total goal states**
Square	16	3	25	5985	2	14,364,000
Line	40	3	25	5985	2	35,910,000
Dline	8	3	25	5985	2	7,182,000
Triangle	48	3	25	5985	2	43,092,000
Circle	9	3	25	5985	2	8,079,750
Eel	48	3	25	5985	2	43,092,000
Snake	48	3	25	5985	2	43,092,000
Connected	200	3	25	5985	2	179,550,000
Disconnected	17	3	25	5985	2	15,261,750
Total						**389M**

A.6. MODELS

In this section we explain in detail the neural architectures that we used in our experiments. We will use $*$ to denote convolution, $\odot$, $\oplus$ to denote element-wise addition of a vector to a 3D tensor with broadcasting (i.e. same vector will be added/multiplied at each location

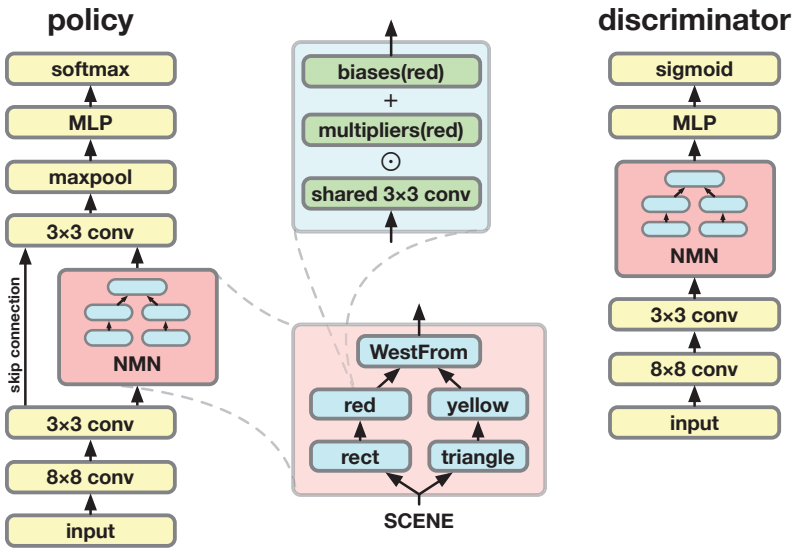


FIGURE A.4. Our policy and discriminator networks with a Neural Module Network (NMN) as the core component. The NMN's structure corresponds to an instruction *WestFrom(Color('red', Shape('rect', SCENE)), Color('yellow', Shape('triangle', SCENE)))*. The modules are depicted as blue rectangles. Subexpressions *Color('red', ...)*, *Shape('rect', ...)*, etc. are depicted as "red" and "rect" to save space. The bottom left of the figure illustrates the computation of a module in our variant of NMN.

of the feature map). We used ReLU as the nonlinearity in all layers with the exception of LSTM.

FiLM-NMN

We will first describe the FiLM-NMN discriminator D_ϕ. The discriminator takes a 56x56 RGB image s as the representation of the state. The image s is fed through a stem convnet that consisted of an $8x8$ convolution with 16 kernels and a 3x3 convolution with 64 kernels. The resulting tensor h_{stem} had a 5x5x64 shape.

As a Neural Module Metwork (Andreas et al., 2016), the FiLM-NMN is constructed of modules. The module m_x corresponding to a token x takes a left-hand side input h_l and a right-hand side input h_r and performs the following computation with them:

$$m_x(h_l, h_r) = ReLU((1 + \gamma_x) \odot (W_m * [h_l; h_r]) \oplus \beta_x), \tag{A.6.1}$$

where γ_x and β_x are FiLM coefficients (Perez et al., 2017) corresponding to the token x, W_m is a weight tensor for a 3x3 convolution with 128 input features and 64 output features. Zero-padding is used to ensure that the output of m_x has the same shape as h_l and h_r. The equation above describes a binary module that takes two operands. For the unary modules that received only one input (e.g. m_{red}, m_{square}) we present the input as h_l and zeroed out

h_r. This way we are able to use the same set of weights W_m for all modules. We have 12 modules in total, 3 for color words, 3 for shape words, 5 for relations words and one m_{AGENT} module used in go-to instructions. The modules are selected and connected based on the instructions, and the output of the root module is used for further processing. For example, the following computation would be performed for the instruction c_1 =NorthFrom(Color('red', Shape('circle', SCENE)), Color('blue', Shape('square', SCENE))):

$$h_{nmn} = m_{NorthFrom}(m_{red}(m_{circle}(h_{stem})), m_{blue}(m_{square}(h_{stem}))), \tag{A.6.2}$$

and the following one for c_2 =NorthFrom(AGENT, Shape('triangle', SCENE)):

$$h_{nmn} = m_{NorthFrom}(m_{AGENT}(h_{stem}), m_{triangle}(h_{stem})). \tag{A.6.3}$$

Finally, the output of the discriminator is computed by max-pooling the output of the FiLM-NMN across spatial dimensions and feeding it to an MLP with a hidden layer of 100 units:

$$D(c, s) = \sigma(w^T ReLU(W \text{maxpool}(h_{nmn}) + b)), \tag{A.6.4}$$

where w, W and b are weights and biases, $\sigma(x) = e^x/(1 + e^x)$ is the sigmoid function.

The policy network π_ϕ is similar to the discriminator network D_θ. The only difference is that (1) it outputs softmax probabilites for 5 actions instead of one real number (2) we use an additional convolutional layer to combine the output of FiLM-NMN and h_{stem}:

$$h_{merge} = ReLU(W_{merge} * [h_{nmn}; h_{stem}] + b_{merge}), \tag{A.6.5}$$

$$\pi(c, s) = \text{softmax}(W_2 ReLU(W_1 \text{maxpool}\,(h_{merge}) + b_1) + b_2), \tag{A.6.6}$$

the output h_{merge} of which is further used in the policy network instead of h_{nmn}.

Figure A.4 illustrates our FiLM-NMN policy and discriminator networks.

FiLM-LSTM

For our structure-agnostic models we use an LSTM of 100 hidden units to predict FiLM biases and multipliers for a 5 layer convnet. More specifically, let h_{LSTM} be the final state of the LSTM after it consumes the instruction c. We compute the FiLM coefficients for the layer $k \in [1; 5]$ as follows:

$$\gamma_k = W_k^\gamma h_{LSTM} + b_k^\gamma, \tag{A.6.7}$$

$$\beta_k = W_k^\beta h_{LSTM} + b_k^\beta, \tag{A.6.8}$$

and use them as described by the equation below:

$$h_k = ReLU((1 + \gamma_k) \odot (W_k * h_{k-1}) \oplus \beta_k), \tag{A.6.9}$$

where W_k are the convolutional weights, h_0 is set to the pixel-level representation of the world state s. The characteristics of the 5 layers were the following: (8x8, 16, VALID), (3x3, 32, VALID), (3x3, 64, SAME), (3x3, 64, SAME), (3x3, 64, SAME), where (mxm, n_{out}, p) stands for a convolutional layer with mxm filters, n_{out} output features, and $p \in \{\text{SAME}, \text{VALID}\}$ padding strategy. Layers with p = VALID do not use padding, whereas in those with p = SAME zero padding is added in order to produce an output with the same shape as the input. The layer 5 is also connected to layer 3 by a residual connection. Similarly to FiLM-NMN, the output h_5 of the convnet is max-pooled and fed into an MLP with 100 hidden units to produce the outputs:

$$D(c, s) = \sigma(w^T ReLU(W maxpool(h_5) + b)), \tag{A.6.10}$$

$$\pi(c, s) = \text{softmax}(W_2 ReLU(W_1 \text{maxpool}(h_5) + b_1) + b_2). \tag{A.6.11}$$

Baseline prediction

In all policy networks the baseline predictor is a linear layer that took the same input as the softmax layer. The gradients of the baseline predictor are allowed to propagate through the rest of the network.

Reward prediction

We use the result $h_{maxpool}$ of the max-pooling operation (which was a part of all models that we considered) as the input to the reward prediction pathway of our model. $h_{maxpool}$ is fed through a linear layer and softmax to produce probabilities of the reward being positive or zero (the reward is never negative in AGILE).

Weight Initialization

We use the standard initialisation methods from the Sonnet library[4]. Bias vectors are initialised with zeros. Weights of fully-connected layers are sampled from a truncated normal distribution with $\sigma = \frac{1}{\sqrt{n_{in}}}$, where n_{in} is the number of input units of the layer. Convolutional weights are sampled from a truncated normal distribution with $\sigma = \frac{1}{\sqrt{fan_{in}}}$, where fan_{in} is the product of kernel width, kernel height and the number of input features.

[4]https://github.com/deepmind/sonnet/

Appendix B

SUPPLEMENTARY MATERIAL FOR THE SECOND ARTICLE

B.1. MiniGrid Environments for OpenAI Gym

The environments used for this research are built on top of MiniGrid, which is an open source gridworld package. This package includes a family of reinforcement learning environments compatible with the OpenAI Gym framework. Many of these environments are parameterizable so that the difficulty of tasks can be adjusted (e.g. the size of rooms is often adjustable).

The World

In MiniGrid, the world is a grid of size NxN. Each tile in the grid contains exactly zero or one object, and the agent can only be on an empty tile or on a tile containing an open door. The possible object types are wall, door, key, ball, box and goal. Each object has an associated discrete color, which can be one of red, green, blue, purple, yellow and grey. By default, walls are always grey and goal squares are always green.

Reward Function

Rewards are sparse for all MiniGrid environments. Each environment has an associated time step limit. The agent receives a positive reward if it succeeds in satisfying an environment's success criterion within the time step limit, otherwise zero. The formula for calculating positive sparse rewards is $1 - 0.9 * (step\_count/max\_steps)$. That is, rewards are always between zero and one, and the quicker the agent can successfully complete an episode, the closer to 1 the reward will be. The *max_steps* parameter is different for each mission, and varies depending on the size of the environment (larger environments having a higher time step limit) and the length of the instruction (more time steps are allowed for longer instructions).

Action Space

There are seven actions in MiniGrid: turn left, turn right, move forward, pick up an object, drop an object, toggle and done. The agent can use the turn left and turn right action to rotate and face one of 4 possible directions (north, south, east, west). The move forward action makes the agent move from its current tile onto the tile in the direction it is currently facing, provided there is nothing on that tile, or that the tile contains an open door. The agent can open doors if they are right in front of it by using the toggle action.

Observation Space

Observations in MiniGrid are partial and egocentric. By default, the agent sees a square of 7x7 tiles in the direction it is facing. These include the tile the agent is standing on. The agent cannot see through walls or closed doors. The observations are provided as a tensor of shape 7x7x3. However, note that these are not RGB images. Each tile is encoded using 3 integer values: one describing the type of object contained in the cell, one describing its color, and a state indicating whether doors are open, closed or locked. This compact encoding was chosen for space efficiency and to enable faster training. The fully observable RGB image view of the environments shown in this paper is provided for human viewing.

Appendix C

SUPPLEMENTARY MATERIAL FOR THE THIRD ARTICLE

C.1. Experiment Details

We trained all models by minimizing the cross entropy loss $\log p(y|x, q)$ on the training set, where $y \in \{\text{yes}, \text{no}\}$ is the correct answer, x is the image, q is the question. In all our experiments we used the Adam optimizer (Kingma and Ba, 2015) with hyperparameters $\alpha = 0.0001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-10}$. We continuously monitored validation set performance of all models during training, selected the best one and reported its performance on the test set. The number of training iterations for each model was selected in preliminary investigations based on our observations of how long it takes for different models to converge. This information, as well as other training details, can be found in Table C.1.

Table C.1. Training details for all models. The subsampling factor is the ratio between the original spatial dimensions of the input image and those of the representation produced by the stem. It is effectively equal to 2^k, where k is the number of 2x2 max-pooling operations in the stem.

model	stem layers	subsampling factor	iterations	batch size
FiLM	6	4	200000	64
MAC	6	4	100000	128
Conv+LSTM	6	4	200000	128
RelNet	8	8	500000	64
NMN (Residual)	6	4	50000	64
NMN (Find)	6	4	200000	64
Stochastic NMN (Residual)	6	4	200000	64
Stochastic NMN (Find)	6	4	200000	64
Attention NMN (Find)	6	4	50000	64

C.2. Additional Results for MAC Model

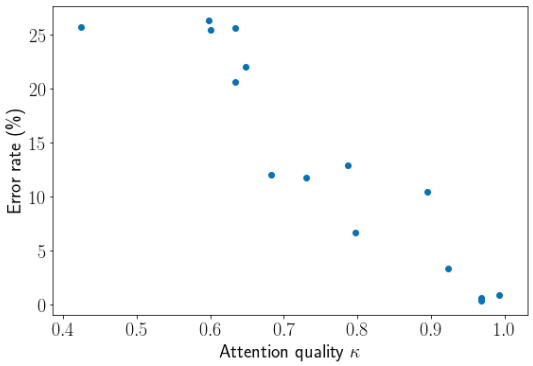
We performed an ablation study in which we varied the number of MAC units, the model dimensionality and the level of weight decay for the MAC model. The results can be found in Table C.2.

Table C.2. Results of an ablation study for MAC. The default model has 12 MAC units of dimensionality 128 and uses no weight decay. For each experiment we report means and standard deviations based on 5 repetitions.

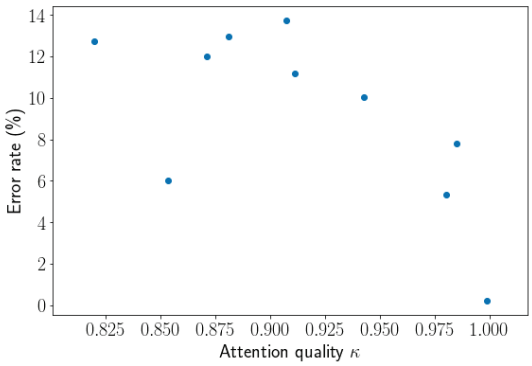
model	#rhs/lhs	train error rate (%)	test error rate (%)
default	1	0.17 ± 0.21	13.67 ± 9.97
1 unit	1	0.27 ± 0.35	28.67 ± 1.91
2 units	1	0.23 ± 0.13	24.28 ± 2.05
3 units	1	0.16 ± 0.15	26.47 ± 1.12
6 units	1	0.18 ± 0.17	20.84 ± 5.56
24 units	1	0.04 ± 0.05	9.11 ± 7.67
dim. 64	1	0.27 ± 0.33	23.61 ± 6.27
dim. 256	1	0.00 ± 0.00	4.62 ± 5.07
dim. 512	1	0.02 ± 0.04	8.37 ± 7.45
weight decay 0.00001	1	0.20 ± 0.23	19.21 ± 9.27
weight decay 0.0001	1	1.00 ± 0.54	31.19 ± 0.87
weight decay 0.001	1	40.55 ± 1.35	45.11 ± 0.74

We also perform qualitative investigations to understand the high variance in MAC's performance. In particular, we focus on control attention weights (c) for each run and aim to understand if runs that generalize have clear differences when compared to runs that failed. Interestingly, we observe that in successful runs each word $w \in \mathrm{X}, \mathrm{Y}$ has a unit that is strongly focused on it. To present our observations in quantitative terms, we plot attention quality $\kappa = \min_{w \in \{X,Y\}} \max_{k \in [1;12]} \alpha_{k,w}/(1 - \alpha_{k,R})$, where α are control scores vs accuracy in Figure C.1 for each run (see Section 8.4.3.2 for an explanation of κ). We can clearly see a positive correlation between κ and error rate, especially for low #rhs/lhs.

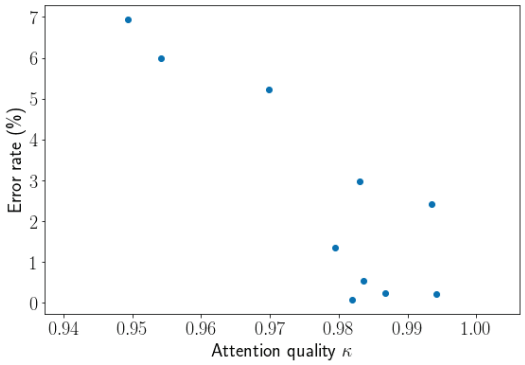
Next, we experiment with a *hard-coded* variation of MAC. In this model, we use hard-coded control scores such that given a SQOOP question X R Y, the first half of all modules focuses on X while the second half focuses on Y. The relationship between MAC and hardcoded MAC is similar to that between NMN-Tree and end-to-end NMN with parameterization induction. However, this model has not performed as well as the successful runs of MAC. We hypothesize that this could be due to the interactions between the control scores and the visual attention part of the model.



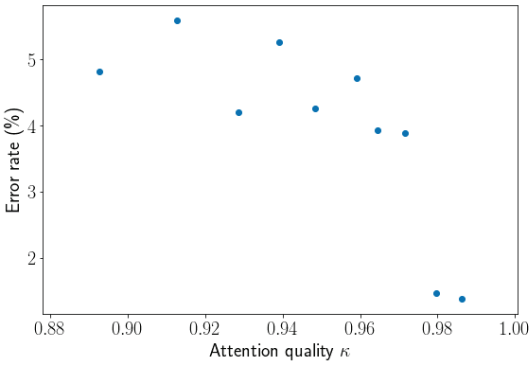
(A) 1 rhs/lhs



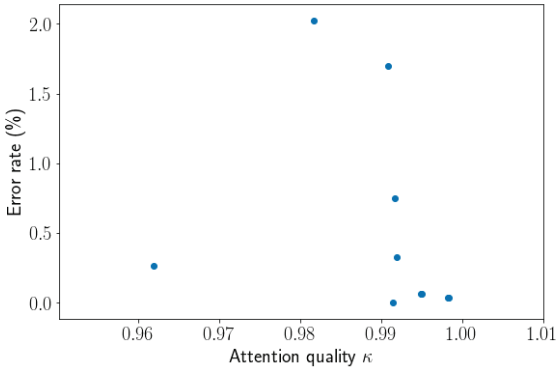
(B) 2 rhs/lhs



(C) 4 rhs/lhs



(D) 8 rhs/lhs



(E) 18 rhs/lhs

FIGURE C.1. Model test accuracy vs κ for the MAC model on different versions of SQOOP. All experiments are run 10 times with different random seeds. We can observe a clear correlation between κ and error rate for 1, 2 and 4 rhs/lhs. Also note that perfect generalization is always associated with κ close to 1.

C.3. Investigation of Correct Predictions with Spurious Layouts

In Section 8.4.3.1 we observed that an NMN with the Residual module can answer test questions with a relative low error rate of $1.64 \pm 1.79\%$, despite being a mixture of a tree and a chain (see results in Table 8.1, $p_0(tree) = 0.5$). Our explanation for this phenomenon is as follows: when connected in a tree, modules of such spurious models generalize well, and when connected as a chain they generalize poorly. The output distribution of the whole model is thus a mixture of the mostly correct $p(y|T = T_{tree}, x, q)$ and mostly random $p(y|T = T_{chain}, x, q)$. We verify our reasoning by explicitly evaluating test accuracies for $p(y|T = T_{tree}, x, q)$ and $p(y|T = T_{chain}, x, q)$, and find them to be around 99% and 60% respectively, confirming our hypothesis. As a result the predictions of the spurious models with $p(tree) \approx 0.5$ have lower confidence than those of sharp tree models, as indicated by the high log loss of 0.27 ± 0.04. We visualize the progress of structure induction for the Residual module with $p_0(tree) = 0.5$ in Figure 8.5 which shows how $p(tree)$ saturates to 1.0 for #rhs/lhs=18 and remains around 0.5 when #rhs/lhs=1.