Learning from Language

Jacob Andreas

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2018-141
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2018/EECS-2018-141.html

November 28, 2018

Copyright © 2018, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

Learning from Language

by

Jacob Andreas

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Dan Klein, Chair
Trevor Darrell
Tom Griffiths
Michael I Jordan

Fall 2018

Learning from Language

Copyright 2018
by
Jacob Andreas

Abstract

Learning from Language

by

Jacob Andreas

Doctor of Philosophy in Computer Science

University of California, Berkeley

Dan Klein, Chair

This dissertation explores the use of linguistic structure to inform the structure and parameterization of machine learning models for language processing and other applications. We introduce models for several tasks—question answering, instruction following, image classification, and programming by demonstration—all built around the common intuition that the compositional structure of the required predictors is reflected in the compositional structure of the language that describes them.

We begin by presenting a class of models called *neural module networks* (NMNs) and their application to natural language question answering problems. NMNs are designed to simultaneously exploit the representational capacity of deep networks and the compositional linguistic structure of questions, in order to target question answering applications not well supported by standard logical approaches. Our approach decomposes questions into their linguistic substructures, and uses these structures to dynamically instantiate question-specific networks built from an inventory of reusable modules. The resulting compound networks are jointly trained. We evaluate our approach on datasets for question answering backed by images and structured knowledge bases.

Next, we apply the same modeling principles to family of policy learning problems. We describe a framework for multitask reinforcement learning guided by *policy sketches*. Sketches annotate each task with a sequence of named subtasks, providing information about high-level structural relationships among tasks, but not the detailed guidance required by previous work on learning policy abstractions for reinforcement learning (e.g. intermediate rewards, subtask completion signals, or intrinsic motivations). Our approach associates every subtask with its own modular subpolicy, and jointly optimizes over full task-specific policies by tying parameters across shared subpolicies. Experiments illustrate two main advantages of this approach: first, it outperforms standard baselines that learn task-specific or shared monolithic policies; second, it naturally induces a library of primitive behaviors that can be recombined to rapidly acquire policies for new tasks.

The final two chapters explore ways of using information from language the context of less explicitly structured models. First, we exhibit a class of problems in which the space of

natural language strings provides a *parameter* space that captures natural task structure. We describe an approach that, in a pretraining phase, learns a language interpretation model that transforms inputs (e.g. images) into outputs (e.g. labels) given natural language descriptions. To learn a new concept (e.g. a classifier), we then propose to search directly in the space of descriptions to minimize the interpreter's loss on training examples. We then show that a related technique can be used to generate *explanations* of model behaviors: using the core insight that learned representations and natural language utterances carry the same meaning when they induce the same distribution over observations, we are able to automatically translate learned communication protocols into natural language.

To L.

Contents

Contents **ii**

List of Figures **iv**

List of Tables **v**

1 Introduction **1**

2 Module Networks: Language and Reasoning **5**
2.1 Motivations 6
2.2 Deep Networks as Functional Programs 7
2.3 Related Work 9
2.4 Model 10
2.5 Experiments 16
2.6 Discussion 19

3 Policy Sketches: Language and Behavior **22**
3.1 Related Work 24
3.2 Learning Modular Policies from Sketches 26
3.3 Experiments 31
3.4 Discussion 35

4 Latent Descriptions: Language and Learning **37**
4.1 Background 38
4.2 Learning with Language 40
4.3 Model and Training Details 41
4.4 Few-shot Classification 42
4.5 Programming by Demonstration 45
4.6 Policy Search 48
4.7 Other Related Work 52
4.8 Discussion 53

5 Translating Neuralese: Language and Belief **54**

5.1 Related Work . . . 56
5.2 Problem Formulation . . . 57
5.3 What's in a Translation? . . . 59
5.4 Translation Models . . . 61
5.5 Modeling details . . . 63
5.6 Belief and Behavior . . . 64
5.7 Evaluation . . . 66
5.8 Results . . . 69
5.9 Discussion . . . 71

6 Conclusion **73**

A Policy Sketches **75**

B Latent Descriptions **77**
B.1 Examples: ShapeWorld . . . 77
B.2 Examples: Navigation . . . 78

C Translating Neuralese **79**
C.1 Proof of Proposition 1 . . . 79

Bibliography **81**

List of Figures

2.1 Neural module network overview 6
2.2 Simple neural module networks 8
2.3 Generation of layout candidates 13
2.4 Sample NMN outputs for the visual question answering task 17
2.5 Example NMN predictions on the GeoQA dataset 20

3.1 Learning from policy sketches 22
3.2 Modular policy overview 28
3.3 Examples from the crafting and cliff environments 32
3.4 Comparing modular learning from sketches with standard RL baselines 32
3.5 Training details in the crafting domain 34

4.1 Example of L^3 on a binary image classification task 38
4.2 Formulation of the L^3 learning problem 39
4.3 The few-shot image classification task 42
4.4 Example predictions for image classification 45
4.5 Example string editing task 46
4.6 Example L^3 predictions for string editing 48
4.7 More example predictions for regular expressions. 49
4.8 Example treasure hunting task 50
4.9 Treasure hunting evaluation 52

5.1 Example interaction between a pair of agents in a deep communicating policy . 55
5.2 Overview of translation approach 56
5.3 Schematic representation of communication games 58
5.4 Cell implementing a single step of agent communication 59
5.5 Simplified game representation used for analysis 64
5.6 Tasks used to evaluate the translation model 67
5.7 Generated translations for color task 70
5.8 Generated translations for driving task 71

List of Tables

2.1 Module implementations 12
2.2 Results on the VQA test server 18
2.3 Results on the GeoQA dataset 19

3.1 Accuracy and generalization of learned models in the crafting domain 36

4.1 Evaluation of L^3 on image classification 44
4.2 Evaluation of L^3 on string editing 47
4.3 Inference and representation experiments for string editing 47

5.1 Evaluation results for reference games 69
5.2 Belief evaluation results for the driving game 70
5.3 Behavior evaluation results for the driving game 71

Acknowledgments

Thanks:

To the Berkeley natural language processing group—David Hall, for lots of early conversations about how to approach the Ph.D. effectively (and for being included in this classification); Jonathan Kummerfeld and Greg Durrett, for companionship in foreign cities and my real education in how to give a good talk; Taylor Berg-Kirkpatrick, for helping me out of a tight spot in Culver City and teaching me most of what I know about structured prediction; Daniel Fried and Mitchell Stern, for helping me forget everything I know about structured prediction; Max Rabinovich, for solutions to a few key problems on and off the boulders; and finally Nikita Kitaev and David Gaddy, for stimulating discussions in the last two years.

To my other collaborators—this is a dissertation about language processing, but the work here has benefited to an unusual extent from close interaction with colleagues in computer vision and robotics. I want to specifically thank Trevor Darrell, Sergey Levine, Anca Dragan, and above all Marcus Rohrbach and Ronghang Hu. But more generally, this work would not have been possible without the environment in Sutardja Dai Hall that grew into the Berkeley AI Research Lab, and from which all these collaborations emerged effortlessly.

To the department staff—especially Xuan Quach, Shirley Salanio, and most of all Angie Abbatecola, whose competence is matched only by patience.

To all my mentors outside of Berkeley—Kathy McKeown, for starting me down this path; Owen Rambow and Nizar Habash, for teaching me how to think about structure in language; Michael Collins and Steve Clark for teaching me how to think about meaning.

To my friends and family—my parents and sister, who I've been lucky to have so close for the last five years; David, who has always been there to talk; Arvind, Will, Gili, Ben, Hannah, Marc and Judy, for a little patch of Storey's way here in California; Dylan, Robert, Eric, Lisa, Sandy, Will, George and Moses, for the company. And of course to Leor, whose love, care and support have made all the difference as I've reached the end of this process.

Finally, to Dan—in my early years, for serving as a sounding board and a repository of knowledge about every question around language and probabilistic inference; for teaching me to read a paper and present one. And later, for help with the bigger questions: how to advise a student, design a lecture, and, at this time of change in the field, how to identify the techniques and ideas that will matter in the long run.

Anyone who understands me eventually recognizes [these propositions] as nonsensical, when he has used them—as steps—to climb beyond them. (He must, so to speak, throw away the ladder after he has climbed up it.)

Ludwig Wittgenstein,
Tractatus Logico-Philosophicus

Those masterful images because complete
Grew in pure mind but out of what began?

[. . .] Now that my ladder's gone
I must lie down where all the ladders start

W.B. Yeats,
"The Circus Animals' Desertion"

Chapter 1

Introduction

The structure of language reflects the structure of the world: the named concepts and relations with which we describe our environment provide a rich source of information about the kinds of abstractions we use to interact with it [Gopnik and Meltzoff, 1987]. In many machine learning problems, efficient automatic discovery of reusable discrete operators for perception and reasoning remains a major challenge [Dietterich, 2000, Ferrari and Zisserman, 2008, Kulkarni et al., 2016]. This dissertation investigates ways in which language might supply them. We are motivated by a variety of prediction problems: *question answering*, in which we attempt to map from natural language strings and environment states to answers; *instruction following*, in which language describes a goal or policy for an automated agent; and more general learning problems like *multitask classification* and *model explanation*, which may not involve text data as input or output, but in which side information from language might still provide a useful guide to problem structure.

To begin, consider the two examples of multitask classification and question answering. A classifier is a function from observations to labels. Model-theoretic approaches to semantics represent the meaning of a question as a function from world states to answers (discussed in more detail in Chapter 2). So we may treat both problems uniformly as involving a collection of tasks i, each associated with:

- a *specification* s_i—in multitask classification, a collection of (observation, label) pairs; in question answering, a natural language question
- a *data distribution* $p_i(X, Y)$—in multitask classification, observations and labels; in question answering, world states and answers to the question s_i in each such state
- a loss function ℓ_i

In each case our goal is to identify a mapping $M : s_i \mapsto f_i$ from specifications to *predictors* of Y given X that minimize expected loss:

$$\sum_i \mathbb{E}_i \ell_i(M(s_i)(X), Y) = \sum_i \mathbb{E}_i \ell_i(f_i(X), Y) \ . \tag{1.1}$$

That is: we can think of a question answering task not as a supervised learning problem of mapping from questions and worlds to answers, but rather as a collection of indirectly supervised classification problems. (Similarly, we can think of instruction following as a collection of indirectly supervised policy-learning problems.) Whether s_i takes the form of a training dataset or a natural language string, our hope in both settings is that shared structure among specifications and their associated predictors makes it possible to identify a good general-purpose inference procedure m. With this view in mind, we can more concisely state the goals of this dissertation—we are interested in problems where, in Equation 1.1,

(1) Predictors are *compositional*: there is some generating set of primitive functions and composition operations such that every f_i can be constructed by composing a small number of primitives.

(2) Predictors are *language-like*: when a specification s_i takes the form of a natural language utterance, the structure of the associated predictor f_i reflects the linguistic structure of s_i.

As we will see, a variety of problems—not just in natural language processing but also computer vision and policy learning—exhibit these properties.

Our central claim is that when properties (1) and (2) hold, it helps to model them explicitly: the right primitive functions are those we can describe with *words*, and the right compositional operators are the ones we use in formal representations of language. For language understanding problems, model structure imitates linguistic structure. And for downstream machine learning problems that do not themselves have anything to do with language, side information from language can help us discover structure in learned models, or supply structure at training time. We investigate this claim empirically—we present a collection of compositional models that can flexibly incorporate linguistic structure during training, evaluation, or both. Along the way, we aim at answers to a few broader questions:

What Can Language Tell Us About Model Design?

Questions about how to model the world—at what level of granularity? with what kind of explicit treatment of objects, properties and events?—remain an active topic of research in computer vision and robotics [Chang et al., 2016, Battaglia et al., 2016]. Though targeted at very different applications, a (millennia-old! [Sharma, 2003]) tradition of research on structured representations of language and linguistic meaning has encountered many of the same questions: most basically, the question of how the surface form of language relates to meaning. But also: with respect to what kind of world representation is linguistic meaning defined? What structure do we need to assume in order to compactly and coherently represent the data produced by language users?

A long line of work on *semantic parsing*, especially grounded in problems like question answering and instruction following, represents a first step in this direction. But such approaches have typically bypassed questions of learning how to represent the world: instead,

they pre-commit to a hand-designed representation and build an analysis of language on top of it [Wong and Mooney, 2006, Liang et al., 2011, Kwiatkowski et al., 2013]. This dissertation aims to bridge the gap, and learn to improved models for the underlying perception and control problems jointly with linguistic analysis. We investigate whether representational theories from language might be useful in this broader learning context. Specifically,

- At a low level, can formal (compositional) representations of sentence meaning help us learn reusable operators for perception and reasoning? Is this process helped by parameter tying with structure inherited from logical structure in language? (Chapter 2)
- Can these same kinds of structured parameter tying schemes help us as we move to problems where we don't really care about language processing, but just about success at some downstream learning task (e.g. policy learning)? (Chapter 3 and Chapter 4)
- To the extent that current learning techniques are capable of solving some of the problems we care about without side information from language, do they solve these problems in the same way? That is, does the functional basis described at the beginning of this section arise spontaneously from standard learning algorithms? How can we identify this structure when it does occur? (Chapter 5)

What Can Model Design Tell Us About Language?

Improved models for language processing might also shed light on a few questions about the acquisition and representation language itself. Considerable effort in computer science, cognitive science and linguistics has been devoted to the *symbol grounding* problem—how the words of a language acquire their meanings by grounding in the world [Harnad, 1990]. Several statistical models have been proposed with the specific goal of studying the symbol grounding problem, using either standard machinery from formal semantics [Krishnamurthy and Kollar, 2013] or other structured meaning representations [Tellex et al., 2011b].

But all existing approaches bottom out in simple functions (perhaps with a few free parameters) of a highly structured world model that has been pre-analyzed into individuals, their relationships and their properties. This is both *too much* structure, in the sense that at supplies an unrealistic inductive bias to learners [Prather and Bacon, 1986], but also *too little*, in the sense that decisions made *a priori* about how to represent the world are necessarily lossy and not guaranteed to reflect future uses of language. Ultimately, the world to which sentences refer is no more than the primitive elements of sensation and action. Then,

- What does a formal semantics look like for such a world? Does pushing some low-level perceptual distinctions into the weights of a learned model (e.g. the distinction between *tall man* and *tall building*) make it easier to design concise and broadly applicable representations of sentence meaning? What sorts of linguistic phenomena should be explicitly articulated in logical meaning representations, and what distinctions should be left to learning? (Chapter 2)

Going even further, we'll see in the final chapters of this dissertation that the denotational perspective on the representation of meaning in language provides a useful tool even independent of the logical meaning representations traditionally used to implement it—in particular, we can do useful things with denotational representations of meaning by learning a mapping directly from utterances to distributions over world states. In these cases:

- Can we specify a useful model-theoretic semantics without any logical forms at all? What questions can we answer about language by going straight to "pragmatic" representations of speaker meaning, without any explicit representation of sentence meaning? (Chapter 5)

The models we present in this thesis are aimed foremost at concrete language processing applications, and so the answers they provide to more general questions about language are at best incomplete. Nevertheless we hope that some of the representational tools we present here are useful beyond the immediate engineering problems.

Chapter 2

Module Networks: Language and Reasoning

We begin with an investigation of the extent to which structured representations of language can assist in the learning of reusable operators for perception and visual reasoning.[1] We focus in this chapter on *question answering* problems, which have long served as a test-bed for reasoning challenges in natural language processing [Winograd, 1972] and artificial intelligence more broadly. This chapter presents a compositional, attentional model for answering questions about a variety of world representations, including images and structured knowledge bases. The model translates from questions to dynamically assembled neural networks, then applies these networks to world representations (images or knowledge bases) to produce answers. We take advantage of two largely independent lines of work: on one hand, an extensive literature on answering questions by mapping from strings to logical representations of meaning; on the other, a series of recent successes in deep neural models for image recognition and captioning. By constructing neural networks instead of logical forms, our model leverages the best aspects of both linguistic compositionality and continuous representations.

Our model has two components, trained jointly: first, a collection of neural "modules" that can be freely composed (Figure 2.1a); second, a network layout predictor that assembles modules into complete deep networks tailored to each question (Figure 2.1b). Training data consists of (world, question, answer) triples: our approach requires no supervision of network layouts. We achieve state-of-the-art performance on two markedly different question

[1] Material in this chapter is adapted from:

- Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Neural module networks. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 2016.
- Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Learning to Compose Neural Networks for Question Answering. In *Meeting of the North American Association for Computational Linguistics*, 2016.

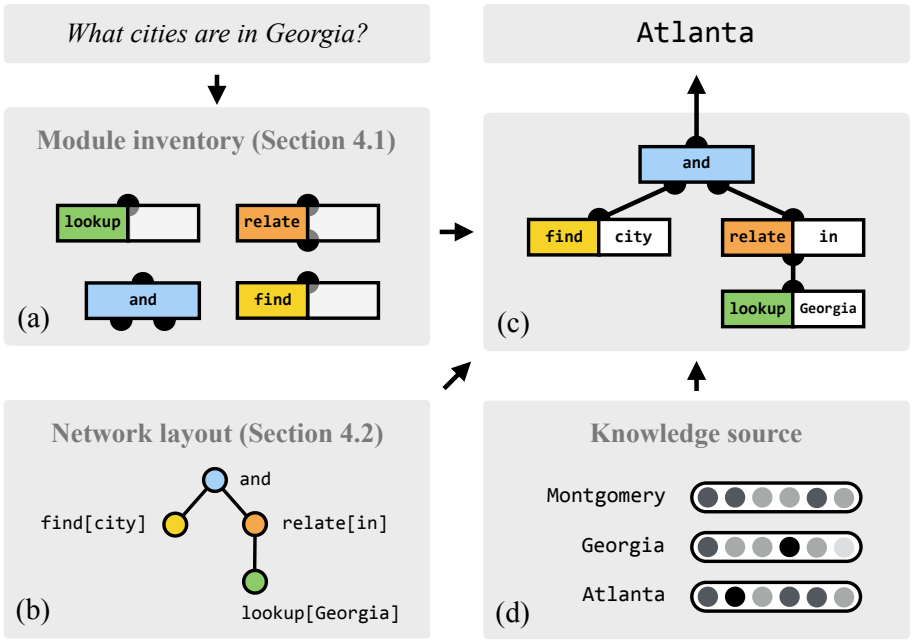


Figure 2.1: Model overview. A learned syntactic analysis (a) is used to assemble a collection of neural modules (b) into a deep neural network (c), and applied to a world representation (d) to produce an answer.

answering tasks: simple questions about natural images, and more compositional questions about United States geography backed by a tabular, rather than visual, world representation.

2.1 Motivations

We begin with a few observations. First, some flexible and general-purpose class of function approximators would appear to be indispensable for visual question answering: one of the major obstacles to integration of purely formal approaches to question answering with more complex world representations like images has been the difficulty of hand-designing a sufficiently expressive inventory of functional primitives [Malinowski et al., 2015]; in vision, especially, recent successes suggest that any kind of question-specific model we construct for visual processing should really be a question-specific neural network [Krizhevsky et al., 2012]. State-of-the-art performance on the full range of computer vision tasks that are studied requires a variety of different deep network topologies—there is no single "best network" for all tasks.

But second: though different networks are used for different purposes, it is commonplace to initialize systems for many of vision tasks with a prefix of a network trained for classification [Girshick et al., 2014]. This has been shown to substantially reduce training time and

improve accuracy. So while network structures are not *universal* (in the sense that the same network is appropriate for all problems), they are at least empirically *modular* (in the sense that intermediate representations for one task are useful for many others).

Can we generalize this idea in a way that is useful for question answering? Rather than thinking of question answering as a problem of learning a single function to map from questions and contexts to answers, it's perhaps useful to think of it as a highly-multitask learning setting, where each problem instance is associated with a novel task, and the identity of that task is expressed only noisily in language. In particular, where a simple question like *is this a truck?* requires us to retrieve only one piece of information from an image, more complicated questions, like *how many objects are to the left of the toaster?* might require multiple processing steps. The compositional nature of language means that the number of such processing such steps is potentially unbounded. Moreover, multiple *kinds* of processing might be required—repeated convolutions might identify a truck, but some kind of recurrent architecture is likely necessary to count up to arbitrary numbers.

Thus our goal in this chapter is to specify a framework for modular, composable, jointly-trained neural networks. In this framework, we first predict the structure of the computation needed to answer each question individually, then realize this structure by constructing an appropriately-shaped neural network from an inventory of reusable modules. These modules are learned jointly, rather than trained in isolation, and specialization to individual tasks (identifying properties, spatial relations, etc.) arises naturally from the training objective.

2.2 Deep Networks as Functional Programs

Consider the example shown in Figure 2.2. The question *What color is the bird?* might be answered in two steps: first, "where is the bird?" (Figure 2.2a), second, "what color is that part of the image?" (Figure 2.2c). This first step, a generic functional primitive or *module* that we will call `find`, can be expressed as a fragment of a neural network that maps from image features and a lexical item (here *bird*) to a distribution over pixels. This operation is commonly referred to as the *attention mechanism*, and is a standard tool for manipulating images [Xu et al., 2015] and text representations [Hermann et al., 2015].

Our first contribution is an extension and generalization of this mechanism to enable fully-differentiable reasoning about more structured semantic representations. Figure 2.2b shows how the same module can be used to focus on the entity *Georgia* in a non-visual grounding domain; more generally, by representing every entity in the universe of discourse as a feature vector, we can obtain a distribution over entities that corresponds roughly to a logical set-valued denotation.

Having obtained such a distribution, existing neural approaches use it to immediately compute a weighted average of image features and project back into a labeling decision—a `describe` module (Figure 2.2c). But the logical perspective suggests a number of novel modules that might operate on attentions: e.g. combining them (by analogy to conjunction or disjunction) or inspecting them directly without a return to feature space (by analogy to

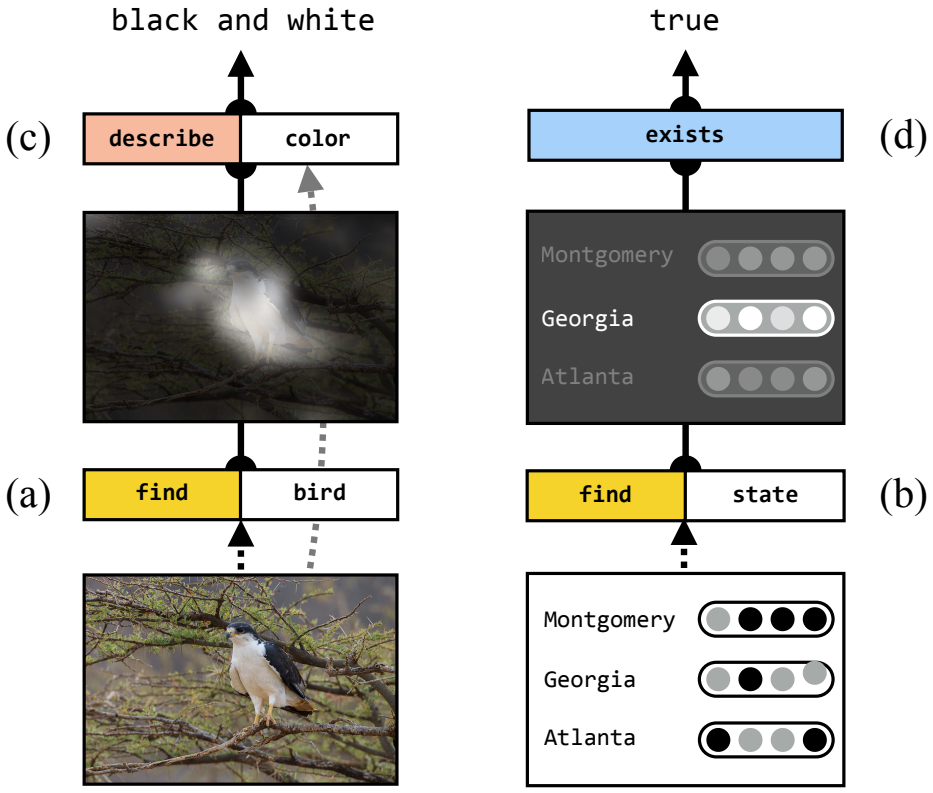


Figure 2.2: Simple neural module networks, corresponding to the questions *What color is the bird?* and *Are there any states?* (a) A neural `find` module for computing an attention over pixels. (b) The same operation applied to a knowledge base. (c) Using an attention produced by a lower module to identify the color of the region of the image attended to. (d) Performing quantification by evaluating an attention directly.

quantification, Figure 2.2d). These modules are discussed in detail in Section 2.4. Unlike their formal counterparts, they are differentiable end-to-end, facilitating their integration into learned models. Building on previous work, we learn behavior for a collection of heterogeneous modules from (world, question, answer) triples.

Our second contribution is an approach for learning to assemble such modules compositionally. Isolated modules are of limited use—to obtain expressive power comparable to either formal approaches or monolithic deep networks, they must be composed into larger structures. Figure 2.2 shows simple examples of composed structures, but for realistic question-answering tasks, even larger networks are required. Thus our goal is to automatically induce variable-free, tree-structured computation descriptors. We can use a familiar functional notation from formal semantics (e.g. Liang et al., 2011) to represent these computations.[2] We write the two examples in Figure 2.2 as

[2]But note that unlike formal semantics, the behavior of the primitive functions here is itself unknown.

```
(describe[color]
  find[bird])
```

and

```
(exists
  find[state])
```

respectively. These are *network layouts*: they specify a structure for arranging modules (and their lexical parameters) into a complete network. While it might be possible to use hand-written rules to deterministically transform dependency trees into layouts, we will likely be restricted to producing simple structures like the above for non-synthetic data. For full generality, we will need to solve harder problems, like transforming *What cities are in Georgia?* (Figure 2.1) into

```
(and
  find[city]
  (relate[in]
    lookup[Georgia]))
```

In this chapter we will present a model for learning to predict such structures jointly with the parameters that determine their low-level behavior. We call this model a *neural module network*.

2.3 Related Work

There is an extensive literature on database question answering, in which strings are mapped to logical forms, then evaluated by a black-box execution model to produce answers. Supervision may be provided either by annotated logical forms [Wong and Mooney, 2007, Kwiatkowski et al., 2010, Andreas et al., 2013] or from (world, question, answer) triples alone [Liang et al., 2011, Pasupat and Liang, 2015]. In general the set of primitive functions from which these logical forms can be assembled is fixed, but one recent line of work focuses on inducing new predicates functions automatically, either from perceptual features [Krishnamurthy and Kollar, 2013] or the underlying schema [Kwiatkowski et al., 2013]. The model we describe here has a unified framework for handling both the perceptual and schema cases, and differs from existing work primarily in learning a differentiable execution model with continuous evaluation results.

Neural models for question answering are also a subject of current interest. These include approaches that model the task directly as a multiclass classification problem [Iyyer et al., 2014], models that attempt to embed questions and answers in a shared vector space [Bordes et al., 2014] and attentional models that select words from documents sources [Hermann et al., 2015]. Such approaches generally require that answers can be retrieved directly based

on surface linguistic features, without requiring intermediate computation. A more structured approach described by Yin et al. [2017] learns a query execution model for database tables without any natural language component. Previous efforts toward unifying formal logic and representation learning include those of Grefenstette [2013], Krishnamurthy and Mitchell [2013], Lewis and Steedman [2013], and Beltagy et al. [2013].

The visually-grounded component of this work relies on recent advances in convolutional networks for computer vision [LeCun et al., 1998, Krizhevsky et al., 2012, Simonyan and Zisserman, 2014], and in particular the fact that late convolutional layers in networks trained for image recognition contain rich features useful for other vision tasks while preserving spatial information Donahue et al. [2014]. These features have been used for both image captioning [Xu et al., 2015] and visual question answering [Yang et al., 2017].

Most previous approaches to visual question answering either apply a recurrent model to deep representations of both the image and the question [Ren et al., 2015, Malinowski et al., 2015], or use the question to compute an attention over the input image, and then answer based on both the question and the image features attended to [Yang et al., 2017, Xu and Saenko, 2016]. Other approaches include the simple classification model described by Zhou et al. [2015] and the dynamic parameter prediction network described by Noh et al. [2016]. All of these models assume that a fixed computation can be performed on the image and question to compute the answer, rather than adapting the structure of the computation to the question.

Other approaches in this general family include the "universal parser" sketched by Bottou [2014], the graph transformer networks of Bottou et al. [1997], the knowledge-based neural networks of Towell and Shavlik [1994] and the recursive neural networks of Socher et al. [2013], which use a fixed tree structure to perform further linguistic analysis without any external world representation. We are unaware of previous work that simultaneously learns both parameters for and structures of instance-specific networks.

2.4 Model

Recall that our goal is to map from questions and world representations to answers. This process involves the following variables:

1. w a world representation
2. x a question
3. y an answer
4. z a network layout
5. θ a collection of model parameters

Our model is built around two distributions: a *layout model* $p(z|x;\theta_\ell)$ which chooses a layout for a sentence, and a *execution model* $p_z(y|w;\theta_e)$ which applies the network specified by z to w.

For ease of presentation, we introduce these models in reverse order. We first imagine that z is always observed, and in Section 2.4 describe how to evaluate and learn modules parameterized by θ_e within fixed structures. In Section 2.4, we move to the real scenario, where z is unknown. We describe how to predict layouts from questions and learn θ_e and θ_ℓ jointly without layout supervision.

Evaluating Modules

Given a layout z, we assemble the corresponding modules into a full neural network (Figure 2.1c), and apply it to the knowledge representation. Intermediate results flow between modules until an answer is produced at the root. We denote the output of the network with layout z on input world w as $[\![z]\!]_w$; when explicitly referencing the substructure of z, we can alternatively write $[\![m(h^1, h^2)]\!]$ for a top-level module m with submodule outputs h^1 and h^2. We then define the execution model:

$$p_z(y|w) = ([\![z]\!]_w)_y \tag{2.1}$$

(This assumes that the root module of z produces a distribution over labels y.) The set of possible layouts z is restricted by module *type constraints*: some modules (like `find` above) operate directly on the input representation, while others (like `describe` above) also depend on input from specific earlier modules. The base types are considered are Attention (a distribution over pixels or entities) and Labels (a distribution over answers).

Parameters are tied across multiple instances of the same module, so different instantiated networks may share some parameters but not others. Modules have both *parameter arguments* (shown in square brackets) and ordinary inputs (shown in parentheses). Parameter arguments, like the running `bird` example in Section 2.2, are provided by the layout, and are used to specialize module behavior for particular lexical items. Ordinary inputs are the result of computation lower in the network. In addition to parameter-specific weights, modules have global weights shared across all instances of the module (but not shared with other modules). We write $A, a, B, b, \ldots$ for global weights and u^i, v^i for weights associated with the parameter argument i. $\oplus$ and $\odot$ denote (possibly broadcast) elementwise addition and multiplication respectively. The complete set of global weights and parameter-specific weights constitutes θ_e. *Every* module has access to the world representation, represented as a collection of vectors $w^1, w^2, \ldots$ (or W expressed as a matrix). The nonlinearity σ denotes a rectified linear unit.

The modules used in this chapter are shown in Table 2.1, with names and type constraints in the first row and a description of the module's computation following.

Learning in this simplified setting is straightforward. Assuming the top-level module in each layout is a `describe` or `exists` module, the fully- instantiated network corresponds to a

Lookup ($\rightarrow$ Attention)

`lookup[`i`]` produces an attention focused entirely at the index $f(i)$, where the relationship f between words and positions in the input map is known ahead of time (e.g. string matches on database fields).

$$[\![\texttt{lookup[}i\texttt{]}]\!] = e_{f(i)} \tag{2.2}$$

where e_i is the basis vector that is 1 in the ith position and 0 elsewhere.

Find ($\rightarrow$ Attention)

`find[`i`]` computes a distribution over indices by concatenating the parameter argument with each position of the input feature map, and passing the concatenated vector through a MLP:

$$[\![\texttt{find[}i\texttt{]}]\!] = \text{softmax}(a \odot \sigma(Bv^i \oplus CW \oplus d)) \tag{2.3}$$

Relate (Attention $\rightarrow$ Attention)

`relate` directs focus from one region of the input to another. It behaves much like the `find` module, but also conditions its behavior on the current region of attention h. Let $\bar{w}(h) = \sum_k h_k w^k$, where h_k is the k^{th} element of h. Then,

$$[\![\texttt{relate[}i\texttt{]}(h)]\!] = \text{softmax}(a \ \odot \sigma(Bv^i \oplus CW \oplus D\bar{w}(h) \oplus e)) \tag{2.4}$$

And (Attention* $\rightarrow$ Attention)

`and` performs an operation analogous to set intersection for attentions. The analogy to probabilistic logic suggests multiplying probabilities:

$$[\![\texttt{and}(h^1, h^2, \ldots)]\!] = h^1 \odot h^2 \odot \cdots \tag{2.5}$$

Describe (Attention $\rightarrow$ Labels)

`describe[`i`]` computes a weighted average of w under the input attention. This average is then used to predict an answer representation. With $\bar{w}$ as above,

$$[\![\texttt{describe[}i\texttt{]}(h)]\!] = \text{softmax}(A\sigma(B\bar{w}(h) + v^i)) \tag{2.6}$$

Exists (Attention $\rightarrow$ Labels)

`exists` is the existential quantifier, and inspects the incoming attention directly to produce a label, rather than an intermediate feature vector like `describe`:

$$[\![\texttt{exists}](h)]\!] = \text{softmax}\Big(\big(\max_k h_k\big)a + b\Big) \tag{2.7}$$

Table 2.1: Module implementations: parameterizations of the various primitive functional types used to implement the models in this chapter.

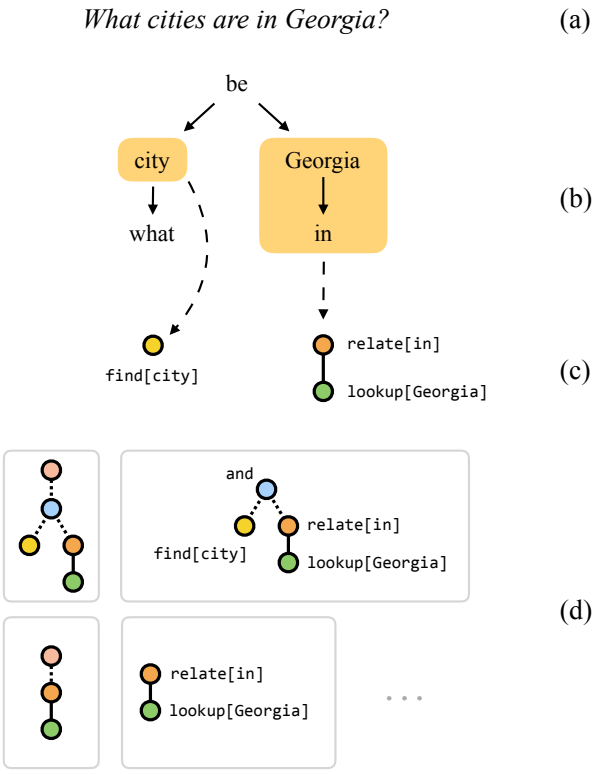


Figure 2.3: Generation of layout candidates. The input sentence (a) is represented as a dependency parse (b). Fragments of this dependency parse are then associated with appropriate modules (c), and these fragments are assembled into full layouts (d).

distribution over labels conditioned on layouts. To train, we maximize $\sum_{(w,y,z)} \log p_z(y|w;\theta_e)$ directly. This can be understood as a parameter-tying scheme, where the decisions about which parameters to tie are governed by the observed layouts z.

Assembling Networks

Next we describe the layout model $p(z|x;\theta_\ell)$. We first use a fixed syntactic parse to generate a small set of candidate layouts, analogously to the way a semantic grammar generates candidate semantic parses in previous work [Berant and Liang, 2014].

A semantic parse differs from a syntactic parse in two primary ways. First, lexical items must be mapped onto a (possibly smaller) set of semantic primitives. Second, these semantic primitives must be combined into a structure that closely, but not exactly, parallels the structure provided by syntax. For example, *state* and *province* might need to be identified with the same field in a database schema, while *all states have a capital* might need to be identified with the correct (*in situ*) quantifier scope.

While we cannot avoid the structure selection problem, continuous representations simplify the lexical selection problem. For modules that accept a vector parameter, we associate these parameters with *words* rather than semantic tokens, and thus turn the combinatorial optimization problem associated with lexicon induction into a continuous one. Now, in order to learn that *province* and *state* have the same denotation, it is sufficient to learn that their associated parameters are close in some embedding space—a task amenable to gradient descent. (Note that this is easy only in an optimizability sense, and not an information-theoretic one—we must still learn to associate each independent lexical item with the correct vector.) The remaining combinatorial problem is to arrange the provided lexical items into the right computational structure. In this respect, layout prediction is more like syntactic parsing than ordinary semantic parsing, and we can rely on an off-the-shelf syntactic parser to get most of the way there. In this work, syntactic structure is provided by the Stanford dependency parser [De Marneffe and Manning, 2008].

The construction of layout candidates is depicted in Figure 2.3, and proceeds as follows:

1. Represent the input sentence as a dependency tree.
2. Collect all nouns, verbs, and prepositional phrases that are attached directly to a wh-word or copula.
3. Associate each of these with a layout fragment: Ordinary nouns and verbs are mapped to a single `find` module. Proper nouns to a single `lookup` module. Prepositional phrases are mapped to a depth-2 fragment, with a `relate` module for the preposition above a `find` module for the enclosed head noun.
4. Form subsets of this set of layout fragments. For each subset, construct a layout candidate by joining all fragments with an `and` module, and inserting either a `measure` or `describe` module at the top (each subset thus results in two parse candidates.)

All layouts resulting from this process feature a relatively flat tree structure with at most one conjunction and one quantifier. This is a strong simplifying assumption, but appears sufficient to cover most of the examples that appear in both of our tasks. As our approach includes both categories, relations and simple quantification, the range of phenomena considered is generally broader than previous perceptually-grounded QA work [Krishnamurthy and Kollar, 2013, Matuszek et al., 2012].

Having generated a set of candidate parses, we need to score them. This is a ranking problem; as in the rest of our approach, we solve it using standard neural machinery. In particular, we produce an LSTM representation of the question, a feature-based representation of the query, and pass both representations through a multilayer perceptron (MLP). The query feature vector includes indicators on the number of modules of each type present, as well as their associated parameter arguments. While one can easily imagine a more sophisticated parse-scoring model, this simple approach works well for our tasks.

Formally, for a question x, let $h_q(x)$ be an LSTM encoding of the question (i.e. the last hidden layer of an LSTM applied word-by-word to the input question). Let $\{z_1, z_2, \ldots\}$ be

the proposed layouts for x, and let $f(z_i)$ be a feature vector representing the ith layout. Then the score $s(z_i|x)$ for the layout z_i is

$$s(z_i|x) = a^\top \sigma(Bh_q(x) + Cf(z_i) + d) \tag{2.8}$$

i.e. the output of an MLP with inputs $h_q(x)$ and $f(z_i)$, and parameters $\theta_\ell = \{a, B, C, d\}$. Finally, we normalize these scores to obtain a distribution:

$$p(z_i|x;\theta_\ell) = e^{s(z_i|x)} \Big/ \sum_{j=1}^{n} e^{s(z_j|x)} \tag{2.9}$$

Having defined a layout selection module $p(z|x;\theta_\ell)$ and a network execution model $p_z(y|w;\theta_e)$, we are ready to define a model for predicting answers given only (world, question) pairs. The key constraint is that we want to minimize evaluations of $p_z(y|w;\theta_e)$ (which involves expensive application of a deep network to a large input representation), but can tractably evaluate $p(z|x;\theta_\ell)$ for all z (which involves application of a shallow network to a relatively small set of candidates). This is the opposite of the situation usually encountered semantic parsing, where calls to the query execution model are fast but the set of candidate parses is too large to score exhaustively.

In fact, the problem more closely resembles the scenario faced by agents in the reinforcement learning setting (where it is cheap to score actions, but potentially expensive to execute them and obtain rewards). We adopt a common approach from that literature, and express our model as a stochastic policy. Under this policy, we first *sample* a layout z from a distribution $p(z|x;\theta_\ell)$, and then apply z to the knowledge source and obtain a distribution over answers $p(y|z,w;\theta_e)$.

After z is chosen, we can train the execution model directly by maximizing $\log p(y|z,w;\theta_e)$ with respect to θ_e as before (this is ordinary backpropagation). Because the hard selection of z is non-differentiable, we optimize $p(z|x;\theta_\ell)$ using a policy gradient method. The gradient of the reward surface J with respect to the parameters of the policy is

$$\nabla J(\theta_\ell) = \mathbb{E}[\nabla \log p(z|x;\theta_\ell) \cdot r] \tag{2.10}$$

(this is the REINFORCE rule [Williams, 1992]). Here the expectation is taken with respect to rollouts of the policy, and r is the reward. Because our goal is to select the network that makes the most accurate predictions, we take the reward to be identically the negative log-probability from the execution phase, i.e.

$$\mathbb{E}[(\nabla \log p(z|x;\theta_\ell)) \cdot \log p(y|z,w;\theta_e)] \tag{2.11}$$

Thus the update to the layout-scoring model at each timestep is simply the gradient of the log-probability of the chosen layout, scaled by the accuracy of that layout's predictions. At training time, we approximate the expectation with a single rollout, so at each step we update θ_ℓ in the direction $(\nabla \log p(z|x;\theta_\ell)) \cdot \log p(y|z,w;\theta_e)$ for a single $z \sim p(z|x;\theta_\ell)$. θ_e and θ_ℓ are optimized using ADADELTA [Zeiler, 2012] with $\rho = 0.95$, $\varepsilon = 1\text{e}{-6}$ and gradient clipping at a norm of 10.

A note on expressive power

While Section 2.3 and Section 2.2 motivated the use of structured, utterance-specific computational structures by analogy to formal semantics, the final predictors we can implement using the modules in Table 2.1 differ in a few significant ways from the kinds of formal representations employed by linguists. The type we have called Attention corresponds to a relaxed version of a model-theoretic space of *individuals*, and the type Labels to a more general notion of truth values (e.g. Montague [1973]); the modules we have defined cover most of the relevant direct transformations between them. But we have not defined any higher-order types (e.g. of the form ((Attention $\rightarrow$ Labels) $\rightarrow$ Attention); this in turn limits the set of phenomena we can actually model in our framework. With this restricted type system, and slight generalization of the `exists` module, it is possible to represent logical forms with at most one generalized quantifier.

While this is adequate for the datasets used for evaluation here, we know that nested quantification is necessary when representing language more generally. The current literature suggests two promising directions for getting around this limitation. The first is to construct NMNs that function as *energy-based models* [LeCun et al., 2006], in which prediction involves inference over network inputs with respect to a scalar score at the output. Such models have been used in the NMN framework for resolving referring expressions [Hu et al., 2017], but not more challenging quantified ones. An alternative possibility is to back away from the explicit analogy between module outputs and a model-theoretic type system, and instead allow these outputs to represent the abstract state of a more general unrolled inference process. This view suggests connections to *proof-theoretic*, rather than model-theoretic accounts of semantics. First steps in this direction are given by Rocktäschel and Riedel [2017], though again for a restricted class of computations.

2.5 Experiments

The framework described in this chapter is general, and we are interested in how well it performs on datasets of varying domain, size and linguistic complexity. To that end, we evaluate our model on tasks at opposite extremes of both these criteria: a large visual question answering dataset, and a small collection of more structured geography questions.

Questions About Images

Our first task is the Visual Question Answering challenge (VQA) [Antol et al., 2015]. The VQA dataset consists of more than 200,000 images paired with human-annotated questions and answers, as in Figure 2.4. We use the VQA 1.0 release, employing the development set for model selection and hyperparameter tuning, and reporting final results from the evaluation server on the test-standard set. For the experiments described in this section, the input feature representations w_i are computed by the the fifth convolutional layer of a 16-layer VGGNet after pooling [Simonyan and Zisserman, 2014]. Input images are scaled to 448×448

Figure 2.4: Sample outputs for the visual question answering task. The second row shows the final attention provided as input to the top-level `describe` module. For the first two examples, the model produces reasonable parses, attends to the correct region of the images (the ear and the woman's clothing), and generates the correct answer. In the third image, the verb is discarded and a wrong answer is produced.

	test-dev				test-std
	Yes/No	Number	Other	All	All
Zhou (2015)	76.6	35.0	42.6	55.7	55.9
Noh (2015)	80.7	37.2	41.7	57.2	57.4
Yang (2015)	79.3	36.6	46.1	58.7	58.9
NMN	81.2	38.0	44.0	58.6	58.7
D-NMN	81.1	38.6	45.5	59.4	**59.4**

Table 2.2: Results on the VQA test server. NMN is a baseline model that takes the largest structure generated by the process depicted in Figure 2.3. D-NMN ("dynamic" NMN) uses a learned structure selector as described above.

before computing their representations. We found that performance on this task was best if the candidate layouts were relatively simple: only `describe`, `and` and `find` modules are used, and layouts contain at most two conjuncts.

One weakness of this basic framework is a difficulty modeling prior knowledge about answers (of the form *most bears are brown*). This kinds of linguistic "prior" is essential for the VQA task, and easily incorporated. We simply introduce an extra hidden layer for recombining the final module network output with the input sentence representation $h_q(x)$ (see Equation 2.8), replacing Equation 2.1 with:

$$\log p_z(y|w, x) = (Ah_q(x) + B[\![z]\!]_w)_y \tag{2.12}$$

(Now modules with output type Labels should be understood as producing an answer embedding rather than a distribution over answers.) This allows the question to influence the answer directly.

Results are shown in Table 2.2. The use of dynamic networks provides a small gain, most noticeably on "other" questions. The proposed approach outperforms a highly effective visual bag-of-words model [Zhou et al., 2015], a model with dynamic network parameter prediction (but fixed network structure) [Noh et al., 2016], a more conventional attentional model [Yang et al., 2017], and an ablation with fixed layouts rather than learned structure prediction.

Some examples are shown in Figure 2.4. In general, the model learns to focus on the correct region of the image, and tends to consider a broad window around the region. This facilitates answering questions like *Where is the cat?*, which requires knowledge of the surroundings as well as the object in question.

Questions About Geography

The next set of experiments we consider focuses on GeoQA, a geography question answering task first introduced by Krishnamurthy and Kollar [2013]. This task was originally paired

	Accuracy	
Model	GeoQA	GeoQA+Q
LSP-F	48	–
LSP-W	51	–
NMN	51.7	35.7
D-NMN	**54.3**	**42.9**

Table 2.3: Results on the GeoQA dataset, and the GeoQA dataset with quantification. Our approach outperforms both a purely logical model (LSP-F) and a model with learned perceptual predicates (LSP-W) on the original dataset, and a fixed-structure NMN under both evaluation conditions.

with a visual question answering task much simpler than the one just discussed, and is appealing for a number of reasons. In contrast to the VQA dataset, GeoQA is quite small, containing only 263 examples. Two baselines are available: one using a classical semantic parser backed by a database, and another which induces logical predicates using linear classifiers over both spatial and distributional features. This allows us to evaluate the quality of our model relative to other perceptually grounded logical semantics, as well as strictly logical approaches.

The GeoQA domain consists of a set of entities (e.g. states, cities, parks) which participate in various relations (e.g. *north-of*, *capital-of*). Here we take the world representation to consist of two pieces: a set of category features (used by the `find` module) and a different set of relational features (used by the `relate` module). For our experiments, we use a subset of the features originally used by Krishnamurthy et al. The original dataset includes no quantifiers, and treats the questions *What cities are in Texas?* and *Are there any cities in Texas?* identically. Because we are interested in testing the parser's ability to predict a variety of different structures, we introduce a new version of the dataset, GeoQA+Q, which distinguishes these two cases, and expects a Boolean answer to questions of the second kind.

Results are shown in Table 2.3. As in the original work, we report the results of leave-one-environment-out cross-validation on the set of 10 environments. Our learned structure prediction model (D-NMN) outperforms both the logical (LSP-F) and perceptual models (LSP-W) described by Krishnamurthy and Kollar [2013], as well as a fixed-structure neural module net (NMN). This improvement is particularly notable on the dataset with quantifiers, where dynamic structure prediction produces a 20% relative improvement over the fixed baseline. A variety of predicted layouts are shown in Figure 2.5.

2.6 Discussion

This chapter has introduced a deep architecture, the *neural module network*, for answering queries about both structured and unstructured sources of information. Given only (question,

Is Key Largo an island? `(exists (and lookup[key-largo] find[island]))` **yes**: correct
What national parks are in Florida? `(and find[park] (relate[in] lookup[florida]))` **everglades**: correct

What are some beaches in Florida?
`(exists (and lookup[beach]`
`(relate[in] lookup[florida])))`
yes (daytona-beach): wrong parse

What beach city is there in Florida?
`(and lookup[beach] lookup[city]`
`(relate[in] lookup[florida]))`
[none] (daytona-beach): wrong module behavior

Figure 2.5: Example layouts and answers selected by the model on the GeoQA dataset. For incorrect predictions, the correct answer is shown in parentheses.

world, answer) triples as training data, the model learns to assemble neural networks on the fly from an inventory of neural models, and simultaneously learns weights for these modules so that they can be composed into novel structures. Our approach achieves strong results on two tasks.

The approach presented in this chapter still requires significant structure to be provided by the modeler, both in the form of an inventory of modules with an approriate space of intermediate representations, and in the form of network layouts (here derived from a parser but in principle supervisable directly instead). A fully general architecture for learning and prediction with modular networks would incorporate both representation design and structure selection into the learning process, without the need for external supervision. Following the publication of the work described here, several steps have been taken in this direction:

Learning intermediate representations The PG+EE model of Johnson et al. [2017b] is an NMN with non-attentional modules, with the intermediate module outputs instead predicted by a generic sequence of convolutions. This approach outperforms the attentional parameterization given in Table 2.1 on a challenging synthetic dataset [Johnson et al., 2017a] but has not been shown to scale to real-world image datasets.

Learning without structure supervision Attempts to learn a discrete layout construction policy with no supervision have met with limited success outside of toy domains [Hu et al., 2017]. However, moving from a reinforcement leanring setting to one with a soft structure selection policy using a differentiable stack [Grefenstette et al., 2015] gives improved performance in both naturalistic and highly compositional domains [Hu et al., 2018], but discovers network layouts that bear limited resemblance to linguistic representations of meaning. An alternative approach, with a similar high-level control scheme but without discrete reuse of a finite inventory of modules, is given by Hudson and Manning [2018].

To summarize: semantic annotations provide a useful framework from which to construct network layouts; effective but non-semantic layouts can be learned from scratch. Generic module parameterizations are effective in simple domains, but parameterizations guided by type systems from semantics seem to be more generally useful. Techniques for learning module nets without manual design of layouts or modules are beginning to be developed, but more work is needed to make them competitive with techniques relying on stronger supervision.

Chapter 3

Policy Sketches: Language and Behavior

In the next chapter, we turn from the problem of learning composable perceptual operators for reasoning to the problem of learning composable policies for interacting with the world.[1]

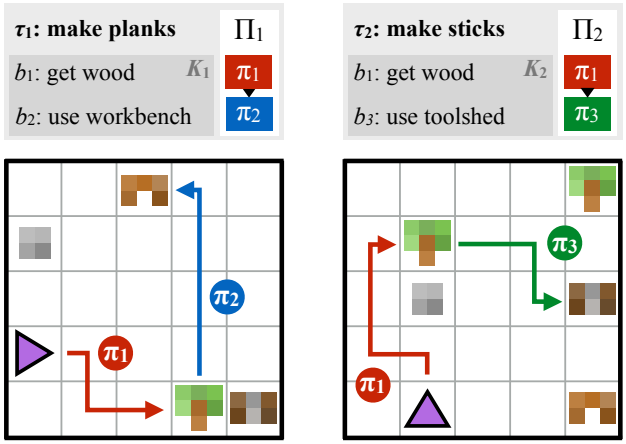


Figure 3.1: Learning from policy sketches. The figure shows simplified versions of two tasks (*make planks* and *make sticks*, each associated with its own policy (Π_1 and Π_2 respectively). These policies share an initial high-level action b_1: both require the agent to *get wood* before taking it to an appropriate crafting station. Even without prior information about how the associated behavior π_1 should be implemented, knowing that the agent should initially follow the same subpolicy in both tasks is enough to learn a reusable representation of their shared structure.

[1]Material in this chapter is adapted from:

- Jacob Andreas, Dan Klein and Sergey Levine. Modular multitask reinforcement with policy sketches. In *Proceedings of the International Conference on Machine Learning*, 2016.

Here we describe a framework for learning composable deep subpolicies in a multitask setting, guided only by abstract sketches of high-level behavior. General reinforcement learning algorithms allow agents to solve tasks in complex environments. But tasks featuring extremely delayed rewards or other long-term structure are often difficult to solve with flat, monolithic policies, and a long line of prior work has studied methods for learning hierarchical policy representations [Sutton et al., 1999, Dietterich, 2000, Konidaris and Barto, 2007, Hauser et al., 2008]. While unsupervised discovery of these hierarchies is possible [Daniel et al., 2012, Bacon and Precup, 2017], practical approaches often require detailed supervision in the form of explicitly specified high-level actions, subgoals, or behavioral primitives [Precup, 2000]. These depend on state representations simple or structured enough that suitable reward signals can be effectively engineered by hand.

But is such fine-grained supervision actually necessary to achieve the full benefits of hierarchy? Specifically, is it necessary to explicitly ground high-level actions into the representation of the environment? Or is it sufficient to simply inform the learner about the abstract *structure* of policies, without ever specifying how high-level behaviors should make use of primitive percepts or actions? To answer these questions, we explore a multitask reinforcement learning setting where the learner is presented with *policy sketches*. Policy sketches are short, ungrounded, symbolic representations of a task that describe its component parts, as illustrated in Figure 3.1. While symbols might be shared across tasks (*get wood* appears in sketches for both the *make planks* and *make sticks* tasks), the learner is told nothing about what these symbols *mean*, in terms of either observations or intermediate rewards.

We present an agent architecture that learns from policy sketches by associating each high-level action with a parameterization of a low-level subpolicy, and jointly optimizes over concatenated task-specific policies by tying parameters across shared subpolicies. We find that this architecture can use the high-level guidance provided by sketches, without any grounding or concrete definition, to dramatically accelerate learning of complex multistage behaviors. Our experiments indicate that many of the benefits to learning that come from highly detailed low-level supervision (e.g. from subgoal rewards) can also be obtained from fairly coarse high-level supervision (i.e. from policy sketches). Crucially, sketches are much easier to produce: they require no modifications to the environment dynamics or reward function, and can be easily provided by non-experts. This makes it possible to extend the benefits of hierarchical RL to challenging environments where it may not be possible to specify by hand the details of relevant subtasks. We show that our approach substantially outperforms purely unsupervised methods that do not provide the learner with any task-specific guidance about how hierarchies should be deployed, and further that the specific use of sketches to parameterize modular subpolicies makes better use of sketches than conditioning on them directly.

The present work may be viewed as an extension of recent approaches for learning compositional deep architectures from structured program descriptors, like those described in the

previous chapter or by Reed and de Freitas [2015]. Here we focus on learning in interactive environments. This extension presents a variety of technical challenges, requiring analogues of these methods that can be trained from sparse, non-differentiable reward signals without demonstrations of desired system behavior. Our contributions are: first, a general paradigm for multitask, hierarchical, deep reinforcement learning guided by abstract sketches of task-specific policies; and second, a concrete recipe for learning from these sketches, built on a general family of modular deep policy representations and a multitask actor–critic training objective.

The modular structure of our approach, which associates every high-level action symbol with a discrete subpolicy, naturally induces a library of interpretable policy fragments that are easily recombined. This makes it possible to evaluate our approach under a variety of different data conditions: (1) learning the full collection of tasks jointly via reinforcement, (2) in a zero-shot setting where a policy sketch is available for a held-out task, and (3) in a adaptation setting, where sketches are hidden and the agent must learn to adapt a pretrained policy to reuse high-level actions in a new task. In all cases, our approach substantially outperforms previous approaches based on explicit decomposition of the Q function along subtasks [Parr and Russell, 1998, Vogel and Jurafsky, 2010], unsupervised option discovery [Bacon and Precup, 2017], and several standard policy gradient baselines.

We consider three families of tasks: a 2-D Minecraft-inspired crafting game (Figure 3.3a), in which the agent must acquire particular resources by finding raw ingredients, combining them together in the proper order, and in some cases building intermediate tools that enable the agent to alter the environment itself; a 2-D maze navigation task that requires the agent to collect keys and open doors, and a 3-D locomotion task (Figure 3.3b) in which a quadrupedal robot must actuate its joints to traverse a narrow winding cliff. In all tasks, the agent receives a reward only after the final goal is accomplished. For the most challenging tasks, involving sequences of four or five high-level actions, a task-specific agent initially following a random policy essentially never discovers the reward signal, so these tasks cannot be solved without considering their hierarchical structure.

3.1 Related Work

The agent representation we describe in this chapter belongs to the broader family of hierarchical reinforcement learners. As detailed in Section 3.2, our approach may be viewed as an instantiation of the *options* framework first described by Sutton et al. [1999]. A large body of work describes techniques for learning options and related abstract actions, in both single- and multitask settings. Most techniques for learning options rely on intermediate supervisory signals, e.g. to encourage exploration [Kearns and Singh, 2002] or completion of pre-defined subtasks [Kulkarni et al., 2016]. An alternative family of approaches employs post-hoc analysis of demonstrations or pretrained policies to extract reusable sub-components [Stolle and Precup, 2002, Konidaris et al., 2011, Niekum et al., 2015]. Techniques for learning options with less guidance than the present work include those of Bacon and Precup [2017]

and Vezhnevets et al. [2016], and other general hierarchical policy learners include those of Daniel et al. [2012], Bakker and Schmidhuber [2004] and Menache et al. [2002]. We will see that the minimal supervision provided by policy sketches results in (sometimes dramatic) improvements over fully unsupervised approaches, while being substantially less onerous for humans to provide compared to the grounded supervision (such as explicit subgoals or feature abstraction hierarchies) used in previous work.

Once a collection of high-level actions exists, agents are faced with the problem of learning meta-level (typically semi-Markov) policies that invoke appropriate high-level actions in sequence [Precup, 2000]. The learning problem in this chapter is in some sense the direct dual to the problem of learning these meta-level policies: there, the agent begins with an inventory of complex primitives and must learn to model their behavior and select among them; here we begin knowing the names of appropriate high-level actions but nothing about how they are implemented, and must infer implementations (but not, initially, abstract plans) from context. Our model can be combined with these approaches to support a "mixed" supervision condition where sketches are available for some tasks but not others (Section 3.3).

Another closely related line of work is the Hierarchical Abstract Machines (HAM) framework introduced by Parr and Russell [1998]. Like our approach, HAMs begin with a representation of a high-level policy as an automaton (or a more general computer program; Andre and Russell [2001]; Marthi et al. [2004]) and use reinforcement learning to fill in low-level details. Because these approaches attempt to learn a single representation of the Q function for all subtasks and contexts, they require strong formal assumptions about the form of the reward function and state representation [Andre and Russell, 2002] that the present work avoids by decoupling the policy representation from the value function. They perform less effectively when applied to arbitrary state representations where these assumptions do not hold (Section 3.3). We are additionally unaware of past work showing that HAM automata can be automatically inferred for new tasks given a pre-trained model, while here we show that it is easy to solve the corresponding problem for sketch followers (Section 3.3).

Our approach is also inspired by a number of recent efforts toward compositional reasoning and interaction with structured deep models. Such models have been previously used for tasks involving question answering [Iyyer et al., 2014] and relational reasoning [Socher et al., 2012], and more recently for multi-task, multi-robot transfer problems [Devin et al., 2016]. In the present work—as in the approach described in the preceding chapter—task-specific training signals are propagated through a collection of composed discrete structures with tied weights. Here the composed structures specify time-varying policies rather than feedforward computations, and their parameters must be learned via interaction rather than direct supervision. Another closely related family of models includes neural programmers [Neelakantan et al., 2016] and programmer–interpreters [Reed and de Freitas, 2015], which generate discrete computational structures but require supervision in the form of output actions or full execution traces.

It is important to note that in this chapter, unlike the other chapters in this dissertation, our annotations are not written in "natural" language: policy sketches are built from a considerably simpler vocabulary and syntax. We view the problem of learning from policy

sketches as complementary to the instruction following problem studied in the natural language processing literature. Existing work on instruction following focuses on mapping from natural language strings to symbolic action sequences that are then executed by a hard-coded interpreter [Branavan et al., 2009, Chen and Mooney, 2011, Artzi and Zettlemoyer, 2013, Tellex et al., 2011a]. Here, by contrast, we focus on learning to execute complex actions given symbolic representations as a starting point. Instruction following models may be viewed as joint policies over instructions and environment observations (so their behavior is not defined in the absence of instructions), while the model described here naturally supports adaptation to tasks where no sketches are available. First steps towards combining the two lines of research—bootstrapping policy learning directly from natural language hints rather than the semi-structured sketches used here—are discussed in Chapter 4.

3.2 Learning Modular Policies from Sketches

We consider a multitask reinforcement learning problem arising from a family of infinite-horizon discounted Markov decision processes in a shared environment. This environment is specified by a tuple $(\mathcal{S}, \mathcal{A}, P, \gamma)$, with $\mathcal{S}$ a set of states, $\mathcal{A}$ a set of low-level actions, $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ a transition probability distribution, and γ a discount factor. Each task $\tau \in \mathcal{T}$ is then specified by a pair (R_τ, ρ_τ), with $R_\tau : \mathcal{S} \to \mathbb{R}$ a task-specific reward function and $\rho_\tau : \mathcal{S} \to \mathbb{R}$ an initial distribution over states. For a fixed sequence $\{(s_i, a_i)\}$ of states and actions obtained from a rollout of a given policy, we will denote the empirical return starting in state s_i as $q_i := \sum_{j=i+1}^{\infty} \gamma^{j-i-1} R(s_j)$. In addition to the components of a standard multitask RL problem, we assume that tasks are annotated with *sketches* K_τ, each consisting of a sequence $(b_{\tau 1}, b_{\tau 2}, \ldots)$ of high-level symbolic labels drawn from a fixed vocabulary $\mathcal{B}$.

Model

We exploit the structural information provided by sketches by constructing for each symbol b a corresponding *subpolicy* π_b. By sharing each subpolicy across all tasks annotated with the corresponding symbol, our approach naturally learns the shared abstraction for the corresponding subtask, without requiring any information about the grounding of that task to be explicitly specified by annotation.

At each timestep, a subpolicy may select either a low-level action $a \in \mathcal{A}$ or a special STOP action. We denote the augmented state space $\mathcal{A}^+ := \mathcal{A} \cup \{\text{STOP}\}$. At a high level, this framework is agnostic to the implementation of subpolicies: any function that takes a representation of the current state onto a distribution over $\mathcal{A}^+$ will do.[2] These subpolicies may be viewed as options of the kind described by Sutton et al. [1999], with the key distinction

[2]For ease of presentation, this section assumes that these subpolicy functions are independently parameterized. As described in Section 3.3, it is also possible to share parameters between subpolicies, and introduce discrete subtask structure by way of an *embedding* of each symbol b.

Algorithm 1 TRAIN-STEP($\mathbf{\Pi}$, curriculum)

$\mathcal{D} \leftarrow \emptyset$
while $|\mathcal{D}| < D$ **do**
  // sample task τ from curriculum (Section 3.2)
  $\tau \sim \text{curriculum}(\cdot)$
  // do rollout
  $d = \{(s_i, a_i, (b_i = K_{\tau,i}), q_i, \tau), \dots\} \sim \Pi_\tau$
  $\mathcal{D} \leftarrow \mathcal{D} \cup d$
// update parameters
for $b \in \mathcal{B}, \tau \in \mathcal{T}$ **do**
  $d = \{(s_i, a_i, b', q_i, \tau') \in \mathcal{D} : b' = b, \tau' = \tau\}$
  // update subpolicy
  $\theta_b \leftarrow \theta_b + \frac{\alpha}{D} \sum_d \big(\nabla \log \pi_b(a_i|s_i)\big)\big(q_i - c_\tau(s_i)\big)$
  // update critic
  $\eta_\tau \leftarrow \eta_\tau + \frac{\beta}{D} \sum_d \big(\nabla c_\tau(s_i)\big)\big(q_i - c_\tau(s_i)\big)$

that they have no initiation semantics, but are instead invocable everywhere, and have no explicit representation as a function from an initial state to a distribution over final states (instead implicitly using the STOP action to terminate).

Given a fixed sketch $(b_1, b_2, \dots)$, a task-specific policy Π_τ is formed by concatenating its associated subpolicies in sequence. In particular, the high-level policy maintains a subpolicy index i (initially 0), and executes actions from π_{b_i} until the STOP symbol is emitted, at which point control is passed to $\pi_{b_{i+1}}$. We may thus think of Π_τ as inducing a Markov chain over the state space $\mathcal{S} \times \mathcal{B}$, with transitions:

$$\begin{aligned} (s, b_i) &\to (s', b_i) && \text{with prob.} \quad \textstyle\sum_{a \in \mathcal{A}} \pi_{b_i}(a|s) \cdot P(s'|s, a) \\ &\to (s, b_{i+1}) && \text{with prob.} \quad \pi_{b_i}(\text{STOP}|s) \end{aligned}$$

Π_τ is semi-Markov with respect to projection of the augmented state space $\mathcal{S} \times \mathcal{B}$ onto the underlying state space $\mathcal{S}$. We denote the complete family of task-specific policies $\mathbf{\Pi} := \bigcup_\tau \{\Pi_\tau\}$, and let each π_b be an arbitrary function of the current environment state parameterized by some weight vector θ_b. The learning problem is to optimize over all θ_b to maximize expected discounted reward

$$J(\mathbf{\Pi}) := \sum_\tau J(\Pi_\tau) := \sum_\tau \mathbb{E}_{s_i \sim \Pi_\tau}\Big[\sum_i \gamma^i R_\tau(s_i)\Big]$$

across all tasks $\tau \in \mathcal{T}$.

Policy Optimization

Here that optimization is accomplished via a simple decoupled actor–critic method. In a standard policy gradient approach, with a single policy π with parameters θ, we compute

Algorithm 2 TRAIN-LOOP()

// *initialize subpolicies randomly*
$\mathbf{\Pi} = \text{INIT}()$
$\ell_{\text{max}} \leftarrow 1$
loop
  $r_{\text{min}} \leftarrow -\infty$
  // *initialize ℓ_{max}-step curriculum uniformly*
  $\mathcal{T}' = \{\tau \in \mathcal{T} : |K_\tau| \leq \ell_{\text{max}}\}$
  $\text{curriculum}(\cdot) = \text{Unif}(\mathcal{T}')$
  while $r_{\text{min}} < r_{\text{good}}$ **do**
    // *update parameters (Algorithm 1)*
    $\text{TRAIN-STEP}(\mathbf{\Pi}, \text{curriculum})$
    $\text{curriculum}(\tau) \propto \mathbb{1}[\tau \in \mathcal{T}'](1 - \hat{\mathbb{E}} r_\tau) \quad \forall \tau \in \mathcal{T}$
    $r_{\text{min}} \leftarrow \min_{\tau \in \mathcal{T}'} \hat{\mathbb{E}} r_\tau$
  $\ell_{\text{max}} \leftarrow \ell_{\text{max}} + 1$

gradient steps of the form [Williams, 1992]:

$$\nabla_\theta J(\pi) = \sum_i \big(\nabla_\theta \log \pi(a_i|s_i)\big)\big(q_i - c(s_i)\big), \tag{3.1}$$

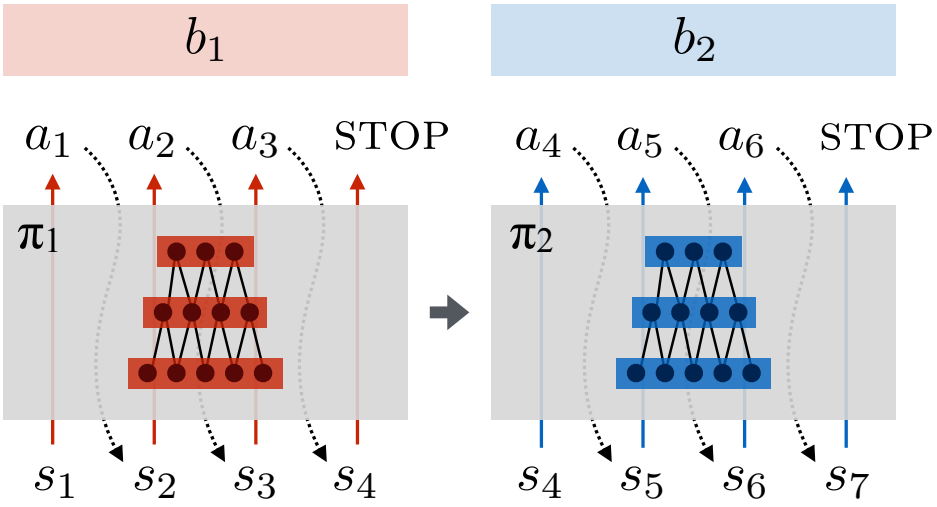


Figure 3.2: Model overview. Each subpolicy π is uniquely associated with a symbol b implemented as a neural network that maps from a state s_i to distributions over $\mathcal{A}^+$, and chooses an action a_i by sampling from this distribution. Whenever the STOP action is sampled, control advances to the next subpolicy in the sketch.

where the baseline or "critic" c can be chosen independently of the future without introducing bias into the gradient. Recalling our previous definition of q_i as the empirical return starting from s_i, this form of the gradient corresponds to a generalized advantage estimator [Schulman et al., 2016] with $\lambda = 1$. Here c achieves close to the optimal variance [Greensmith et al., 2004] when it is set exactly equal to the state-value function $V_\pi(s_i) = \mathbb{E}_\pi q_i$ for the target policy π starting in state s_i.

The situation becomes slightly more complicated when generalizing to modular policies built by sequencing subpolicies. In this case, we will have one subpolicy per symbol but one critic per *task*. This is because subpolicies π_b might participate in a number of composed policies Π_τ, each associated with its own reward function R_τ. Thus individual subpolicies are not uniquely identified with value functions, and the aforementioned subpolicy-specific state-value estimator is no longer well-defined. We extend the actor–critic method to incorporate the decoupling of policies from value functions by allowing the critic to vary per-sample (that is, per-task-and-timestep) depending on the reward function with which the sample is associated. Noting that

$$\nabla_{\theta_b} J(\mathbf{\Pi}) = \sum_{t:b\in K_\tau} \nabla_{\theta_b} J(\Pi_\tau),$$

i.e. the sum of gradients of expected rewards across all tasks in which π_b participates, we have:

$$\nabla_\theta J(\mathbf{\Pi}) = \sum_\tau \nabla_\theta J(\Pi_\tau) = \sum_\tau \sum_i \big(\nabla_{\theta_b} \log \pi_b(a_{\tau i}|s_{\tau i})\big)\big(q_i - c_\tau(s_{\tau i})\big), \tag{3.2}$$

where each state-action pair $(s_{\tau i}, a_{\tau i})$ was selected by the subpolicy π_b in the context of the task τ.

Now minimization of the gradient variance requires that each c_τ actually depend on the task identity. (This follows immediately by applying the corresponding argument in Greensmith et al. [2004] individually to each term in the sum over τ in Equation 3.2.) Because the value function is itself unknown, an approximation must be estimated from data. Here we allow these c_τ to be implemented with an arbitrary function approximator with parameters η_τ. This is trained to minimize a squared error criterion, with gradients given by

$$\nabla_{\eta_\tau}\left[-\frac{1}{2}\sum_i (q_i - c_\tau(s_i))^2\right] = \sum_i \big(\nabla_{\eta_\tau} c_\tau(s_i)\big)\big(q_i - c_\tau(s_i)\big). \tag{3.3}$$

Alternative forms of the advantage estimator (e.g. the TD residual $R_\tau(s_i)+\gamma V_\tau(s_{i+1})-V_\tau(s_i)$ or any other member of the generalized advantage estimator family) can be easily substituted by simply maintaining one such estimator per task. Experiments (Section 3.3) show that conditioning on both the state and the task identity results in noticeable performance improvements, suggesting that the variance reduction provided by this objective is important for efficient joint learning of modular policies.

The complete procedure for computing a *single* gradient step is given in Algorithm 1. (The outer training loop over these steps, which is driven by a curriculum learning procedure, is specified in Algorithm 2.) This is an on-policy algorithm. In each step, the agent samples tasks from a task distribution provided by a curriculum (described in the following subsection). The current family of policies $\mathbf{\Pi}$ is used to perform rollouts in each sampled task, accumulating the resulting tuples of (states, low-level actions, high-level symbols, rewards, and task identities) into a dataset $\mathcal{D}$. Once $\mathcal{D}$ reaches a maximum size D, it is used to compute gradients w.r.t. both policy and critic parameters, and the parameter vectors are updated accordingly. The step sizes α and β in Algorithm 1 can be chosen adaptively using any first-order method.

Curriculum Learning

For complex tasks, like the one depicted in Figure 3.3b, it is difficult for the agent to discover any states with positive reward until many subpolicy behaviors have already been learned. It is thus a better use of the learner's time to focus on "easy" tasks, where many rollouts will result in high reward from which appropriate subpolicy behavior can be inferred. But there is a fundamental tradeoff involved here: if the learner spends too much time on easy tasks before being made aware of the existence of harder ones, it may overfit and learn subpolicies that no longer generalize or exhibit the desired structural properties.

To avoid both of these problems, we use a curriculum learning scheme [Bengio et al., 2009, Kumar et al., 2010] that allows the model to smoothly scale up from easy tasks to more difficult ones while avoiding overfitting. Initially the model is presented with tasks associated with short sketches. Once average reward on all these tasks reaches a certain threshold, the length limit is incremented. We assume that rewards across tasks are normalized with maximum achievable reward $0 < q_i < 1$. Let $\hat{\mathbb{E}}r_\tau$ denote the empirical estimate of the expected reward for the current policy on task τ. Then at each timestep, tasks are sampled in proportion to $1 - \hat{\mathbb{E}}r_\tau$, which by assumption is positive.

Intuitively, the tasks that provide the strongest learning signal are those in which (1) the agent does not on average achieve reward close to the upper bound, but (2) many episodes result in high reward. The expected reward component of the curriculum addresses condition (1) by ensuring that time is not spent on nearly solved tasks, while the length bound component of the curriculum addresses condition (2) by ensuring that tasks are not attempted until high-reward episodes are likely to be encountered. Experiments show that both components of this curriculum learning scheme improve the rate at which the model converges to a good policy (Section 3.3).

The complete curriculum-based training procedure is specified in Algorithm 2. Initially, the maximum sketch length $\ell_{\max}$ is set to 1, and the curriculum initialized to sample length-1 tasks uniformly. (Neither of the environments we consider in this chapter feature any length-1 tasks; in this case, observe that Algorithm 2 will simply advance to length-2 tasks without any parameter updates.) For each setting of $\ell_{\max}$, the algorithm uses the current collection of task policies $\mathbf{\Pi}$ to compute and apply the gradient step described in Algorithm 1. The

rollouts obtained from the call to TRAIN-STEP can also be used to compute reward estimates $\hat{\mathbb{E}}r_\tau$; these estimates determine a new task distribution for the curriculum. The inner loop is repeated until the reward threshold r_{good} is exceeded, at which point ℓ_{max} is incremented and the process repeated over a (now-expanded) collection of tasks.

3.3 Experiments

We evaluate the performance of our approach in three environments: a crafting environment, a maze navigation environment, and a cliff traversal environment. These environments involve various kinds of challenging low-level control: agents must learn to avoid obstacles, interact with various kinds of objects, and relate fine-grained joint activation to high-level locomotion goals. They also feature hierarchical structure: most rewards are provided only after the agent has completed two to five high-level actions in the appropriate sequence, without any intermediate goals to indicate progress towards completion.

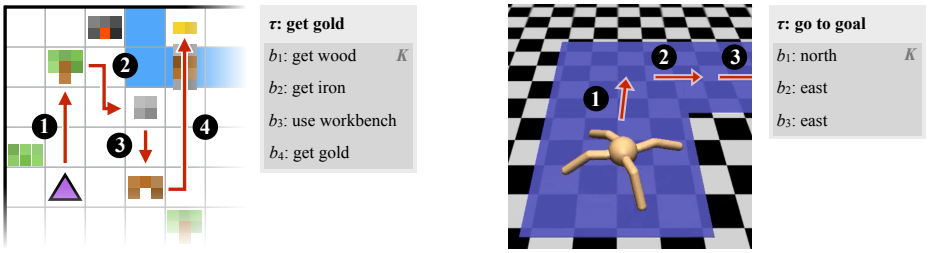
Implementation

In all experiments in this chapter, we implement each subpolicy as a feedforward neural network with ReLU nonlinearities and a hidden layer with 128 hidden units, and each critic as a linear function of the current state. Each subpolicy network receives as input a set of features describing the current state of the environment, and outputs a distribution over actions. The agent acts at every timestep by sampling from this distribution. The gradient steps given in lines 8 and 9 of Algorithm 1 are implemented using RMSPROP [Tieleman, 2012] with a step size of 0.001 and gradient clipping to a unit norm. We take the batch size D in Algorithm 1 to be 2000, and set $\gamma = 0.9$ in both environments. For curriculum learning, the improvement threshold r_{good} is 0.8.

Environments

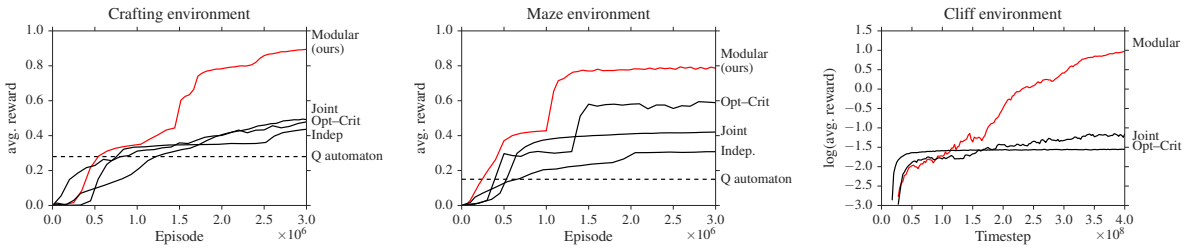
The crafting environment (Figure 3.3a) is inspired by the popular game Minecraft, but is implemented in a discrete 2-D world. The agent may interact with objects in the world by facing them and executing a special USE action. Interacting with raw materials initially scattered around the environment causes them to be added to an inventory. Interacting with different crafting stations causes objects in the agent's inventory to be combined or transformed. Each task in this game corresponds to some crafted object the agent must produce; the most complicated goals require the agent to also craft intermediate ingredients, and in some cases build tools (like a pickaxe and a bridge) to reach ingredients located in initially inaccessible regions of the environment.

The maze environment (not pictured) corresponds closely to the the "light world" described by Konidaris and Barto [2007]. The agent is placed in a discrete world consisting of a series of rooms, some of which are connected by doors. Some doors require that the



(a) (b)

Figure 3.3: Examples from the crafting and cliff environments. An additional maze environment is also investigated. (a) In the crafting environment, an agent seeking to pick up the gold nugget in the top corner must first collect wood (1) and iron (2), use a workbench to turn them into a bridge (3), and use the bridge to cross the water (4). (b) In the cliff environment, the agent must reach a goal position by traversing a winding sequence of tiles without falling off. Control takes place at the level of individual joint angles; high-level behaviors like "move north" must be learned.



(a) (b) (c)

Figure 3.4: Comparing modular learning from sketches with standard RL baselines. **Modular** is the approach described in this chapter, while **Independent** learns a separate policy for each task, **Joint** learns a shared policy that conditions on the task identity, **Q automaton** learns a single network to map from states and action symbols to Q values, and **Opt–Crit** is an unsupervised option learner. Performance for the best iteration of the (off-policy) Q automaton is plotted. Performance is shown in (a) the crafting environment, (b) the maze environment, and (c) the cliff environment. The modular approach is eventually able to achieve high reward on all tasks, while the baseline models perform considerably worse on average.

agent first pick up a key to open them. For our experiments, each task corresponds to a goal room (always at the same position relative to the agent's starting position) that the agent must reach by navigating through a sequence of intermediate rooms. The agent has one sensor on each side of its body, which reports the distance to keys, closed doors, and open doors in the corresponding direction. Sketches specify a particular sequence of directions for the agent to traverse between rooms to reach the goal. The sketch always corresponds to a viable traversal from the start to the goal position, but other (possibly shorter) traversals may also exist.

The cliff environment (Figure 3.3b) is intended to demonstrate the applicability of our approach to problems involving high-dimensional continuous control. In this environment, a quadrupedal robot [Schulman et al., 2015] is placed on a variable-length winding path, and must navigate to the end without falling off. This task is designed to provide a substantially more challenging RL problem, due to the fact that the walker must learn the low-level walking skill before it can make any progress, but has simpler hierarchical structure than the crafting environment. The agent receives a small reward for making progress toward the goal, and a large positive reward for reaching the goal square, with a negative reward for falling off the path.

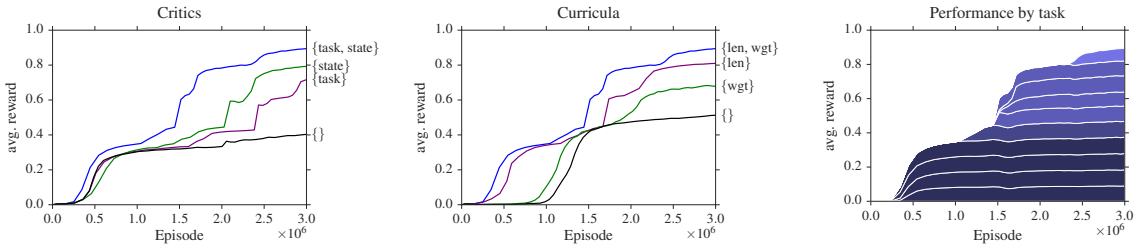
A listing of tasks and sketches is given in Appendix A.

Multitask Learning

In this chapter, our primary experimental question is whether the extra structure provided by policy sketches alone is enough to enable fast learning of coupled policies across tasks. We aim to explore the differences between the approach described in Section 3.2 and relevant prior work that performs either unsupervised or weakly supervised multitask learning of hierarchical policy structure. Specifically, we compare our **modular** to approach to:

1. Structured hierarchical reinforcement learners:
   (a) the fully unsupervised **option–critic** algorithm of Bacon and Precup [2017]
   (b) a **Q automaton** that attempts to explicitly represent the Q function for each task / subtask combination (essentially a HAM [Andre and Russell, 2002] with a deep state abstraction function)
2. Alternative ways of incorporating sketch data into standard policy gradient methods:
   (c) learning an **independent** policy for each task
   (d) learning a **joint** policy across all tasks, conditioning directly on both environment features and a representation of the complete sketch

The joint and independent models performed best when trained with the same curriculum described in Section 3.2, while the option–critic model performed best with a length–weighted curriculum that has access to all tasks from the beginning of training.



(a) (b) (c)

Figure 3.5: Training details in the crafting domain. (a) Critics: lines labeled "task" include a baseline that varies with task identity, while lines labeled "state" include a baseline that varies with state identity. Estimating a baseline that depends on both the representation of the current state and the identity of the current task is better than either alone or a constant baseline. (b) Curricula: lines labeled "len" use a curriculum with iteratively increasing sketch lengths, while lines labeled "wgt" sample tasks in inverse proportion to their current reward. Adjusting the sampling distribution based on both task length and performance return improves convergence. (c) Individual task performance. Colors correspond to task length. Sharp steps in the learning curve correspond to increases of $\ell_{\max}$ in the curriculum.

Learning curves for baselines and the modular model are shown in Figure 3.4. It can be seen that in all environments, our approach substantially outperforms the baselines: it induces policies with substantially higher average reward and converges more quickly than the policy gradient baselines. It can further be seen in Figure 3.4c that after policies have been learned on simple tasks, the model is able to rapidly adapt to more complex ones, even when the longer tasks involve high-level actions not required for any of the short tasks (Appendix A).

Having demonstrated the overall effectiveness of our approach, our remaining experiments explore (1) the importance of various components of the training procedure, and (2) the learned models' ability to generalize or adapt to held-out tasks. For compactness, we restrict our consideration on the crafting domain, which features a larger and more diverse range of tasks and high-level actions.

Ablations

In addition to the overall modular parameter-tying structure induced by our sketches, the key components of our training procedure are the decoupled critic and the curriculum. Our next experiments investigate the extent to which these are necessary for good performance.

To evaluate the the critic, we consider three ablations: (1) removing the dependence of the model on the environment state, in which case the baseline is a single scalar per task; (2) removing the dependence of the model on the task, in which case the baseline is

a conventional generalized advantage estimator; and (3) removing both, in which case the baseline is a single scalar, as in a vanilla policy gradient approach. Results are shown in Figure 3.5a. Introducing both state and task dependence into the baseline leads to faster convergence of the model: the approach with a constant baseline achieves less than half the overall performance of the full critic after 3 million episodes. Introducing task and state dependence independently improve this performance; combining them gives the best result.

We also investigate two aspects of our curriculum learning scheme: starting with short examples and moving to long ones, and sampling tasks in inverse proportion to their accumulated reward. Experiments are shown in Figure 3.5b. Both components help; prioritization by both length and weight gives the best results.

Zero-shot and Adaptation Learning

In our final experiments, we consider the model's ability to generalize beyond the standard training condition. We first consider two tests of generalization: a **zero-shot** setting, in which the model is provided a sketch for the new task and must immediately achieve good performance, and a **adaptation** setting, in which no sketch is provided and the model must learn the form of a suitable sketch via interaction in the new task.

We hold out two length-four tasks from the full inventory used in Section 3.3, and train on the remaining tasks. For zero-shot experiments, we simply form the concatenated policy described by the sketches of the held-out tasks, and repeatedly execute this policy (without learning) in order to obtain an estimate of its effectiveness. For adaptation experiments, we consider ordinary RL over high-level actions $\mathcal{B}$ rather than low-level actions $\mathcal{A}$, implementing the high-level learner with the same agent architecture as described in Section 3.2. Note that the Independent and Option–Critic models cannot be applied to the zero-shot evaluation, while the Joint model cannot be applied to the adaptation baseline (because it depends on pre-specified sketch features). Results are shown in Table 3.1. The held-out tasks are sufficiently challenging that the baselines are unable to obtain more than negligible reward: in particular, the joint model overfits to the training tasks and cannot generalize to new sketches, while the independent model cannot discover enough of a reward signal to learn in the adaptation setting. The modular model does comparatively well: individual subpolicies succeed in novel zero-shot configurations (suggesting that they have in fact discovered the behavior suggested by the semantics of the sketch) and provide a suitable basis for adaptive discovery of new high-level policies.

3.4 Discussion

We have described an approach for multitask learning of deep multitask policies guided by symbolic policy sketches. By associating each symbol appearing in a sketch with a modular neural subpolicy, we have shown that it is possible to build agents that share behavior across tasks in order to achieve success in tasks with sparse and delayed rewards.

Model	Multitask	0-shot	Adaptation
Joint	.49	.01	–
Independent	.44	–	.01
Option–Critic	.47	–	.42
Modular (ours)	**.89**	**.77**	**.76**

Table 3.1: Accuracy and generalization of learned models in the crafting domain. The table shows the task completion rate for each approach after convergence under various training conditions. **Multitask** is the multitask training condition described in Section 3.3, while **0-Shot** and **Adaptation** are the generalization experiments described in Section 3.3. Our modular approach consistently achieves the best performance.

This process induces an inventory of reusable and interpretable subpolicies which can be employed for zero-shot generalization when further sketches are available, and hierarchical reinforcement learning when they are not. Our work suggests that these sketches, which are easy to produce and require no grounding in the environment, provide an effective scaffold for learning hierarchical policies from minimal supervision.

Chapter 4

Latent Descriptions: Language and Learning

The final set of experiments in Chapter 3 showed that, for a particular class of structured policies, language-like annotations could be used in *pretraining* to support generalization to new reward functions even in the absence of further instructions. This chapter investigates a more general approach for using language to help learning even for tasks that do not directly involve language data.[1]

Here we specifically propose to use language as a latent *parameter space* for few-shot learning problems of all kinds, including classification, transduction and policy search. We aim to show that this linguistic parameterization produces models that are both more accurate and more interpretable than direct approaches to few-shot learning.

Like many recent frameworks for multitask- and meta-learning, our approach consists of three phases: a pretraining phase, a concept-learning phase, and an evaluation phase. Here, the product of pretraining is a language interpretation model that maps from descriptions to predictors (e.g. image classifiers or reinforcement learners). Our thesis is that language learning is a powerful, general-purpose kind of pretraining, even for tasks that do not directly involve language.

New concepts are learned by searching directly in the space of natural language strings to minimize the loss incurred by the language interpretation model (Figure 4.1). Especially on tasks that require the learner to model high-level compositional structure shared by training examples, natural language hypotheses serve a threefold purpose: they make it easier to discover these compositional concepts, harder to overfit to few examples, and easier for humans to understand inferred patterns.

Our approach can be implemented using a standard kit of model components, and is

[1]Material in this chapter is adapted from:

- Jacob Andreas and Dan Klein. Learning with latent language. In *Proceedings of the Annual Meeting of the North American Association for Computational Linguistics*, 2018.

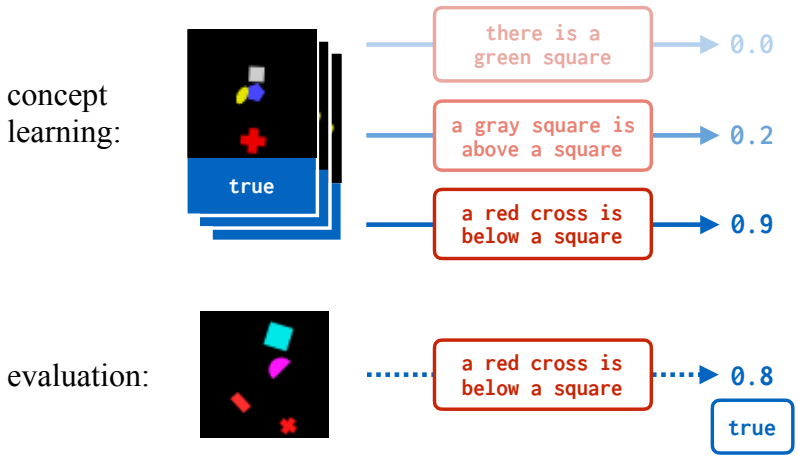


Figure 4.1: Example of our approach on a binary image classification task. We assume access to a pretrained language interpretation model that outputs the probability that an image matches a given description. To learn a new visual concept, we search in the space of natural language descriptions to maximize the interpretation model's score (top). The chosen description can be used with the interpretation model to classify new images (bottom).

simple and general. In a variety of settings, we find that the structure imposed by a natural-language parameterization is helpful for efficient learning and exploration. The approach outperforms both multitask- and meta-learning approaches that map directly from training examples to outputs by way of a real-valued parameterization, as well as approaches that make use of natural language annotations as an additional supervisory signal rather than an explicit latent parameter. The natural language concept descriptions inferred by our approach often agree with human annotations when they are correct, and provide an interpretable debugging signal when incorrect. In short, by equipping models with the ability to "think out loud" when learning, they become both more comprehensible and more accurate.

4.1 Background

Suppose we wish to solve an image classification problem like the one shown in Figure 4.2b–c, mapping from images x to binary labels y. One straightforward way to do this is to solve a learning problem of the following form:

$$\arg\min_{\eta \in \mathbf{H}} \sum_{(x,\, y)} L(f(x;\, \eta),\, y)\ , \tag{4.1}$$

where L is a loss function and f is a richly-parameterized class of models (e.g. convolutional networks) indexed by η (e.g. weight matrices) that map from images to labels. Given a new image x', $f(x';\, \eta)$ can be used to predict its label.

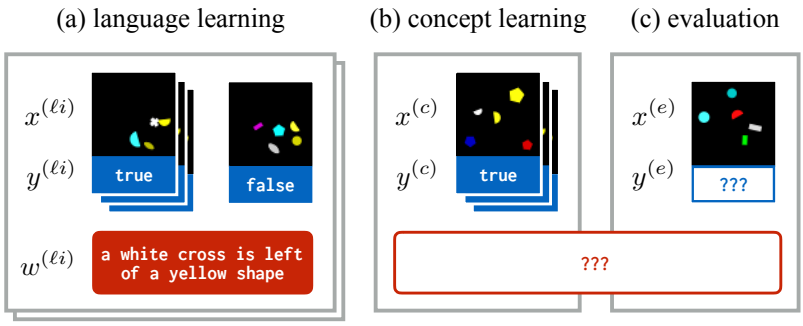


Figure 4.2: Formulation of the learning problem. Ultimately, we care about our model's ability to learn a concept from a small number of training examples (b) and successfully generalize it to held-out data (c). In this chapter, concept learning is supported by a language learning phase (a) that makes use of natural language annotations on other learning problems. These annotations are not provided for the real target task in (b–c).

In the present work, we are particularly interested in *few-shot* learning problems where the number of (x, y) pairs is small—on the order of five or ten examples. Under these conditions, directly solving Equation 4.1 is a risky proposition—any model class powerful enough to capture the true relation between inputs and outputs is also likely to overfit. For few-shot learning to be successful, extra structure must be supplied to the learner. Existing approaches obtain this structure by either carefully structuring the hypothesis space or providing the learner with alternative training data. The approach we present in this chapter combines elements of both, so we begin with a review of existing work.

(Inductive) *program synthesis* approaches (e.g. Gulwani, 2011) reduce the effective size of the hypothesis class $\mathbf{H}$ by moving the optimization problem out of the continuous space of weight vectors and into a discrete space of formal program descriptors (e.g. regular expressions or Prolog queries). Domain-specific structure like version space algebras [Lau et al., 2003] or type systems [Kitzelmann and Schmid, 2006] can be brought to bear on the search problem, and the bias inherent in the syntax of the formal language provides a strong prior. But while program synthesis techniques are powerful, they are also limited in their application: a human designer must hand-engineer the computational primitives necessary to compactly describe every learnable hypothesis. While reasonable for some applications (like string editing), this is challenging or impossible for others (like computer vision).

An alternative class of *multitask learning* approaches [Caruana, 1998] import the relevant structure from other learning problems rather than defining it manually (Figure 4.2a, top). Since we may not know *a priori* what set of learning problems we ultimately wish to evaluate on, it is useful to think of learning as taking places in three phases:

1. a *pretraining* (sometimes "meta-training") phase that makes use of various different datasets i with examples $\{(x_1^{(\ell i)}, y_1^{(\ell i)}), \ldots, (x_n^{(\ell i)}, y_n^{(\ell i)})\}$ (Figure 4.2a)

2. a *concept-learning* phase in which the pretrained model is adapted to fit data $\{(x_1^{(c)}, y_1^{(c)}), \ldots, (x_n^{(c)}, y_n^{(c)})\}$ for a specific new task (Figure 4.2b)

3. an *evaluation* phase in which the learned concept is applied to a new input $x^{(e)}$ to predict $y^{(e)}$ (Figure 4.2c)

In these approaches, learning operates over two collections of parameters: shared parameters η and task-specific parameters θ. In pretraining, multitask approaches find:

$$\underset{\eta \in \mathbb{R}^a,\ \theta^{(\ell i)} \in \mathbb{R}^b}{\arg\min} \quad \sum_{i,j} L\big(f(x_j^{(\ell i)};\, \eta,\, \theta^{(\ell i)}),\, y_j^{(\ell i)}\big)\ . \tag{4.2}$$

At concept learning time, they solve for:

$$\underset{\theta^{(c)} \in \mathbb{R}^b}{\arg\min} \quad \sum_{j} L\big(f(x_j^{(c)};\, \eta,\, \theta^{(c)}),\, y_j^{(c)}\big) \tag{4.3}$$

on the new dataset, then make predictions for new inputs using $f(x^{(e)};\, \eta,\, \theta^{(c)})$.

Closely related *meta-learning* approaches (e.g. Schmidhuber, 1987; Santoro et al., 2016; Vinyals et al., 2016) make use of the same data, but collapse the inner optimization over $\theta^{(c)}$ and prediction of $y^{(e)}$ into a single learned model.

4.2 Learning with Language

In this work, we are interested in developing a learning method that enjoys the benefits of both approaches. In particular, we seek an intermediate language of task representations that, like in program synthesis, is both expressive and compact, but like in multitask approaches is learnable directly from training data without domain engineering. We propose to use natural language as this intermediate representation. We call our approach *learning with latent language* (L^3).

Natural language shares many structural advantages with the formal languages used in synthesis approaches: it is discrete, has a rich set of compositional operators, and comes equipped with a natural description length prior. But it also has a considerably more flexible semantics. And crucially, plentiful annotated data exists for *learning* this semantics: we may not be able to hand-write a computer program to recognize a *small dog*, but we can learn how to do it from image captions. More basically, the set of primitive operators available in language provides a strong prior about the kinds of abstractions that are useful for natural learning problems.

Concretely, we replace the pretraining phase above with a *language-learning* phase. We assume that at language-learning time we have access to natural-language *descriptions* $w^{(\ell i)}$ (Figure 4.2a, bottom). We use these w as *parameters*, in place of the task-specific parameters θ—that is, we learn a language *interpretation* model $f(x;\, \eta,\, w)$ that uses shared parameters η to turn a description w into a function from inputs to outputs. For the example in Figure 4.2,

f might be an image rating model [Socher et al., 2014] that outputs a scalar judgment y of how well an image x matches a caption w.

Because these natural language parameters are observed at language-learning time, we need only learn the real-valued shared parameters η used for their interpretation (e.g. the weights of a neural network that implements the image rating model):

$$\arg\min_{\eta \in \mathbb{R}^a} \quad \sum_{i,\,j} L\big(f(x_j^{(\ell i)};\, \eta,\, w^{(\ell i)}),\, y_j^{(\ell i)}\big) \ . \tag{4.4}$$

At concept-learning time, conversely, we solve only the part of the optimization problem over natural language strings:

$$\arg\min_{w^{(c)} \in \Sigma^*} \quad \sum_{j} L\big(f(x_j^{(c)};\, \eta,\, w^{(c)}),\, y_j^{(c)}\big) \ . \tag{4.5}$$

This last step presents something of a challenge. When solving the corresponding optimization problem, synthesis techniques can exploit the algebraic structure of the formal language, while end-to-end learning approaches take advantage of differentiability. Here we can't do either—the language of strings is discrete, and any structure in the interpretation function is wrapped up inside the black box of f. Inspired by related techniques aimed at making synthesis more efficient [Devlin et al., 2017], we use learning to help us develop an effective optimization procedure for natural language parameters.

In particular, we simply use the language-learning datasets, consisting of pairs $(x_j^{(\ell i)}, y_j^{(\ell i)})$ and descriptions w_i, to fit a reverse *proposal* model, estimating:

$$\arg\max_\lambda \textstyle\sum_i \log q(w_i | x_1^{(\ell i)}, y_1^{(\ell i)}, \ldots, x_n^{(\ell i)}, y_n^{(\ell i)}; \lambda)$$

where q provides a (suitably normalized) approximation to the distribution of descriptions given task data. In the running example, this proposal distribution is essentially an image captioning model [Donahue et al., 2015]. By sampling from q, we expect to obtain candidate descriptions that are likely to obtain small loss. But our ultimate inference criterion is still the true model f: at evaluation time we perform the minimization in Equation 4.5 by drawing a fixed number of samples, selecting the hypothesis $w^{(c)}$ that obtains the lowest loss, and using $f(x^{(e)};\, \eta,\, w^{(c)})$ to make predictions.

What we have described so far is a generic procedure for equipping collections of related learning problems with a natural language hypothesis space. In Sections 4.4 and 4.5, we describe how this procedure can be turned into a concrete algorithm for supervised classification and sequence prediction. In Section 4.6, we describe how to extend these techniques to reinforcement learning.

4.3 Model and Training Details

In all models, RNN encoders and decoders use gated recurrent units [Cho et al., 2014].

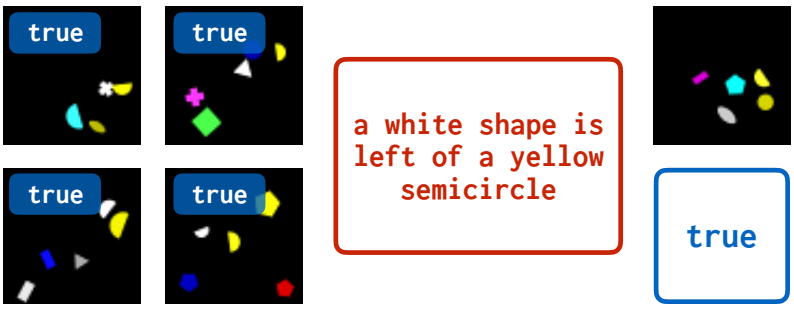


Figure 4.3: The few-shot image classification task. Learners are shown four positive examples of a visual concept (left) and must determine whether a fifth image matches the pattern (right). Natural language annotations are provided during language learning but must be inferred for concept learning.

Few-shot classification Models are trained with the ADAM optimizer [Kingma and Ba, 2014] with a step size of 0.0001 and batch size of 100. The number of pretraining iterations is tuned based on subsequent concept-learning performance on the development set. Neural network hidden states, task parameters, and word vectors are all of size 512. 10 hypotheses are sampled during for each evaluation task in the concept-learning phase.

Programming by demonstration Training as in the classification task, but with a step size of 0.001. Hidden states are of size 512, task parameters of size 128 and word vectors of size 32. 100 hypotheses are sampled for concept learning.

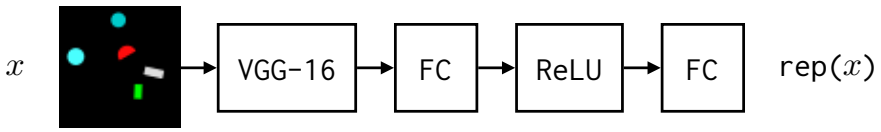
Policy search DAgger [Ross et al., 2011] is used for pre-training and vanilla policy gradient [Williams, 1992] for concept learning. Both learning algorithms use ADAM with a step size of 0.001 and a batch size of 5000 samples. For imitation learning, rollouts are obtained from the expert policy on a schedule with probability 0.95^t (for t the current epoch). For reinforcement learning, a discount of 0.9 is used. Because this dataset contains no development data, pretraining is run until performance on the pretraining tasks reaches a plateau. Hidden states and task embeddings are of size 64. 100 hypotheses are sampled for concept learning, and 1000 episodes (divided evenly among samples) are used to estimate hypothesis quality before fine-tuning.

4.4 Few-shot Classification

We begin by investigating whether natural language can be used to support high-dimensional few-shot classification. Our focus is on visual reasoning tasks like the one shown in Figure 4.3. In these problems, the learner is presented with four images, all positive examples of some

visual concept like *a blue shape near a yellow triangle*, and must decide whether a fifth, held-out image matches the same concept. These kinds of reasoning problems have been well-studied in visual question answering settings [Johnson et al., 2017a, Suhr et al., 2017]. Our version of the problem, where the input and output feature no text data, but an explanation must be inferred, is similar to the visual reasoning problems proposed by Raven [1936] and Bongard [1968].

To apply the recipe in Section 4.1, we need to specify an implementation of the interpretation model f and the proposal model q. We begin by computing representations of input images x. We start with a pre-trained 16-layer VGGNet [Simonyan and Zisserman, 2014]. Because spatial information is important for these tasks, we extract a feature representation from the final convolutional layer of the network. This initial featurization is passed through two fully-connected layers to form a final image representation, as follows:



We define interpretation and proposal models:[2]

$$f(x; w) = \sigma\big(\texttt{rnn-encode}(w)^\top \texttt{rep}(x)\big)$$
$$q(w \mid \{x_j\}) = \texttt{rnn-decode}\big(w \mid \tfrac{1}{n}\textstyle\sum_j \texttt{rep}(x_j)\big)$$

The interpretation model f outputs the probability that x is assigned a positive class label, and is trained to maximize log-likelihood. Because only positive examples are provided in each language learning set, the proposal model q can be defined in terms of inputs alone.

Our evaluation aims to answer two questions. First, does the addition of language to the learning process provide any benefit over ordinary multitask or meta-learning? Second, is it specifically better to use language as a hypothesis space for concept learning rather than just an additional signal for pretraining? We use several baselines to answer these questions:

1. *Multitask*: a multitask baseline in which the definition of f above is replaced by $\sigma(\theta_i^\top \texttt{rep}(x))$ for task-specific parameters θ_i that are optimized during both pretraining and concept-learning.

2. *Meta*: a meta-learning baseline in which f is defined by $\sigma([\frac{1}{n}\sum_j \texttt{rep}(x_j)]^\top \texttt{rep}(x))$.[3]

[2]Suppressing shared parameters η and λ for clarity.

[3]Many state-of-the-art approaches to meta-learning for classification (e.g. Snell et al., 2017) are not well-defined for possibly-overlapping evaluation classes with only positive examples provided. Here we have attempted to provide a robust implementation that is as close as possible to the other systems under evaluation.

3. *Meta+Joint*: as in *Meta*, but the pretraining objective includes an additional term for predicting q (discarded for concept learning).

We report results on a dataset derived from the ShapeWorld corpus of Kuhnle and Copestake [2017]. In this dataset the held-out image matches the target concept 50% of the time. In the validation and test folds, half of learning problems feature a concept that also appears in the language learning set (but with different exemplar images), while the other half feature both new images and a new concept. Images contain two or three distractor shapes unrelated to the objects that define the target concept. Captions in this dataset were generated from DMRS representations using an HPS grammar [Copestake et al., 2016]. (This is the only fully-synthetic dataset used in our experiments.) Each scene features 4 or 5 non-overlapping entities. Descriptions refer to spatial relationships between pairs of entities identified by shape, color, or both. There are 8 colors and 8 shapes. The total vocabulary size is only 30 words, but the dataset contains 2643 distinct captions. Descriptions are on average 12.0 words long. The dataset contains a total of 9000 pretraining tasks and 1000 of each validation and test tasks.

Results are shown in Table 4.1. It can be seen that L^3 provides consistent improvements over the baselines, and that these improvements are present both when identifying new instances of previously-learned concepts and when discovering new ones. Some example model predictions are shown in Figure 4.4. The model often succeeds in making correct predictions, even though its inferred descriptions rarely match the ground truth. Sometimes this is because of inherent ambiguity in the description language (Figure 4.4a), and sometimes because the model is able to rule out candidates on the basis of partial captions alone (Figure 4.4b, where it is sufficient to recognize that the target concept involves a *circle*). More examples are provided in Appendix B.

Model	Val (old)	Val (new)	Val	Test
Random	50	50	50	50
Multitask	64	49	57	59
Meta	63	62	62	64
Meta+Joint	63	69	66	64
L^3 (ours)	**70**	**72**	**71**	**70**
L^3 *(oracle)*	77	80	79	78

Table 4.1: Evaluation on image classification. *Val (old)* and *Val (new)* denote subsets of the validation set that contain respectively previously-used and novel visual concepts. L^3 consistently outperforms alternative learning methods based on multitask learning, meta-learning, and meta-learning jointly trained to predict descriptions (*Meta+Joint*). The last row shows results when the model is given a ground-truth concept description rather than having to infer it from examples.

4.5 Programming by Demonstration

Next we explore whether the same technique can be applied to tasks that involve more than binary similarity judgments. We focus on structured prediction: specifically a family of string processing tasks. In these tasks, the model is presented with examples of five strings transformed according to some rule; it must then apply an appropriate transformation to a sixth (Figure 4.5). Learning proceeds as in the previous section, with:

$$\begin{aligned} \texttt{rep}(x, y) &= \texttt{rnn-encode}([x, y]) \\ f(y \mid x; w) &= \texttt{rnn-decode}\big(y \mid [\texttt{rnn-encode}(x), \texttt{rnn-encode}(w)]\big) \\ q(w \mid \{(x_j, y_j)\}) &= \texttt{rnn-decode}\big(w \mid \tfrac{1}{n}\textstyle\sum_j \texttt{rep}(x_j, y_j)\big) \end{aligned}$$

Baselines are analogous to those for classification.

While string editing tasks of the kind shown in Figure 4.5 are popular in both the programming by demonstration literature [Singh and Gulwani, 2012] and the semantic parsing literature [Kushman and Barzilay, 2013], we are unaware of any datasets that support both learning paradigms at the same time. We have thus created a new dataset of string editing tasks by (1) sampling random regular transducers, (2) applying these transducers to collections of dictionary words, and (3) showing the collected examples to Mechanical Turk users

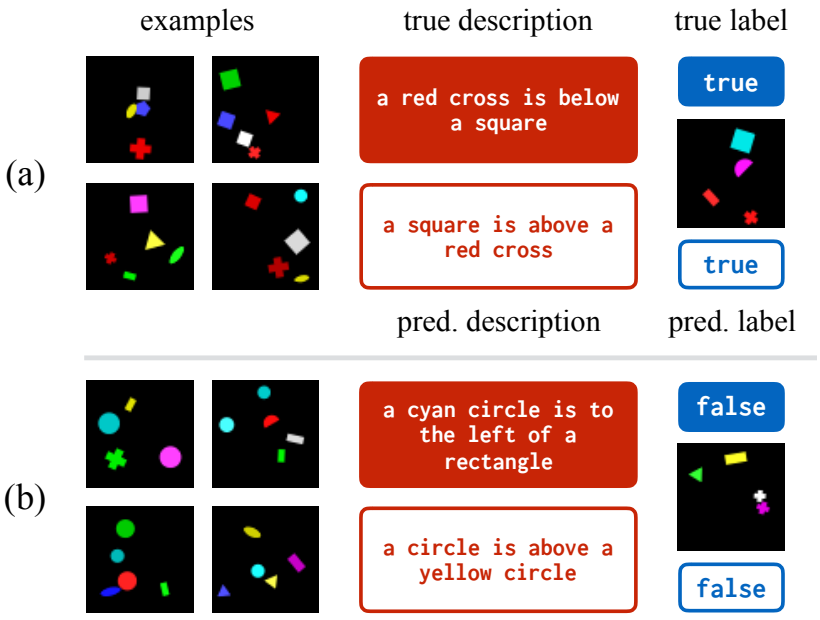


Figure 4.4: Example predictions for image classification. The model achieves high accuracy even though predicted descriptions rarely match the ground truth. High-level structure like the presence of certain shapes or spatial relations is consistently recovered.

and asking them to provide a natural language explanation with their best guess about the underlying rule. The dataset thus features both multi-example learning problems, as well as structured and unstructured annotations for each target concept.

Each user was presented with the same task as the learner in this paper: they observed five strings being transformed, and had to predict how to transform a sixth. Only after they correctly generated the held-out word were they asked for a description of the rule. Workers were additionally presented with hints like "look at the beginning of the word" or "look at the vowels". Descriptions are automatically preprocessed to strip punctuation and ensure that every character literal appears as a single token. The regular expression data has a vocabulary of 1015 rules and a total of 1986 distinct descriptions. Descriptions are on average 12.3 words in length but as long as 46 words in some cases. There are 3000 tasks for language learning and 500 tasks for each of validation and testing.

Due to its comparatively small size, a data augmentation scheme [Jia and Liang, 2016] is employed. In particular, wherever a description contains a recognizable entity name (i.e. a character literal), a description template is extracted. These templates are then randomly swapped in at training time on other examples with the same high-level semantics. For example, the description *replace first b with e* is abstracted to *replace first CHAR1 with CHAR2*, and can subsequently be specialized to, e.g., *replace first c with d*. This templating is easy to implement because we have access to ground-truth structured concept representations at training time. If these were not available it would be straightforward to employ an automatic template induction system [Kwiatkowski et al., 2011] instead.

Results are shown in Table 4.2. In these experiments, all models that use descriptions have been trained on the natural language supplied by human annotators. While we did find that the Meta+Joint model converges considerably faster than all the others, its final performance is somewhat lower than the baseline Meta model. As before, L^3 outperforms alternative approaches for learning directly from examples with or without descriptions.

Because all of the transduction rules in this dataset were generated from known formal descriptors, these tasks provide an opportunity to perform additional analysis comparing natural language to more structured forms of annotation (since we have access to ground-truth regular expressions) and more conventional synthesis-based methods (since we have

Figure 4.5: Example string editing task. Learners are presented with five examples of strings transformed according to some rule (left), and must apply an appropriate transformation to a sixth string (right). Language-learning annotations (center) may take the form of either natural language or regular expressions.

Model	Val	Test
Identity	18	18
Multitask	54	50
Meta	66	62
Meta+Joint	63	59
L^3	**80**	**76**

Table 4.2: Results for string editing. The reported number is the percentage of cases in which the predicted string exactly matches the reference. L^3 is the best performing model; using language data for joint training rather than as a hypothesis space provides little benefit.

access to a ground-truth regular expression execution engine). We additionally investigate the effect of the number of samples drawn from the proposal model. These results are shown in Table 4.3.

A few facts stand out. Under the ordinary evaluation condition (with no ground-truth annotations provided), language-learning with natural language data is actually better than language-learning with regular expressions. This might be because the extra diversity helps the model determine the relevant axes of variation and avoid overfitting to individual strings. Allowing the model to do its own inference is also better than providing ground-truth natural language descriptions, suggesting that it is actually better at generalizing from the relevant concepts than our human annotators (who occasionally write things like *I have no idea* for the inferred rule). Unsurprisingly, with ground truth REs (which unlike the human data are always correct) we can do better than any of the models that require inference. Coupling our inference procedure with an oracle RE evaluator, we essentially recover the synthesis-based approach of Devlin et al. [2017]. Our findings are consistent with theirs: when an exact execution engine is available, there is no reason not to use it. But we can get almost 90% of the way there with an execution model learned from scratch. Examples of model behavior

Annotations	Samples		Oracle	
	1	100	Ann.	Eval.
None (Meta)	*66*	–	–	–
Natural language	66	*80*	75	–
Regular expressions	60	76	88	90

Table 4.3: Inference and representation experiments for string editing. Italicized numbers correspond to entries in Table 4.2. Allowing the model to use multiple samples rather than the 1-best decoder output substantially improves performance. The full model does better with inferred natural language descriptions than either regular expressions or ground-truth natural language.

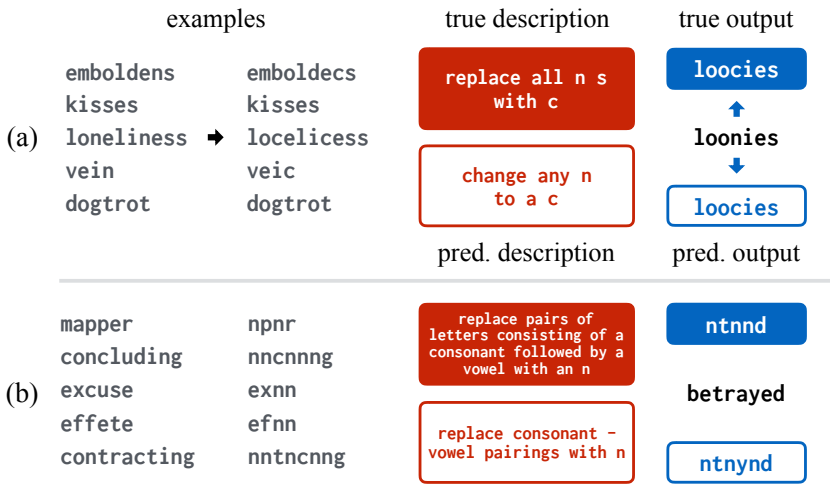


Figure 4.6: Example predictions for string editing.

are shown in Figure 4.6; more are in Figure 4.7.

4.6 Policy Search

The previous two sections examined supervised settings where the learning signal comes from few examples but is readily accessible. In this section, we move to a set of reinforcement learning problems, where the learning signal is instead sparse and time-consuming to obtain. We evaluate on a collection of 2-D treasure hunting tasks. These tasks require the agent to discover a rule that determines the location of buried treasure in a large collection of environments of the kind shown in Figure 4.8. To recover the treasure, the agent must navigate (while avoiding water) to its goal location, then perform a `DIG` action. At this point the episode ends; if the treasure is located in the agent's current position, it receives a reward, otherwise it does not. In every task, the treasure has consistently been buried at a fixed position relative to some landmark (in Figure 4.8 a heart). Both the offset and the identity of the target landmark are unknown to the agent, and the location of the landmark varies across maps. Indeed, there is nothing about the agent's observations or action space to suggest that landmarks and offsets are even the relevant axes of variation across tasks: only the language reveals this structure.

The data used was obtained from Janner et al. [2017]. We created our own variant of the dataset containing collections of related tasks. Beginning with the "local" tasks in the dataset, we generated alternative goal positions at fixed offsets from landmarks as described in the main section of this paper. Natural-language descriptions were selected for each task collection from the human annotations provided with the dataset. The vocabulary size is 74 and the number of distinct hints 446. The original action space for the environment is

Example in:	**Example out:**	**Human description:**		**True out:**
mediaeval	ilediaeval	leading consonant si replaced with i l		ilhaser
paneling	ilaneling		**Input:**	
wafer	ilafer		chaser	
conventions	ilonventions	**Inferred description:**		**Pred. out:**
handsprings	ilandsprings	first consonant of a word is replaced with i l		ilhaser
uptakes	uptakes	replace every o with u n		regulatiunn
pouching	punuching		regulation	
embroidery	embrunidery	change all o to u n		regulatinun
rebelliousness	rebelliunusness			
stoplight	stunplight			
fluffiest	fluffiest	the leter m is replaced by ee		chartering
kidnappers	kidnappers		chartering	
matting	eeatting	change every m to ee		chartering
griping	griping			
disagreements	disagreeeeents			
clandestine	clandqtine	e		gratuity
limning	limning		gratuity	
homes	homq	where e appears , replace it and the following letter with q		gratuity
lifeblood	lifqlood			
inflates	inflatq			
fruitlessly	fruitlessly	if the word ends with an s , replace the last two letters with m u		promptemu
sandier	sandier		prompters	
washers	washemu	change last to m u if consonant		promptemu
revelries	revelrimu			
dewlaps	dewlamu			
ladylike	ladylike	ending consonant is replaced with n l		initialnl
flintlocks	flintlocknl		initials	
student	studennl	drop last two and add n l		initialnl
surtaxes	surtaxenl			
bedecks	bedecknl			
porringer	porringeer	add e next to letter g		rage
puddling	puddlinge		rag	
synagog	synageoge	when a letter is preceded by a g , e is added after that letter		rage
curtseying	curtseyinge			
monsieur	monsieur			
trivializes	trivializes	replace the 1st 2 letters of the word with a g x if the word begins with a consonant then a vowel		gxndlords
tried	tried		landlords	
tearfully	gxarfully	if the second letter is a vowel , replace the first two letters with g x		gxndlords
hospitalize	gxspitalize			
patronizing	gxtronizing			
microseconds	microsecnyr	replace consonants with y r		exertion
antiviral	antiviral		exertion	
flintlock	flintloyr	the last two letters are replaced by y r		exertiyr
appreciable	appreciabyr			
stricter	stricter			

Figure 4.7: More example predictions for regular expressions.

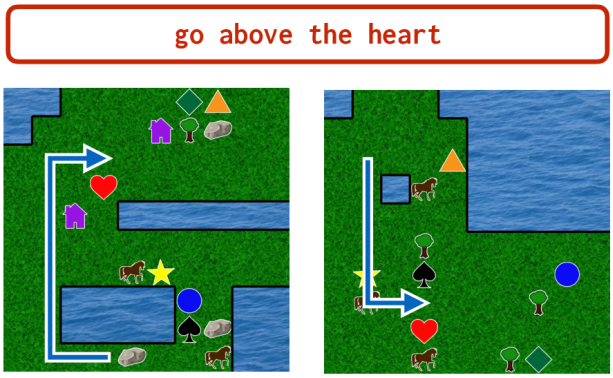


Figure 4.8: Example treasure hunting task: the agent is placed in a random environment and must collect a reward that has been hidden at a consistent offset with respect to some landmark. At language-learning time only, natural language instructions and expert policies are provided. The agent must both learn primitive navigation skills, like avoiding water, as well as the high-level structure of the reward functions for this domain.

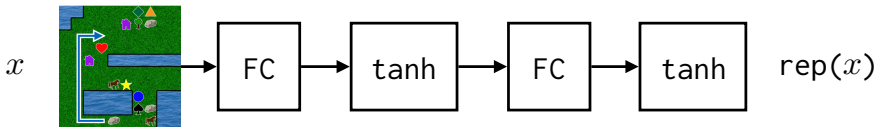
also modified slightly: rather than simply reaching the goal cell (achieved with reasonably high frequency by a policy that takes random moves), we require the agent to commit to an individual goal cell and end the episode with the `DIG` action. A similar data augmentation scheme to the regular expression data was employed, with templates constructed by abstracting over landmark names.

The interaction between language and learning in these tasks is rather different from the supervised settings. In the supervised case, language serves mostly as a guard against overfitting, and can be generated conditioned on a set of pre-provided concept-learning observations. Here, agents are free to interact with the environment as much as they need, but receive observations only during interaction. Thus our goal here will be to build agents that can *adapt quickly* to new environments, rather than requiring them to immediately perform well on held-out data.

Why should we expect L^3 to help in this setting? In reinforcement learning, we typically encourage our models to explore by injecting randomness into either the agent's action space or its underlying parameterization. But most random policies exhibit nonsensical behaviors; as a result, it is inefficient both to sample in the space of network weights and to perform policy optimization from a random starting point. Our hope is that when parameters are chosen from within a structured family, a stochastic search in this structured space will only ever consider behaviors corresponding to a reasonable final policy, and in this way discover good behavior faster than ordinary RL.

Here the interpretation model f describes a policy that chooses actions conditioned on the current environment state and a linguistic parameterization. As the agent initially has no observations at all, we simply design the proposal model to generate unconditional

samples from a prior over descriptions. Taking x to be an agent's current observation of the environment state, we define a state representation network and models:



$$f(a \mid x; w) \propto \texttt{rnn-encode}(w)^\top \, W_a \, \texttt{rep}(x)$$
$$q(w) = \texttt{rnn-decode}(w)$$

This parameterization assumes a discrete action space, and assigns to each action a probability proportional to a bilinear function of the encoded description and world state. f is an instruction following model of a kind well-studied in natural language processing [Branavan et al., 2009]; the proposal model allows it to generate its own instructions without external direction. To learn, we sample a fixed number of descriptions w from q. For each description, we sample multiple rollouts of the policy it induces to obtain an estimate of its average reward. Finally, we take the highest-scoring description and fine-tune its induced policy.

At language-learning time, we assume access to both natural language descriptions of these target locations provided by human annotators, as well as expert policies for navigating to the location of the treasure. The multitask model we compare to replaces these descriptions with trainable task embeddings.[4] The learner is trained from task-specific expert policies using DAgger [Ross et al., 2011] during the language-learning phase, and adapts to individual environments using "vanilla" policy gradient [Williams, 1992] during the concept learning phase.

The environment implementation and linguistic annotations are in this case adapted from a natural language navigation dataset originally introduced by Janner et al. [2017]. In our version of the problem (Figure 4.8), the agent begins each episode in a random position on a randomly-chosen map and must attempt to obtain the treasure. Relational concepts describing target locations are reused between language learning and concept-learning phases, but the environments themselves are distinct. For language learning the agent has access to 250 tasks, and is evaluated on an additional 50.

Averaged learning curves for held-out tasks are shown in Figure 4.9. As expected, reward for the L^3 model remains low during the initial exploration period, but once a description is chosen the score improves rapidly. Immediately L^3 achieves better reward than the multitask baseline, though it is not perfect; this suggests that the interpretation model is somewhat overfit to the pretraining environments. After fine-tuning even better results are rapidly

[4] In RL, the contribution of L^3 is orthogonal to that of meta-learning—one could use a technique like RL^2 [Duan et al., 2016] to generate candidate descriptions more efficiently, or MAML [Finn et al., 2017] rather than zero-shot reward as the training criterion for the interpretation model.

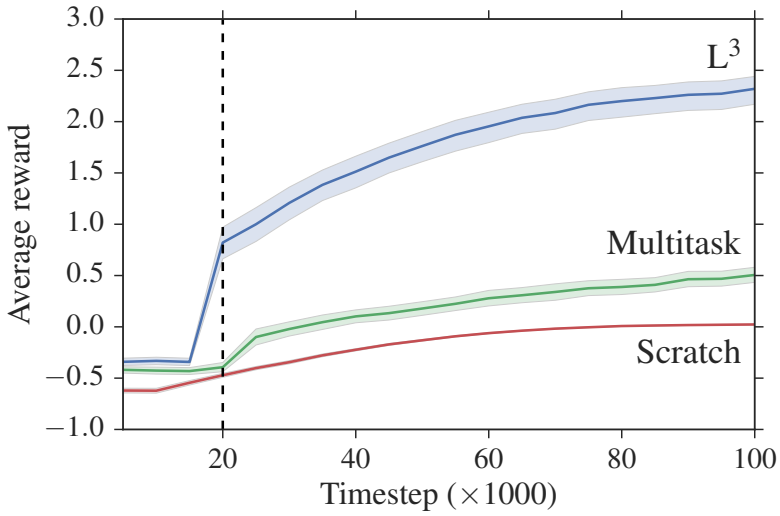


Figure 4.9: Treasure hunting reward obtained by each learning algorithm across multiple evaluation environments, after language learning has already taken place (bands show 95% confidence intervals for mean performance). *Multitask* learns an embedding for each task, while *Scratch* trains on every task individually. L^3 rapidly discovers high-scoring policies in most environments. Dashed line indicates the end of the concept-learning phase; subsequent performance comes from fine-tuning. The max reward for this task is 3.

obtained. Example rollouts are visualized in Appendix B. These results show that the model has used the structure provided by language to *learn* a better representation space for policies—one that facilitates sampling from a distribution over interesting and meaningful behaviors.

4.7 Other Related Work

This is the first approach we are aware of to frame a general learning problem as optimization over a space of natural language strings. However, many closely related ideas have been explored in the literature. String-valued latent variables are widely used in language processing tasks ranging from morphological analysis [Dreyer and Eisner, 2009] to sentence compression [Miao and Blunsom, 2016]. Natural language annotations have been used in conjunction with training examples to guide the discovery of logical descriptions of concepts [Ling et al., 2017, Srivastava et al., 2017], and used as an auxiliary loss for training [Frome et al., 2013], analogously to the Meta+Joint baseline. Structured language-like annotations have been used to improve learning of generalizable structured policies as in the previous chapter and Oh et al. [2017], Denil et al. [2017]. Finally, natural language instructions available at *concept-learning* time (rather than language-learning time) have been used to provide

side information to reinforcement learners about high-level strategy [Branavan et al., 2011], environment dynamics [Narasimhan et al., 2017] and exploration [Harrison et al., 2017].

4.8 Discussion

We have presented an approach for learning in a space parameterized by natural language. Using simple models for representation and search in this space, we demonstrated that our approach outperforms standard baselines on classification, structured prediction and reinforcement learning tasks.

Chapter 5

Translating Neuralese: Language and Belief

In the final chapter of this thesis, we turn from the problem of using linguistic structure to build more accurate models toward the problem of using language to interpret existing models.[1] We will focus specifically on the problem of interpreting *communication* in learned multiagent policies.

Several recent papers have described approaches for learning *deep communicating policies* (DCPs): decentralized representations of behavior that enable multiple agents to communicate via a differentiable channel that can be formulated as a recurrent neural network. DCPs have been shown to solve a variety of coordination problems, including reference games [Lazaridou et al., 2016], logic puzzles [Foerster et al., 2016], and simple control [Sukhbaatar et al., 2016]. Appealingly, the agents' communication protocol can be learned via direct backpropagation through the communication channel, avoiding many of the challenging inference problems associated with learning in classical decentralized decision processes [Roth et al., 2005].

But analysis of the strategies induced by DCPs has remained a challenge. As an example, Figure 5.1 depicts a driving game in which two cars, which are unable to see each other, must both cross an intersection without colliding. In order to ensure success, it is clear that the cars must communicate with each other. But a number of successful communication strategies are possible—for example, they might report their exact (x, y) coordinates at every timestep, or they might simply announce whenever they are entering and leaving the intersection. If these messages were communicated in natural language, it would be straightforward to determine which strategy was being employed. However, DCP agents instead communicate with an automatically induced protocol of unstructured, real-valued recurrent state vectors—

[1] Material in this chapter is adapted from:

- Jacob Andreas, Anca Dragan and Dan Klein. Translating neuralese. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2017.

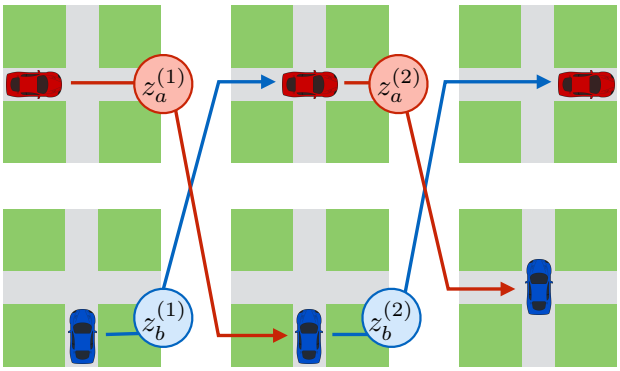


Figure 5.1: Example interaction between a pair of agents in a deep communicating policy. Both cars are attempting to cross the intersection, but cannot see each other. By exchanging message vectors $z^{(t)}$, the agents are able to coordinate and avoid a collision. This chapter presents an approach for *understanding* the contents of these message vectors by translating them into natural language.

an artificial language we might call "neuralese," which superficially bears little resemblance to natural language, and thus frustrates attempts at direct interpretation.

We propose to understand neuralese messages by *translating* them. In this work, we present a simple technique for inducing a dictionary that maps between neuralese message vectors and short natural language strings, given only examples of DCP agents interacting with other agents, and humans interacting with other humans. Natural language already provides a rich set of tools for describing beliefs, observations, and plans—our thesis is that these tools provide a useful complement to the visualization and ablation techniques used in previous work on understanding complex models [Strobelt et al., 2016, Ribeiro et al., 2016].

While structurally quite similar to the task of machine translation between pairs of human languages, interpretation of neuralese poses a number of novel challenges. First, there is no natural source of parallel data: there are no bilingual "speakers" of both neuralese and natural language. Second, there may not be a direct correspondence between the strategy employed by humans and DCP agents: even if it were constrained to communicate using natural language, an automated agent might choose to produce a different message from humans in a given state. We tackle both of these challenges by appealing to the grounding of messages in gameplay. Our approach is based on one of the core insights in natural language semantics: messages (whether in neuralese or natural language) have similar meanings *when they induce similar beliefs about the state of the world.*

Based on this intuition, we introduce a translation criterion that matches neuralese messages with natural language strings by minimizing statistical distance in a common representation space of distributions over speaker states. We explore several related questions:

- What makes a good translation, and under what conditions is translation possible at all? (Section 5.3)

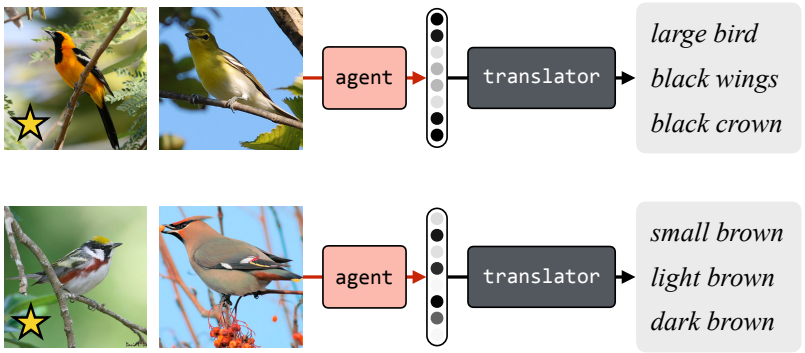


Figure 5.2: Overview of our approach—best-scoring translations generated for a reference game involving images of birds. The speaking agent's goal is to send a message that uniquely identifies the bird on the left. From these translations it can be seen that the learned model appears to discriminate based on coarse attributes like size and color.

- How can we build a model to translate between neuralese and natural language? (Section 5.4)
- What kinds of theoretical guarantees can we provide about the behavior of agents communicating via this translation model? (Section 5.6)

Our translation model and analysis are general, and in fact apply equally to human–computer and human–human translation problems grounded in gameplay. In this chapter, we focus our experiments specifically on the problem of interpreting communication in deep policies, and apply our approach to the driving game in Figure 5.1 and two reference games of the kind shown in Figure 5.2. We find that this approach outperforms a more conventional machine translation criterion both when attempting to interoperate with neuralese speakers and when predicting their state.

5.1 Related Work

A variety of approaches for learning deep policies with communication were proposed essentially simultaneously in the past year. We have broadly labeled these as "deep communicating policies"; concrete examples include Lazaridou et al. [2016], Foerster et al. [2016], and Sukhbaatar et al. [2016]. The policy representation we employ in this chapter is similar to the latter two of these, although the general framework is agnostic to low-level modeling details and could be straightforwardly applied to other architectures. Analysis of communication strategies in all these papers has been largely ad-hoc, obtained by clustering states from which similar messages are emitted and attempting to manually assign semantics to

these clusters. The present work aims at developing tools for performing this analysis automatically.

Most closely related to our approach is that of Lazaridou et al. [2017], who also develop a model for assigning natural language interpretations to learned messages; however, this approach relies on supervised cluster labels and is targeted specifically towards referring expression games. Here we attempt to develop an approach that can handle more general multiagent interactions.

The literature on learning decentralized multi-agent policies in general is considerably larger [Bernstein et al., 2002, Dibangoye et al., 2016]. This includes work focused on communication in multiagent settings [Roth et al., 2005] and even communication using natural language messages [Vogel et al., 2013b]. All of these approaches employ structured communication schemes with manually engineered messaging protocols; these are, in some sense, automatically interpretable, but at the cost of introducing considerable complexity into both training and inference.

Our evaluation investigates communication strategies that arise in a number of different games, including reference games and an extended-horizon driving game. Communication strategies for reference games were previously explored by Vogel et al. [2013a], Andreas and Klein [2016] and Kazemzadeh et al. [2014], and reference games specifically featuring end-to-end communication protocols by Yu et al. [2017]. On the control side, a long line of work considers nonverbal communication strategies in multiagent policies [Dragan and Srinivasa, 2013].

Another group of related approaches focuses on the development of more general machinery for interpreting deep models in which messages have no explicit semantics. This includes both visualization techniques [Zeiler and Fergus, 2014, Strobelt et al., 2016], and approaches focused on generating explanations in the form of natural language [Hendricks et al., 2016, Vedantam et al., 2017].

5.2 Problem Formulation

Games Consider a cooperative game with two players a and b of the form given in Figure 5.3. At every step t of this game, player a makes an observation $x_a^{(t)}$ and receives a message $z_b^{(t-1)}$ from b. It then takes an action $u_a^{(t)}$ and sends a message $z_a^{(t)}$ to b. (The process is symmetric for b.) The distributions $p(u_a|x_a, z_b)$ and $p(z_a|x_a)$ together define a policy π which we assume is shared by both players, i.e. $p(u_a|x_a, z_b) = p(u_b|x_b, z_a)$ and $p(z_a|x_a) = p(z_b|x_b)$. As in a standard Markov decision process, the actions $(u_a^{(t)}, u_b^{(t)})$ alter the world state, generating new observations for both players and a reward shared by both.

The distributions $p(z|x)$ and $p(u|x, z)$ may also be viewed as defining a *language*: they specify how a speaker will generate messages based on world states, and how a listener will respond to these messages. Our goal in is to learn to translate between pairs of languages generated by different policies. Specifically, we assume that we have access to two policies for the same game: a "robot policy" π_r and a "human policy" π_h. We would like to use

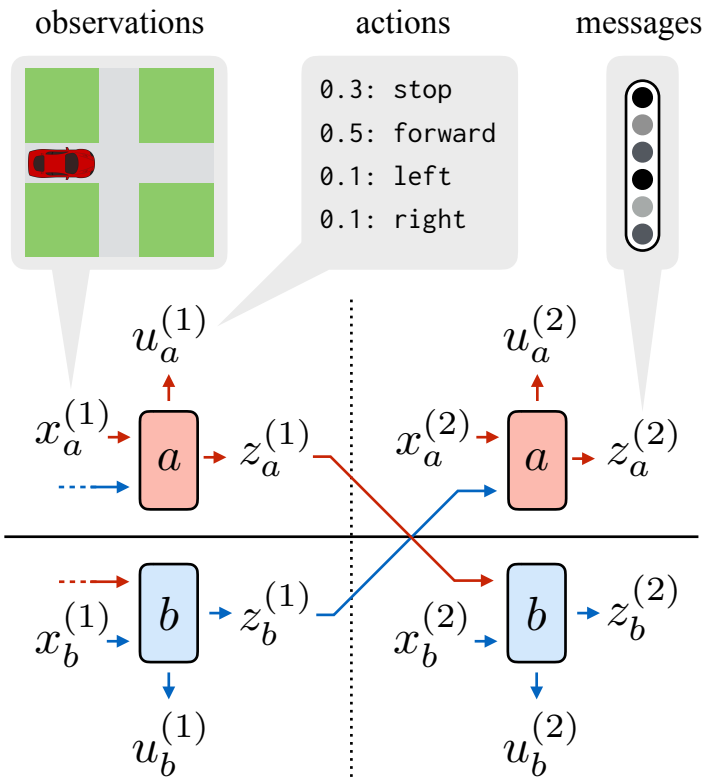


Figure 5.3: Schematic representation of communication games. At every timestep t, players a and b make an observation $x^{(t)}$ and receive a message $z^{(t-1)}$, then produce an action $u^{(t)}$ and a new message $z^{(t)}$.

the representation of π_h, the behavior of which is transparent to human users, in order to *understand* the behavior of π_r (which is in general an uninterpretable learned model); we will do this by inducing bilingual dictionaries that map message vectors z_r of π_r to natural language strings z_h of π_h and vice-versa.

Learned agents $\boldsymbol{\pi_r}$ Our goal is to present tools for interpretation of learned messages that are agnostic to the details of the underlying algorithm for acquiring them. We use a generic DCP model as a basis for the techniques developed in this chapter. Here each agent policy is represented as a deep recurrent Q network [Hausknecht and Stone, 2015]. This network is built from communicating cells of the kind depicted in Figure 5.4. At every timestep, this agent receives three pieces of information: an observation of the current state of the world, the agent's memory vector from the previous timestep, and a message from the other player. It then produces three outputs: a predicted Q value for every possible action, a new memory vector for the next timestep, and a message to send to the other agent.

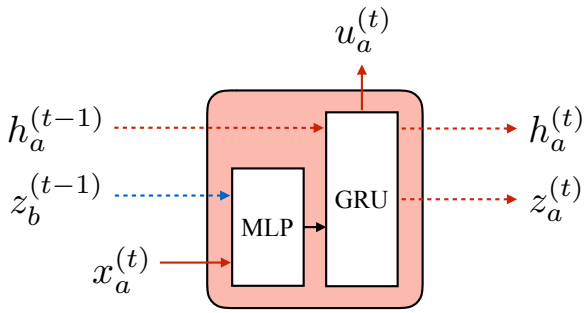


Figure 5.4: Cell implementing a single step of agent communication (compare with Sukhbaatar et al. [2016] and Foerster et al. [2016]). *MLP* denotes a multilayer perceptron; *GRU* denotes a gated recurrent unit [Cho et al., 2014]. Dashed lines represent recurrent connections.

Sukhbaatar et al. [2016] observe that models of this form may be viewed as specifying a single RNN in which weight matrices have a particular block structure. Such models may thus be trained using the standard recurrent Q-learning objective, with communication protocol learned end-to-end.

Human agents $\boldsymbol{\pi_h}$ The translation model we develop requires a representation of the distribution over messages $p(z_a|x_a)$ employed by human speakers (without assuming that humans and agents produce equivalent messages in equivalent contexts). We model the human message generation process as categorical, and fit a simple multilayer perceptron model to map from observations to words and phrases used during human gameplay.

5.3 What's in a Translation?

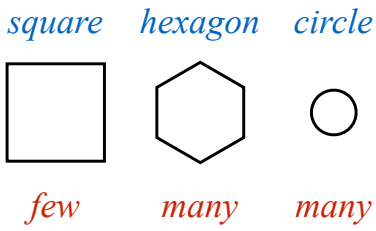
What does it mean for a message z_h to be a "translation" of a message z_r? In standard machine translation problems, the answer is that z_h is likely to co-occur in parallel data with z_r; that is, $p(z_h|z_r)$ is large. Here we have no parallel data: even if we could observe natural language and neuralese messages produced by agents in the same state, we would have no guarantee that these messages actually served the same function. Our answer must instead appeal to the fact that both natural language and neuralese messages are grounded in a common environment. For a given neuralese message z_r, we will first compute a grounded representation of that message's meaning; to translate, we find a natural-language message whose meaning is most similar. The key question is then what form this grounded meaning representation should take. The existing literature suggests two broad approaches:

Semantic representation The meaning of a message z_a is given by its denotations: that is, by the set of world states of which z_a may be felicitously predicated, given the existing

context available to a listener. In probabilistic terms, this says that the meaning of a message z_a is represented by the distribution $p(x_a|z_a, x_b)$ it induces over speaker states. Examples of this approach include Guerin and Pitt [2001] and Pasupat and Liang [2016].

Pragmatic representation The meaning of a message z_a is given by the behavior it induces in a listener. In probabilistic terms, this says that the meaning of a message z_a is represented by the distribution $p(u_b|z_a, x_b)$ it induces over actions given the listener's observation x_b. Examples of this approach include Vogel et al. [2013a] and Gauthier and Mordatch [2016].

These two approaches can give rise to rather different behaviors. Consider the following example:



The top language (in blue) has a unique name for every kind of shape, while the bottom language (in red) only distinguishes between shapes with few sides and shapes with many sides. Now imagine a simple reference game with the following form: player a is covertly assigned one of these three shapes as a reference target, and communicates that reference to b; b must then pull a lever labeled `large` or `small` depending on the size of the target shape. (The circle is small and the other shapes are large.) Blue language speakers can achieve perfect success at this game, while red language speakers can succeed at best two out of three times.

How should we translate the blue word *hexagon* into the red language? The semantic approach suggests that we should translate *hexagon* as *many*: while *many* does not uniquely identify the hexagon, it produces a distribution over shapes that is closest to the truth. The pragmatic approach instead suggests that we should translate *hexagon* as *few*, as this is the only message that guarantees that the listener will pull the correct lever `large`. So in order to produce a correct listener action, the translator might have to "lie" and produce a maximally inaccurate listener belief.

If we were exclusively concerned with building a translation layer that allowed humans and DCP agents to interoperate as effectively as possible, it would be natural to adopt a pragmatic representation strategy. But our goals here are broader: we also want to facilitate *understanding*, and specifically to help users of learned systems form true beliefs about the systems' computational processes and representational abstractions. The example above

demonstrates that "pragmatically" optimizing directly for task performance can sometimes lead to translations that produce inaccurate beliefs.

We instead build our approach around semantic representations of meaning. By preserving semantics, we allow listeners to reason accurately about the content and interpretation of messages. We might worry that by adopting a semantics-first view, we have given up all guarantees of effective interoperation between humans and agents using a translation layer. Fortunately, this is not so: as we will see in Section 5.6, it is possible to show that players communicating via a semantic translator perform only boundedly worse (and sometimes better!) than pairs of players with a common language.

5.4 Translation Models

In this section, we build on the intuition that messages should be translated via their semantics to define a concrete translation model—a procedure for constructing a natural language $\leftrightarrow$ neuralese dictionary given agent and human interactions.

We understand the meaning of a message z_a to be represented by the distribution $p(x_a|z_a, x_b)$ it induces over speaker states given listener context. We can formalize this by defining the belief distribution β for a message z and context x_b as:

$$\beta(z_a, x_b) = p(x_a|z_a, x_b) = \frac{p(z_a|x_a)p(x_a, x_b)}{\sum_{x'_a} p(z_a|x'_a)p(x'_a, x_b)} \ .$$

Here we have modeled the listener as performing a single step of Bayesian inference, using the listener state and the message generation model (by assumption shared between players) to compute the posterior over speaker states. While in general neither humans nor DCP agents compute explicit representations of this posterior, past work has found that both humans and suitably-trained neural networks can be modeled as Bayesian reasoners [Frank et al., 2009, Paige and Wood, 2016].

This provides a context-specific representation of belief, but for messages z and z' to have the same semantics, they must induce the same belief over *all* contexts in which they occur. In our probabilistic formulation, this introduces an outer expectation over contexts, providing a final measure q of the quality of a translation from z to z':

$$\begin{aligned} q(z, z') &= \mathbb{E}\big[\mathcal{D}_{\mathrm{KL}}(\beta(z, X_b) \mid\mid \beta(z', X_b)) \mid z, z'\big] \\ &= \sum_{x_a, x_b} p(x_a, x_b|z, z')\mathcal{D}_{\mathrm{KL}}(\beta(z, x_b) \mid\mid \beta(z', x_b)) \\ &\propto \frac{1}{p(z')} \sum_{x_a, x_b} p(x_a, x_b) \cdot p(z|x_a) \cdot p(z'|x_a) \cdot \mathcal{D}_{\mathrm{KL}}(\beta(z, x_b) \mid\mid \beta(z', x_b)) \ ; \end{aligned} \tag{5.1}$$

Algorithm 3 Translating messages

given: a phrase inventory L
function TRANSLATE(z)
  return $\arg\min_{z' \in L} \hat{q}(z, z')$

function $\hat{q}(z, z')$
  // sample contexts and distractors
  $x_{ai}, x_{bi} \sim p(X_a, X_b)$ for $i = 1..n$
  $x'_{ai} \sim p(X_a | x_{bi})$
  // compute context weights
  $\tilde{w}_i \leftarrow p(z|x_{ai}) \cdot p(z'|x_{ai})$
  $w_i \leftarrow \tilde{w}_i / \sum_j \tilde{w}_j$
  // compute divergences
  $k_i \leftarrow \sum_{x \in \{x_{ai}, x'_{ai}\}} p(x|z, x_{bi}) \log \frac{p(x|z,x_{bi})}{p(x|z',x_{bi})}$
  return $\sum_i w_i k_i$

recalling that in this setting

$$\mathcal{D}_{\mathrm{KL}}(\beta \mid\mid \beta') = \sum_{x_a} p(x_a|z, x_b) \log \frac{p(x_a|z, x_b)}{p(x_a|z', x_b)} \tag{5.2}$$

which is zero when the messages z and z' give rise to identical belief distributions and increases as they grow more dissimilar. To translate, we would like to compute $tr(z_r) = \arg\min_{z_h} q(z_r, z_h)$ and $tr(z_h) = \arg\min_{z_r} q(z_h, z_r)$. Intuitively, Equation 5.1 says that we will measure the quality of a proposed translation $z \mapsto z'$ by asking the following question: in contexts where z is likely to be used, how frequently does z' induce the same belief about speaker states as z?

While this translation criterion directly encodes the semantic notion of meaning described in Section 5.3, it is doubly intractable: the KL divergence and outer expectation involve a sum over all observations x_a and x_b respectively; these sums are not in general possible to compute efficiently. To avoid this, we approximate Equation 5.1 by sampling. We draw a collection of samples (x_a, x_b) from the prior over world states, and then generate for each sample a sequence of distractors (x'_a, x_b) from $p(x'_a|x_b)$ (we assume access to both of these distributions from the problem representation). The KL term in Equation 5.1 is computed over each true sample and its distractors, which are then normalized and averaged to compute the final score.

Sampling accounts for the outer $p(x_a, x_b)$ in Equation 5.1. One of the two remaining quantities has the form $p(x_a|z, x_b)$. In the case of neuralese, can be obtained via Bayes' rule from the agent policy π_r. For natural language, we use transcripts of human interactions to fit a model that maps from frequent utterances to a distribution over world states as discussed in Section 5.2. The last quantity is a $p(z')$, the prior probability of the candidate

translation; this is approximated as uniform. The full translation procedure is given in Algorithm 3.

5.5 Modeling details

Agents Agents have the form shown in Figure 5.4, where h is a hidden state, z is a message from the other agent, u is a distribution over actions, and x is an observation of the world. A single hidden layer with 256 units and a tanh nonlinearity is used for the MLP. The GRU hidden state is also of size 256, and the message vector is of size 64.

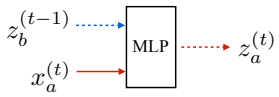
Agents are trained via interaction with the world as in Hausknecht and Stone [2015] using the ADAM optimizer [Kingma and Ba, 2014] and a discount factor of 0.9. The step size was chosen as 0.003 for reference games and 0.0003 for the driving game. An ϵ-greedy exploration strategy is employed, with the exploration parameter for timestep t given by:

$$\epsilon = \max \begin{cases} (1000 - t)/1000 \\ (5000 - t)/50000 \\ 0 \end{cases}$$

As in Foerster et al. [2016], we found it useful to add noise to the communication channel: in this case, isotropic Gaussian noise with mean 0 and standard deviation 0.3. This also helps smooth $p(z|x_a)$ when computing the translation criterion.

Representational models As discussed in Section 5.4, the translation criterion is computed based on the quantity $p(z|x)$. The policy representation above actually defines a distribution $p(z|x, h)$, additionally involving the agent's hidden state h from a previous timestep. While in principle it is possible to eliminate the dependence on h by introducing an additional sampling step into Algorithm 3, we found that it simplified inference to simply learn an additional model of $p(z|x)$ directly. For simplicity, we treat the term $\log(p(z')/p(z))$ as constant, those these could be more accurately approximated with a learned density estimator.

This model is trained alongside the learned agent to imitate its decisions, but does not get to observe the recurrent state, like so:



Here the multilayer perceptron has a single hidden layer with tanh nonlinearities and size 128. It is also trained with ADAM and a step size of 0.0003.

We use exactly the same model and parameters to implement representations of $p(z|x)$ for human speakers, but in this case the vector z is taken to be a distribution over messages in the natural language inventory, and the model is trained to maximize the likelihood of labeled human traces.

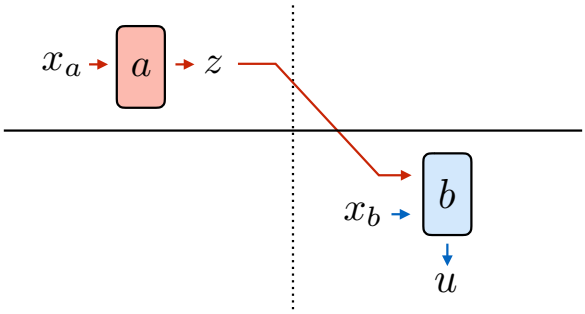


Figure 5.5: Simplified game representation used for analysis in Section 5.6. A speaker agent sends a message to a listener agent, which takes a single action and receives a reward.

5.6 Belief and Behavior

The translation criterion in the previous section makes no reference to listener actions at all. The shapes example in Section 5.3 shows that some model performance might be lost under translation. It is thus reasonable to ask whether this translation model of Section 5.4 can make any guarantees about the effect of translation on behavior. In this section we explore the relationship between belief-preserving translations and the behaviors they produce, by examining the effect of belief accuracy and strategy mismatch on the reward obtained by cooperating agents.

To facilitate this analysis, we consider a simplified family of communication games with the structure depicted in Figure 5.5. These games can be viewed as a subset of the family depicted in Figure 5.3; and consist of two steps: a listener makes an observation x_a and sends a single message z to a speaker, which makes its own observation x_b, takes a single action u, and receives a reward. We emphasize that the results in this section concern the theoretical properties of idealized games, and are presented to provide intuition about high-level properties of our approach. Section 5.8 investigates empirical behavior of this approach on real-world tasks where these ideal conditions do not hold.

Our first result is that translations that minimize semantic dissimilarity q cause the listener to take near-optimal actions:[2]

[2]Proof is provided in Section C.1.

Proposition 1.
Semantic translations reward rational listeners. Define a *rational listener* as one that chooses the best action in expectation over the speaker's state:

$$U(z, x_b) = \arg\max_u \sum_{x_a} p(x_a|x_b, z) r(x_a, x_b, u)$$

for a reward function $r \in [0, 1]$ that depends only on the two observations and the action.[3] Now let a be a speaker of a language r, b be a listener of the same language r, and b' be a listener of a different language h. Suppose that we wish for a and b' to interact via the translator $tr : z_r \mapsto z_h$ (so that a produces a message z_r, and b' takes an action $U(z_h = tr(z_r), x_{b'})$). If tr respects the semantics of z_r, then the bilingual pair a and b' achieves only boundedly worse reward than the monolingual pair a and b. Specifically, if $q(z_r, z_h) \leq D$, then

$$\mathbb{E}r(X_a, X_b, U(tr(Z)) \geq \mathbb{E}r(X_a, X_b, U(Z)) - \sqrt{2D} \tag{5.3}$$

So as discussed in Section 5.3, even by committing to a semantic approach to meaning representation, we have still succeeded in (approximately) capturing the nice properties of the pragmatic approach.

Section 5.3 examined the consequences of a mismatch between the set of primitives available in two languages. In general we would like some measure of our approach's robustness to the lack of an exact correspondence between two languages. In the case of humans in particular we expect that a variety of different strategies will be employed, many of which will not correspond to the behavior of the learned agent. It is natural to want some assurance that we can identify the DCP's strategy as long as *some* human strategy mirrors it. Our second observation is that it is possible to exactly recover a translation of a DCP strategy from a mixture of humans playing different strategies:

Proposition 2.
Semantic translations find hidden correspondences. Consider a fixed agent policy π_r and a set of human policies $\{\pi_{h1}, \pi_{h2}, \dots\}$ (recalling from Section 5.2 that each π is defined by distributions $p(z|x_a)$ and $p(u|z, x_b)$). Suppose further that the messages employed by these human strategies are *disjoint*; that is, if $p_{hi}(z|x_a) > 0$, then $p_{hj}(z|x_a) = 0$ for all $j \neq i$. Now suppose that all $q(z_r, z_h) = 0$ for all messages in the support of some $p_{hi}(z|x_a)$ and > 0 for all $j \neq i$. Then every message z_r is translated into a message produced by π_{hi}, and messages from other strategies are ignored.

[3]This notion of rationality is a fairly weak one: it permits many suboptimal communication strategies, and requires only that the listener do as well as possible given a fixed speaker—a first-order optimality criterion likely to be satisfied by any richly-parameterized model trained via gradient descent.

This observation follows immediately from the definition of $q(z_r, z_h)$, but demonstrates one of the key distinctions between our approach and a conventional machine translation criterion. Maximizing $p(z_h|z_r)$ will produce the natural language message most often produced in contexts where z_r is observed, regardless of whether that message is useful or informative. By contrast, minimizing $q(z_h, z_r)$ will find the z_h that corresponds most closely to z_r even when z_h is rarely used.

The disjointness condition, while seemingly quite strong, in fact arises naturally in many circumstances—for example, players in the driving game reporting their spatial locations in absolute vs. relative coordinates, or speakers in a color reference game (Figure 5.6) discriminating based on lightness vs. hue. It is also possible to relax the above condition to require that strategies be only *locally* disjoint (i.e. with the disjointness condition holding for each fixed x_a), in which case overlapping human strategies are allowed, and the recovered robot strategy is a context-weighted mixture of these.

5.7 Evaluation

Tasks

In the remainder of the paper, we evaluate the empirical behavior of our approach to translation. Our evaluation considers two kinds of tasks: reference games and navigation games. In a reference game (e.g. Figure 5.6a), both players observe a pair of candidate referents. A speaker is assigned a target referent; it must communicate this target to a listener, who then performs a choice action corresponding to its belief about the true target. Here consider two variants on the reference game: a simple color-naming task, and a more complex task involving natural images of birds. For examples of human communication strategies for these tasks, we obtain the XKCD color dataset [McMahan and Stone, 2015, Monroe et al., 2016] and the Caltech–UCSD Birds dataset [Welinder et al., 2010] with accompanying natural language descriptions [Reed et al., 2016]. In the color task, the input feature vector is simply the LAB representation of each color, and the message inventory taken to be all unigrams that appear at least five times. In the bird task, the model's input feature representations are a final 256-dimensional hidden feature vector from a compact bilinear pooling model [Gao et al., 2016] pre-trained for classification. The message inventory consists of the 50 most frequent bigrams to appear in natural language descriptions; example human traces are generated by for every frequent (bigram, image) pair in the dataset. We use standard train / validation / test splits for both of these datasets.

The final task we consider is the driving task (Figure 5.6c) first discussed in the introduction. In this task, two cars, invisible to each other, must each navigate between randomly assigned start and goal positions without colliding. This task takes a number of steps to complete, and potentially involves a much broader range of communication strategies. To obtain human annotations for this task, we recorded both actions and messages generated by pairs of human Amazon Mechanical Turk workers playing the driving game with each

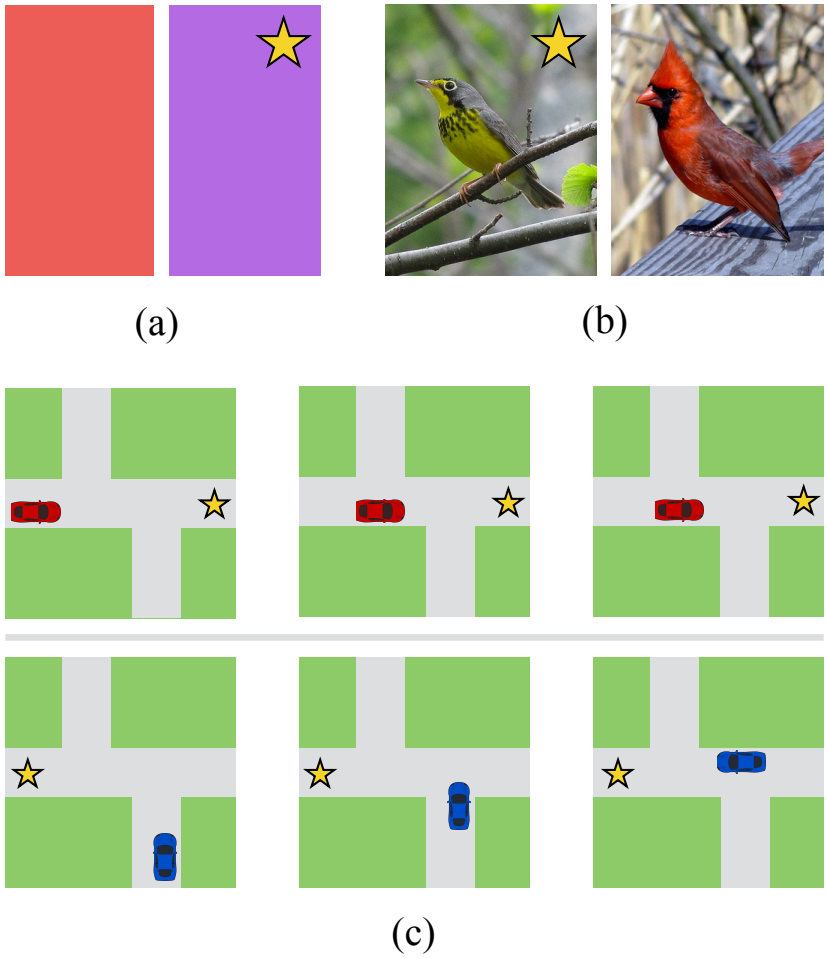


Figure 5.6: Tasks used to evaluate the translation model. (a–b) Reference games: both players observe a pair of reference candidates (colors or images); Player a is assigned a target (marked with a star), which player b must guess based on a message from a. (c) Driving game: each car attempts to navigate to its goal (marked with a star). The cars cannot see each other, and must communicate to avoid a collision.

other. We collected close to 400 games, with a total of more than 2000 messages exchanged, from which we held out 100 game traces as a test set. Driving data is collected from pairs of human workers on Mechanical Turk. Workers received the following description of the task:

> Your goal is to drive the red car onto the red square. Be careful! You're driving in a thick fog, and there is another car on the road that you cannot see. However, you can talk to the other driver to make sure you both reach your destinations safely.

Players were restricted to messages of 1–3 words, and required to send at least one message per game. Each player was paid $0.25 per game. 382 games were collected with 5 different road layouts, each represented as an 8x8 grid presented to players as in Figure 5.8. The action space is discrete: players can move forward, back, turn left, turn right, or wait. These were divided into a 282-game training set and 100-game test set. The message inventory consists of all messages sent more than 3 times. Input features consists of indicators on the agent's current position and orientation, goal position, and map identity.

Metrics

A mechanism for understanding the behavior of a learned model should allow a human user both to correctly infer its beliefs and to successfully interoperate with it; we accordingly report results of both "belief" and "behavior" evaluations.

To support easy reproduction and comparison (and in keeping with standard practice in machine translation), we focus on developing automatic measures of system performance. We use the available training data to develop simulated models of human decisions; by first showing that these models track well with human judgments, we can be confident that their use in evaluations will correlate with human understanding. We employ the following two metrics:

Belief evaluation This evaluation focuses on the denotational perspective in semantics that motivated the initial development of our model. We have successfully understood the semantics of a message z_r if, after translating $z_r \mapsto z_h$, a human listener can form a correct belief about the state in which z_r was produced. We construct a simple state-guessing game where the listener is presented with a translated message and two state observations, and must guess which state the speaker was in when the message was emitted.

When translating from natural language to neuralese, we use the learned agent model to directly guess the hidden state. For neuralese to natural language we must first construct a "model human listener" to map from strings back to state representations; we do this by using the training data to fit a simple regression model that scores (state, sentence) pairs using a bag-of-words sentence representation. We find that our "model human" matches the judgments of real humans 83% of the time on the colors task, 77% of the time on the birds task, and 77% of the time on the driving task. This gives us confidence that the model human gives a reasonably accurate proxy for human interpretation.

(a)

as listener \ as speaker	R	H	
R	1.00	0.50	random
		0.70	direct
		0.73	belief (ours)
H*	0.50	0.83	
	0.72		
	0.86		

(b)

as listener \ as speaker	R	H	
R	0.95	0.50	random
		0.55	direct
		0.60	belief (ours)
H*	0.50	0.77	
	0.57		
	0.75		

Table 5.1: Evaluation results for reference games. (a) The colors task. (b) The birds task. Whether the model human is in a listener or speaker role, translation based on belief matching outperforms both random and machine translation baselines.

Behavior evaluation This evaluation focuses on the cooperative aspects of interpretability: we measure the extent to which learned models are able to interoperate with each other by way of a translation layer. In the case of reference games, the goal of this semantic evaluation is identical to the goal of the game itself (to identify the hidden state of the speaker), so we perform this additional pragmatic evaluation only for the driving game. We found that the most reliable way to make use of human game traces was to construct a *speaker-only* model human. The evaluation selects a full game trace from a human player, and replays both the human's actions and messages exactly (disregarding any incoming messages); the evaluation measures the quality of the natural-language-to-neuralese translator, and the extent to which the learned agent model can accommodate a (real) human given translations of the human's messages.

Baselines We compare our approach to two baselines: a *random* baseline that chooses a translation of each input uniformly from messages observed during training, and a *direct* baseline that directly maximizes $p(z'|z)$ (by analogy to a conventional machine translation system). This is accomplished by sampling from a DCP speaker in training states labeled with natural language strings.

5.8 Results

In all below, "R" indicates a DCP agent, "H" indicates a real human, and "H*" indicates a model human player.

Reference games Results for the two reference games are shown in Table 5.1. The end-to-end trained model achieves nearly perfect accuracy in both cases, while a model trained to communicate in natural language achieves somewhat lower performance. Regardless of

Figure 5.7: Best-scoring translations generated for color task.

		as speaker R	as speaker H	
as listener	R	0.85	0.50	random
			0.45	direct
			0.61	belief (ours)
	H*	0.5	0.77	
		0.45		
		0.57		

Table 5.2: Belief evaluation results for the driving game. Driving states are challenging to identify based on messages alone (as evidenced by the comparatively low scores obtained by single-language pairs). Translation based on belief achieves the best overall performance in both directions.

whether the speaker is a DCP and the listener a model human or vice-versa, translation based on the belief-matching criterion in Section 5.4 achieves the best performance; indeed, when translating neuralese color names to natural language, the listener is able to achieve a slightly higher score than it is natively. This suggests that the automated agent has discovered a more effective strategy than the one demonstrated by humans in the dataset, and that the effectiveness of this strategy is preserved by translation. Example translations from the reference games are depicted in Figure 5.2 and Figure 5.7.

Driving game Behavior evaluation of the driving game is shown in Table 5.3, and belief evaluation is shown in Table 5.2. Translation of messages in the driving game is considerably more challenging than in the reference games, and scores are uniformly lower; however, a clear benefit from the belief-matching model is still visible. Belief matching leads to higher scores on the belief evaluation in both directions, and allows agents to obtain a higher reward

R / R	H / H	R / H	
1.93 / 0.71	— / 0.77	1.35 / 0.64	random
		1.49 / **0.67**	direct
		1.54 / 0.67	belief (ours)

Table 5.3: Behavior evaluation results for the driving game. Scores are presented in the form "reward / completion rate". While less accurate than either humans or DCPs with a shared language, the models that employ a translation layer obtain higher reward and a greater overall success rate than baselines.

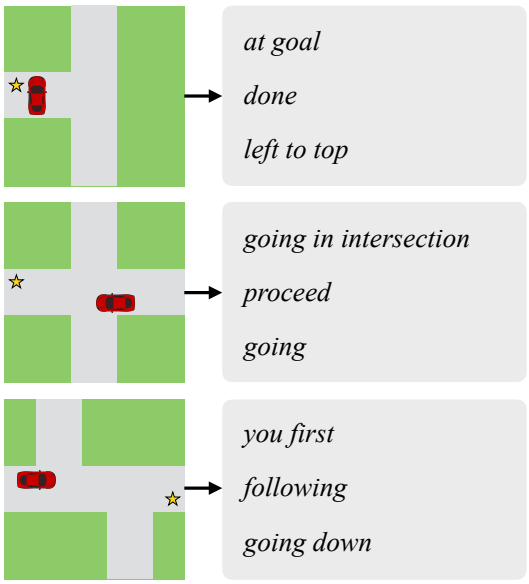


Figure 5.8: Best-scoring translations generated for driving task generated from the given speaker state.

on average (though task completion rates remain roughly the same across all agents). Some example translations of driving game messages are shown in Figure 5.8.

5.9 Discussion

We have investigated the problem of interpreting message vectors from deep networks by translating them. After introducing a translation criterion based on matching listener beliefs about speaker states, we presented both theoretical and empirical evidence that this criterion outperforms a conventional machine translation approach at recovering the content of message vectors and facilitating collaboration between humans and learned agents.

While our evaluation has focused on understanding the behavior of deep communicating

policies, the framework proposed here could be much more generally applied. Any encoder–decoder model [Sutskever et al., 2014] can be thought of as a kind of communication game played between the encoder and the decoder, so we can analogously imagine computing and translating "beliefs" induced by the encoding to explain what features of the input are being transmitted. The current work has focused on learning a purely categorical model of the translation process, supported by an unstructured inventory of translation candidates. Related techniques can also be used to explore *compositional* structure in messages, as discussed in subsequent work [Andreas and Klein, 2017].

More broadly, the work here shows that the denotational perspective from formal semantics provides a framework for precisely framing the demands of interpretable machine learning [Wilson et al., 2016, Doshi-Velez and Kim, 2017]. One major area of focus in interpretability is the study of visual features learned for image classification tasks. In this setting, one standard technique for generating visual explanations of an individual hidden unit is to aggregate the training examples for which that unit is maximally activated [Zeiler and Fergus, 2014]. The technique presented in this chapter may be viewed as extension of the standard visual approach, in which the set of aggregated images is subsequently summarized with a natural language string. Concurrent work [Bau et al., 2017] describes a similar idea for the restricted task of generating categorical "explanations" of this kind; future applications of the translation technique in this chapter might focus on scaling to complex, novel descriptions.

Chapter 6

Conclusion

This dissertation has presented a family of techniques for building models to solve language processing tasks and other learning problems, using linguistic structure to inform model structure. First, we presented a pair of modular deep architectures for question answering and reinforcement learning, and an approach for simultaneously learning the parameters of these modules and the larger structures into which they can be composed. Second, we presented an approach for learning and interpreting representations in a space parameterized by natural language. Using simple models for search in this space, we demonstrated that our approach outperforms standard baselines both on various model interpretability tasks and on downstream learning tasks like classification, structured prediction and reinforcement learning tasks. We believe that these results suggest the following overall conclusions:

Continuous representations improve expressiveness and learnability in semantic parsers: by replacing discrete predicates with differentiable neural network fragments, we bypass the challenging combinatorial optimization problem associated with induction of a semantic lexicon. In structured world representations, neural predicate representations allow the model to invent reusable attributes and relations not expressed in the schema. Perhaps more importantly, we can extend compositional question-answering machinery to complex, continuous world representations like images.

Linguistic structure encourages compositional generalization: by replacing a fixed neural architecture with an input-specific one, we can tailor the computation performed to each problem instance, using deeper networks for more complex questions and concepts and representing combinatorially many hypotheses with comparatively few parameters. In practice, this results in considerable gains in speed and sample efficiency, even with little training data. By forcing decisions to pass through a linguistic bottleneck in which the underlying compositional structure of concepts is explicitly expressed, efficient compositional generalization becomes possible.

Language simplifies structured exploration. Natural language scaffolding provides dramatic advantages in problems like reinforcement learning that require exploration: models with latent linguistic parameterizations can sample in this space, and limit exploration to a class of behaviors that are likely *a priori* to be goal-directed and interpretable.

And in general multitask settings, *language can help learning*—even in tasks for which no language data is available at training or test time. While some of these advantages are also provided by techniques built on top of formal languages, natural language is at once more expressive and easier to obtain than formal supervision. We believe this work hints at broader opportunities for using naturally-occurring language data to improve machine learning for problems of all kinds.

Appendix A

Policy Sketches

The complete list of tasks, sketches, and symbols is given below. Tasks marked with an asterisk* are held out for the generalization experiments described in Section 3.3, but included in the multitask training experiments in Section 3.3.

Goal	Sketch				
Crafting environment					
make plank	get wood	use toolshed			
make stick	get wood	use workbench			
make cloth	get grass	use factory			
make rope	get grass	use toolshed			
make bridge	get iron	get wood	use factory		
make bed*	get wood	use toolshed	get grass	use workbench	
make axe*	get wood	use workbench	get iron	use toolshed	
make shears	get wood	use workbench	get iron	use workbench	
get gold	get iron	get wood	use factory	use bridge	
get gem	get wood	use workbench	get iron	use toolshed	use axe
Maze environment					
room 1	left	left			
room 2	left	down			
room 3	right	down			
room 4	up	left			
room 5	up	right			
room 6	up	right	up		
room 7	down	right	up		
room 8	left	left	down		
room 9	right	down	down		
room 10	left	up	right		

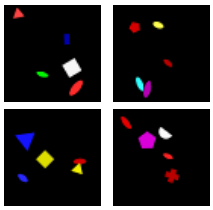
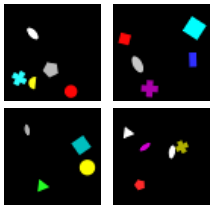
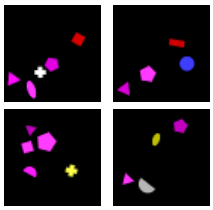
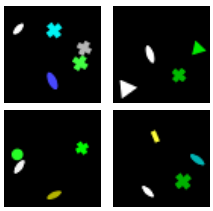
Cliff environment			
Goal	Sketch		
path 0	north		
path 1	east		
path 2	south		
path 3	west		
path 4	west	south	
path 5	west	north	north
path 6	north	east	north
path 7	west	north	
path 8	east	south	
path 9	north	west	west
path 10	east	north	east
path 11	south	east	
path 12	south	west	
path 13	south	south	
path 14	south	south	west
path 15	east	south	south
path 16	east	east	
path 17	east	north	
path 18	north	east	
path 19	west	west	
path 20	north	north	
path 21	north	west	
path 22	west	west	south
path 23	south	east	south

Appendix B

Latent Descriptions

B.1 Examples: ShapeWorld

Positive examples:

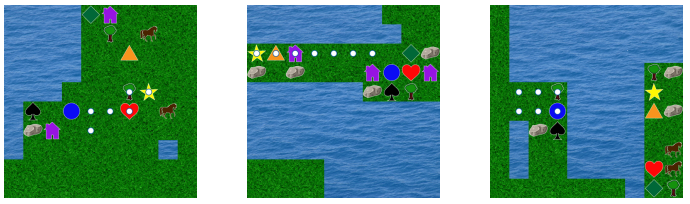
	**True description:** a red ellipse is to the right of an ellipse **Inferred description:** a red shape is to the right of a red semicircle	**Input:** 	**True label:** true **Pred. label:** true
	a shape is below a white ellipse a white shape is to the left of a yellow ellipse		false true
	a magenta triangle is to the left of a magenta pentagon a magenta triangle is to the left of a pentagon		true true
	a white ellipse is to the left of a green cross a green cross is to the right of a white ellipse		true true

B.2 Examples: Navigation

White breadcrumbs show the path taken by the agent.

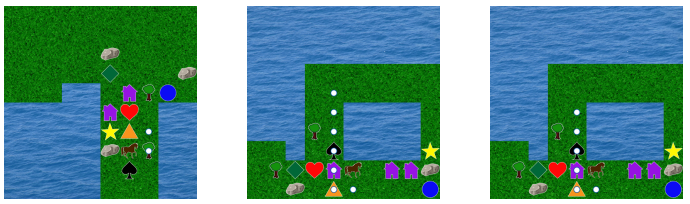
Human description:
`move to the star`

Inferred description:
`reach the star cell`



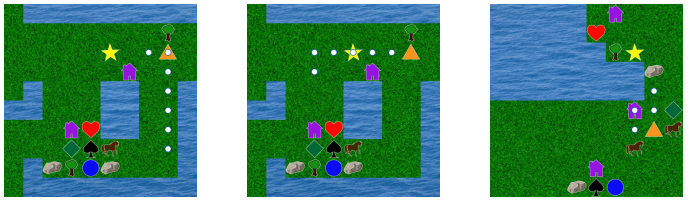
`reach square one right of triangle`

`reach cell to the right of the triangle`



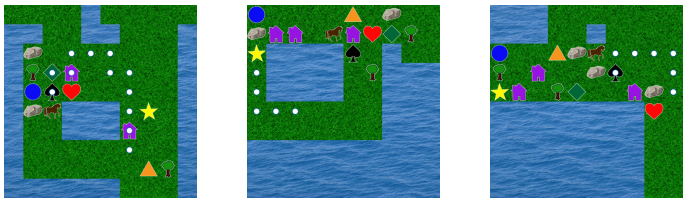
`reach cell on left of triangle`

`reach square left of triangle`



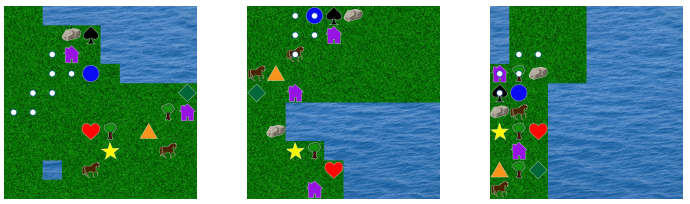
`reach spade`

`go to the spade`



`left of the circle`

`go to the cell to the left of the circle`



Appendix C

Translating Neuralese

C.1 Proof of Proposition 1

We know that

$$U(z, x_b) := \arg\max_u \sum_{x_a} p(x_a|x_b, z) r(x_a, x_b, z)$$

and that for all translations $(z, z' = t(r))$

$$D \geq \sum_{x_b} p(x_b|z, z') \mathcal{D}_{\mathrm{KL}}(\beta(z, x_b) \;||\; \beta(z', x_b)) \; .$$

Applying Pinsker's inequality:

$$\geq 2 \sum_{x_b} p(x_b|z, z') \delta(\beta(z, x_b), \beta(z', x_b))^2$$

and Jensen's inequality:

$$\geq 2 \Big(\sum_{x_b} p(x_b|z, z') \delta(\beta(z, x_b), \beta(z', x_b))) \Big)^2$$

so

$$\sqrt{D/2} \geq \sum_{x_b} p(x_b|z, z') \delta(\beta(z, x_b), \beta(z', x_b)) \; .$$

The next step relies on the following standard property of the total variation distance: for distributions p and q and a function f bounded by $[0, 1]$,

$$|\mathbb{E}_p f(x) - \mathbb{E}_q f(x)| \leq \delta(p, q) \; . \qquad (*)$$

For convenience we will write

$$\delta := \delta(\beta(z, x_b), \beta(z', x_b)) \ .$$

A listener using the speaker's language expects a reward of

$$\begin{aligned}
&\sum_{x_b} p(x_b) \sum_{x_a} p(x_a|x_b, z) r(x_a, x_b, U(z, x_b)) \\
&\leq \sum_{x_b} p(x_b) \Bigg(\sum_{x_a} p(x_a|x_b, z') r(x_a, x_b, U(z, x_b)) + \delta \Bigg)
\end{aligned}$$

via (*). From the assumption of player rationality:

$$\leq \sum_{x_b} p(x_b) \Bigg(\sum_{x_a} p(x_a|x_b, z') r(x_a, x_b, U(z', x_b)) + \delta \Bigg)$$

using (*) again:

$$\begin{aligned}
&\leq \sum_{x_b} p(x_b) \Bigg(\sum_{x_a} p(x_a|x_b, z) r(x_a, x_b, U(z', x_b)) + 2\delta \Bigg) \\
&\leq \sum_{x_a, x_b} p(x_a, x_b|z) r(x_a, x_b, U(z', x_b)) + \sqrt{2D} \ .
\end{aligned}$$

So the true reward achieved by a z'-speaker receiving a translated code is only additively worse than the native z-speaker reward:

$$\Bigg(\sum_{x_a, x_b} p(x_a, x_b|z) r(x_a, x_b, U(z, x_b)) \Bigg) - \sqrt{2D} \quad \square$$

Bibliography

David Andre and Stuart Russell. Programmable reinforcement learning agents. In *Advances in Neural Information Processing Systems*, 2001.

David Andre and Stuart Russell. State abstraction for programmable reinforcement learning agents. In *Proceedings of the Meeting of the Association for the Advancement of Artificial Intelligence*, 2002.

Jacob Andreas and Dan Klein. Reasoning about pragmatics with neural listeners and speakers. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2016.

Jacob Andreas and Dan Klein. Analogs of linguistic structure in deep representations. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2017.

Jacob Andreas, Andreas Vlachos, and Stephen Clark. Semantic parsing as machine translation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, Sofia, Bulgaria, 2013.

Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C Lawrence Zitnick, and Devi Parikh. VQA: Visual question answering. In *Proceedings of the International Conference on Computer Vision*, 2015.

Yoav Artzi and Luke Zettlemoyer. Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association for Computational Linguistics*, 2013.

Pierre-Luc Bacon and Doina Precup. The option-critic architecture. In *Proceedings of the Meeting of the Association for the Advancement of Artificial Intelligence*, 2017.

Bram Bakker and Jürgen Schmidhuber. Hierarchical reinforcement learning based on subgoal discovery and subpolicy specialization. In *Proceedings of the Conference on Intelligent Autonomous Systems*, 2004.

Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende Rezende, and Koray Kavukcuoglu. Interaction networks for learning about objects, relations and physics. In *Advances in Neural Information Processing Systems*, 2016.

David Bau, Bolei Zhou, Aditya Khosla, Aude Oliva, and Antonio Torralba. Network dissection: Quantifying interpretability of deep visual representations. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 2017.

Islam Beltagy, Cuong Chau, Gemma Boleda, Dan Garrette, Katrin Erk, and Raymond Mooney. Montague meets Markov: Deep semantics with probabilistic logical form. *Proceedings of the Joint Conference on Distributional and Logical Semantics*, 2013.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the International Conference on Machine Learning*, 2009.

Jonathan Berant and Percy Liang. Semantic parsing via paraphrasing. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2014.

Daniel S Bernstein, Robert Givan, Neil Immerman, and Shlomo Zilberstein. The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research*, 2002.

Mikhail Moiseevich Bongard. The recognition problem. Technical report, 1968.

Antoine Bordes, Sumit Chopra, and Jason Weston. Question answering with subgraph embeddings. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2014.

Léon Bottou. From machine learning to machine reasoning. *Machine learning*, 2014.

Léon Bottou, Yoshua Bengio, and Yann Le Cun. Global training of document processing systems using graph transformer networks. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 1997.

S.R.K. Branavan, Harr Chen, Luke S. Zettlemoyer, and Regina Barzilay. Reinforcement learning for mapping instructions to actions. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2009.

S.R.K. Branavan, David Silver, and Regina Barzilay. Learning to win by reading manuals in a Monte-Carlo framework. In *Proceedings of the Human Language Technology Conference of the Association for Computational Linguistics*, 2011.

Rich Caruana. Multitask learning. In *Learning to learn*. 1998.

Michael B Chang, Tomer Ullman, Antonio Torralba, and Joshua B Tenenbaum. A compositional object-based approach to learning physical dynamics. *arXiv preprint arXiv:1612.00341*, 2016.

David L. Chen and Raymond J. Mooney. Learning to interpret natural language navigation instructions from observations. In *Proceedings of the Meeting of the Association for the Advancement of Artificial Intelligence*, 2011.

Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. In *Proceedings of the Workshop on Syntax, Semantics and Structure in Statistical Translation*, 2014.

Ann A Copestake, Guy Emerson, Michael Wayne Goodman, Matic Horvat, Alexander Kuhnle, and Ewa Muszynska. Resources for building applications with Dependency Minimal Recursion Semantics. In *Proceedings of the Conference on Language Resources and Computation*, 2016.

Christian Daniel, Gerhard Neumann, and Jan Peters. Hierarchical relative entropy policy search. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, 2012.

Marie-Catherine De Marneffe and Christopher D Manning. The Stanford typed dependencies representation. In *Proceedings of the International Conference on Computational Linguistics*, 2008.

Misha Denil, Sergio Gómez Colmenarejo, Serkan Cabi, David Saxton, and Nando de Freitas. Programmable agents. *arXiv preprint arXiv:1706.06383*, 2017.

Coline Devin, Abhishek Gupta, Trevor Darrell, Pieter Abbeel, and Sergey Levine. Learning modular neural network policies for multi-task and multi-robot transfer. In *Proceedings of the International Conference on Robotics and Automation*, 2016.

Jacob Devlin, Jonathan Uesato, Surya Bhupatiraju, Rishabh Singh, Abdel-rahman Mohamed, and Pushmeet Kohli. RobustFill: Neural program learning under noisy I/O. In *Proceedings of the International Conference on Machine Learning*, 2017.

Jilles Steeve Dibangoye, Christopher Amato, Olivier Buffet, and François Charpillet. Optimally solving Dec-POMDPs as continuous-state MDPs. *Journal of Artificial Intelligence Research*, 2016.

Thomas G Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, 2000.

Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *Proceedings of the International Conference on Machine Learning*, 2014.

Jeffrey Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. Long-term recurrent convolutional networks for visual recognition and description. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 2015.

Finale Doshi-Velez and Been Kim. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608*, 2017.

Anca Dragan and Siddhartha Srinivasa. Generating legible motion. In *Robotics: Science and Systems*, 2013.

Markus Dreyer and Jason Eisner. Graphical models over multiple strings. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2009.

Yan Duan, John Schulman, Xi Chen, Peter L Bartlett, Ilya Sutskever, and Pieter Abbeel. RL^2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

Vittorio Ferrari and Andrew Zisserman. Learning visual attributes. In *Advances in Neural Information Processing Systems*, 2008.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the International Conference on Machine Learning*, 2017.

Jakob Foerster, Yannis M Assael, Nando de Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*, 2016.

Michael C Frank, Noah D Goodman, Peter Lai, and Joshua B Tenenbaum. Informative communication in word production and word learning. In *Proceedings of the Annual Conference of the Cognitive Science Society*, 2009.

Andrea Frome, Greg Corrado, Jonathon Shlens, Samy Bengio, Jeffrey Dean, Marc'Aurelio Ranzato, and Tomas Mikolov. DeViSE: A deep visual-semantic embedding model. In *Advances in Neural Information Processing Systems*, 2013.

Yang Gao, Oscar Beijbom, Ning Zhang, and Trevor Darrell. Compact bilinear pooling. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 2016.

Jon Gauthier and Igor Mordatch. A paradigm for situated and goal-driven language learning. *arXiv preprint arXiv:1610.03585*, 2016.

Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 2014.

Alison Gopnik and Andrew Meltzoff. The development of categorization in the second year and its relation to other cognitive and linguistic developments. *Child Development*, 1987.

Evan Greensmith, Peter L Bartlett, and Jonathan Baxter. Variance reduction techniques for gradient estimates in reinforcement learning. *Journal of Machine Learning Research*, 2004.

Edward Grefenstette. Towards a formal distributional semantics: Simulating logical calculi with tensors. In *Proceedings of the Joint Conference on Lexical and Computational Semantics*, 2013.

Edward Grefenstette, Karl Moritz Hermann, Mustafa Suleyman, and Phil Blunsom. Learning to transduce with unbounded memory. In *Advances in Neural Information Processing Systems*, 2015.

Frank Guerin and Jeremy Pitt. Denotational semantics for agent communication language. In *Proceedings of the International Conference on Autonomous Agents*, 2001.

Sumit Gulwani. Automating string processing in spreadsheets using input-output examples. *ACM SIGPLAN Notices*, 2011.

Stevan Harnad. The symbol grounding problem. *Physica D: Nonlinear Phenomena*, 1990.

Brent Harrison, Upol Ehsan, and Mark Riedl. Guiding reinforcement learning exploration using natural language. *arXiv preprint arXiv:1707.08616*, 2017.

Kris Hauser, Timothy Bretl, Kensuke Harada, and Jean-Claude Latombe. Using motion primitives in probabilistic sample-based planning for humanoid robots. In *Algorithmic Foundation of Robotics*. 2008.

Matthew Hausknecht and Peter Stone. Deep recurrent Q-learning for partially observable MDPs. In *Proceedings of the Meeting of the Association for the Advancement of Artificial Intelligence*, 2015.

Lisa Anne Hendricks, Zeynep Akata, Marcus Rohrbach, Jeff Donahue, Bernt Schiele, and Trevor Darrell. Generating visual explanations. In *Proceedings of the European Conference on Computer Vision*, 2016.

Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems*, 2015.

Ronghang Hu, Marcus Rohrbach, Jacob Andreas, Trevor Darrell, and Kate Saenko. Modeling relationships in referential expressions with compositional modular networks. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 2017.

Ronghang Hu, Jacob Andreas, Kate Saenko, and Trevor Darrell. Explainable neural computation via stack neural module networks. In *Proceedings of the European Conference on Computer Vision*, 2018.

Drew A Hudson and Christopher D Manning. Compositional attention networks for machine reasoning. *Proceedings of the International Conference on Learning Representations*, 2018.

Mohit Iyyer, Jordan Boyd-Graber, Leonardo Claudino, Richard Socher, and Hal Daumé III. A neural network for factoid question answering over paragraphs. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2014.

Michael Janner, Karthik Narasimhan, and Regina Barzilay. Representation learning for grounded spatial reasoning. *Transactions of the Association for Computational Linguistics*, 2017.

Robin Jia and Percy Liang. Data recombination for neural semantic parsing. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2016.

Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning. *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 2017a.

Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Judy Hoffman, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. Inferring and executing programs for visual reasoning. 2017b.

Sahar Kazemzadeh, Vicente Ordonez, Mark Matten, and Tamara L Berg. ReferItGame: Referring to objects in photographs of natural scenes. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2014.

Michael Kearns and Satinder Singh. Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 2002.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations*, 2014.

Emanuel Kitzelmann and Ute Schmid. Inductive synthesis of functional programs: An explanation based generalization approach. *Journal of Machine Learning Research*, 2006.

George Konidaris and Andrew G Barto. Building portable options: Skill transfer in reinforcement learning. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 2007.

George Konidaris, Scott Kuindersma, Roderic Grupen, and Andrew Barto. Robot learning from demonstration by constructing skill trees. *International Journal of Robotics Research*, 2011.

Jayant Krishnamurthy and Thomas Kollar. Jointly learning to parse and perceive: Connecting natural language to the physical world. *Transactions of the Association for Computational Linguistics*, 2013.

Jayant Krishnamurthy and Tom Mitchell. Vector space semantic parsing: A framework for compositional vector space models. In *Proceedings of the Workshop on Continuous Vector Space Models and their Compositionality*, 2013.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 2012.

Alexander Kuhnle and Ann Copestake. ShapeWorld: A new test methodology for multimodal language understanding. *arXiv preprint arXiv:1704.04517*, 2017.

Tejas D Kulkarni, Karthik R Narasimhan, Ardavan Saeedi, and Joshua B Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in Neural Information Processing Systems*, 2016.

M Pawan Kumar, Benjamin Packer, and Daphne Koller. Self-paced learning for latent variable models. In *Advances in Neural Information Processing Systems*, 2010.

Nate Kushman and Regina Barzilay. Using semantic unification to generate regular expressions from natural language. In *Proceedings of the Annual Meeting of the North American Chapter of the Association for Computational Linguistics*, 2013.

Tom Kwiatkowski, Luke Zettlemoyer, Sharon Goldwater, and Mark Steedman. Inducing probabilistic CCG grammars from logical form with higher-order unification. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2010.

Tom Kwiatkowski, Luke Zettlemoyer, Sharon Goldwater, and Mark Steedman. Lexical generalization in CCG grammar induction for semantic parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2011.

Tom Kwiatkowski, Eunsol Choi, Yoav Artzi, and Luke Zettlemoyer. Scaling semantic parsers with on-the-fly ontology matching. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2013.

Tessa Lau, Steven A Wolfman, Pedro Domingos, and Daniel S Weld. Programming by demonstration using version space algebra. *Machine Learning*, 2003.

Angeliki Lazaridou, Nghia The Pham, and Marco Baroni. Towards multi-agent communication-based language learning. *arXiv preprint arXiv:1605.07133*, 2016.

Angeliki Lazaridou, Alexander Peysakhovich, and Marco Baroni. Multi-agent cooperation and the emergence of (natural) language. In *Proceedings of the International Conference on Learning Representations*, 2017.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Yann LeCun, Sumit Chopra, Raia Hadsell, M Ranzato, and F Huang. A tutorial on energy-based learning. *Predicting structured data*, 2006.

Mike Lewis and Mark Steedman. Combining distributional and logical semantics. *Transactions of the Association for Computational Linguistics*, 2013.

Percy Liang, Michael Jordan, and Dan Klein. Learning dependency-based compositional semantics. In *Proceedings of the Human Language Technology Conference of the Association for Computational Linguistics*, Portland, Oregon, 2011.

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. Program induction by rationale generation: Learning to solve and explain algebraic word problems. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2017.

Mateusz Malinowski, Marcus Rohrbach, and Mario Fritz. Ask your neurons: A neural-based approach to answering questions about images. In *Proceedings of the International Conference on Computer Vision*, 2015.

Bhaskara Marthi, David Lantham, Carlos Guestrin, and Stuart Russell. Concurrent hierarchical reinforcement learning. In *Proceedings of the Meeting of the Association for the Advancement of Artificial Intelligence*, 2004.

Cynthia Matuszek, Nicholas FitzGerald, Luke Zettlemoyer, Liefeng Bo, and Dieter Fox. A joint model of language and perception for grounded attribute learning. In *Proceedings of the International Conference on Machine Learning*, 2012.

Brian McMahan and Matthew Stone. A Bayesian model of grounded color semantics. *Transactions of the Association for Computational Linguistics*, 2015.

Ishai Menache, Shie Mannor, and Nahum Shimkin. Q-cut: Dynamic discovery of sub-goals in reinforcement learning. In *Proceedings of the European Conference on Machine Learning*, 2002.

Yishu Miao and Phil Blunsom. Language as a latent variable: Discrete generative models for sentence compression. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2016.

Will Monroe, Noah D Goodman, and Christopher Potts. Learning to generate compositional color descriptions. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2016.

Richard Montague. The proper treatment of quantification in ordinary english. In *Approaches to natural language*. 1973.

Karthik Narasimhan, Regina Barzilay, and Tommi Jaakkola. Deep transfer in reinforcement learning by language grounding. *arXiv preprint arXiv:1708.00133*, 2017.

Arvind Neelakantan, Quoc V Le, and Ilya Sutskever. Neural programmer: Inducing latent programs with gradient descent. In *Proceedings of the International Conference on Learning Representations*, 2016.

Scott Niekum, Sarah Osentoski, George Konidaris, Sachin Chitta, Bhaskara Marthi, and Andrew G Barto. Learning grounded finite-state representations from unstructured demonstrations. *International Journal of Robotics Research*, 2015.

Hyeonwoo Noh, Paul Hongsuck Seo, and Bohyung Han. Image question answering using convolutional neural network with dynamic parameter prediction. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 2016.

Junhyuk Oh, Satinder Singh, Honglak Lee, and Pushmeet Kohli. Zero-shot task generalization with multi-task deep reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, 2017.

Brooks Paige and Frank Wood. Inference networks for Sequential Monte Carlo in graphical models. In *Proceedings of the International Conference on Machine Learning*, volume 48, 2016.

Ron Parr and Stuart Russell. Reinforcement learning with hierarchies of machines. In *Advances in Neural Information Processing Systems*, 1998.

Panupong Pasupat and Percy Liang. Compositional semantic parsing on semi-structured tables. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2015.

Panupong Pasupat and Percy Liang. Inferring logical forms from denotations. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2016.

PA Prather and Joshua Bacon. Developmental differences in part/whole identification. *Child development*, 1986.

Doina Precup. *Temporal abstraction in reinforcement learning*. PhD thesis, University of Massachusetts, Amherst, 2000.

John C Raven. Mental tests used in genetic studies: The performance of related individuals on tests mainly educative and mainly reproductive. *Unpublished master's thesis, University of London*, 1936.

Scott Reed and Nando de Freitas. Neural programmer-interpreters. In *Proceedings of the International Conference on Learning Representations*, 2015.

Scott Reed, Zeynep Akata, Honglak Lee, and Bernt Schiele. Learning deep representations of fine-grained visual descriptions. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 2016.

Mengye Ren, Ryan Kiros, and Richard S. Zemel. Image question answering: A visual semantic embedding model and a new dataset. In *Advances in Neural Information Processing Systems*, 2015.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Why should I trust you? explaining the predictions of any classifier. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, 2016.

Tim Rocktäschel and Sebastian Riedel. End-to-end differentiable proving. In *Advances in Neural Information Processing Systems*, 2017.

Stéphane Ross, Geoffrey J Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, 2011.

Maayan Roth, Reid Simmons, and Manuela Veloso. Reasoning about joint beliefs for execution-time communication decisions. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems*, 2005.

Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. Meta-learning with memory-augmented neural networks. In *Proceedings of the International Conference on Machine Learning*, 2016.

Jürgen Schmidhuber. Evolutionary principles in self-referential learning. *Diplom Thesis, Institut für Informatik, Technische Universität München*, 1987.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. Trust region policy optimization. In *Proceedings of the International Conference on Machine Learning*, 2015.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *Proceedings of the International Conference on Learning Representations*, 2016.

Rama Nath Sharma. *The astadhyayi of Panini: English translation of adhyayas seven and eight with Sanskrit text, transliteration, word-boundary, anuvrtti, vrtti, explanatory notes, derivational history of examples, and indices*. Munshirm Manoharlal Pub Pvt Limited, 2003.

K Simonyan and A Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Rishabh Singh and Sumit Gulwani. Learning semantic string transformations from examples. In *Proceedings of the International Conference on Very Large Databases*, 2012.

Jake Snell, Kevin Swersky, and Richard S Zemel. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems*, 2017.

Richard Socher, Brody Huval, Christopher Manning, and Andrew Ng. Semantic compositionality through recursive matrix-vector spaces. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2012.

Richard Socher, John Bauer, Christopher D. Manning, and Andrew Y. Ng. Parsing with compositional vector grammars. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2013.

Richard Socher, Andrej Karpathy, Quoc V Le, Christopher D Manning, and Andrew Y Ng. Grounded compositional semantics for finding and describing images with sentences. *Transactions of the Association for Computational Linguistics*, 2014.

Shashank Srivastava, Igor Labutov, and Tom Mitchell. Joint concept learning and semantic parsing from natural language explanations. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2017.

Martin Stolle and Doina Precup. Learning options in reinforcement learning. In *Proceedings of the International Symposium on Abstraction, Reformulation, and Approximation*, 2002.

Hendrik Strobelt, Sebastian Gehrmann, Bernd Huber, Hanspeter Pfister, and Alexander M Rush. Visual analysis of hidden state dynamics in recurrent neural networks. *IEEE Transactions on Visualization and Computer Graphics*, 2016.

Alane Suhr, Mike Lewis, James Yeh, and Yoav Artzi. A corpus of natural language for visual reasoning. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2017.

Sainbayar Sukhbaatar, Arthur Szlam, and Rob Fergus. Learning multiagent communication with backpropagation. In *Advances in Neural Information Processing Systems*, 2016.

Ilya Sutskever, Oriol Vinyals, and Quoc Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, 2014.

Richard S Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 1999.

Stefanie Tellex, Thomas Kollar, Steven Dickerson, Matthew R. Walter, Ashis Gopal Banerjee, Seth Teller, and Nicholas Roy. Understanding natural language commands for robotic navigation and mobile manipulation. In *In Proceedings of the National Conference on Artificial Intelligence*, 2011a.

Stefanie Tellex, Thomas Kollar, Steven Dickerson, Matthew R Walter, Ashis Gopal Banerjee, Seth Teller, and Nicholas Roy. Approaching the symbol grounding problem with probabilistic graphical models. *AI magazine*, 2011b.

Tijmen Tieleman. RMSProp (unpublished), 2012.

Geoffrey G Towell and Jude W Shavlik. Knowledge-based artificial neural networks. *Artificial Intelligence*, 1994.

Ramakrishna Vedantam, Samy Bengio, Kevin Murphy, Devi Parikh, and Gal Chechik. Context-aware captions from context-agnostic supervision. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 2017.

Alexander Vezhnevets, Volodymyr Mnih, John Agapiou, Simon Osindero, Alex Graves, Oriol Vinyals, and Koray Kavukcuoglu. Strategic attentive writer for learning macro-actions. In *Advances in Neural Information Processing Systems*, 2016.

Oriol Vinyals, Charles Blundell, Tim Lillicrap, Koray Kavukcuoglu, and Daan Wierstra. Matching networks for one shot learning. In *Advances in Neural Information Processing Systems*, 2016.

Adam Vogel and Dan Jurafsky. Learning to follow navigational directions. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2010.

Adam Vogel, Max Bodoia, Christopher Potts, and Daniel Jurafsky. Emergence of Gricean maxims from multi-agent decision theory. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, 2013a.

Adam Vogel, Christopher Potts, and Dan Jurafsky. Implicatures and nested beliefs in approximate decentralized-pomdps. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2013b.

P. Welinder, S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, and P. Perona. Caltech-UCSD Birds 200. Technical Report CNS-TR-2010-001, California Institute of Technology, 2010.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 1992.

Andrew Gordon Wilson, Been Kim, and William Herlands. Proceedings of NIPS Workshop on Interpretable Machine Learning for Complex Systems. 2016.

Terry Winograd. Understanding natural language. *Cognitive psychology*, 1972.

Yuk Wah Wong and Raymond Mooney. Learning for semantic parsing with statistical machine translation. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pages 439–446, New York, New York, 2006.

Yuk Wah Wong and Raymond J. Mooney. Learning synchronous grammars for semantic parsing with lambda calculus. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2007.

Huijuan Xu and Kate Saenko. Ask, attend and answer: Exploring question-guided spatial attention for visual question answering. In *Proceedings of the European Conference on Computer Vision*, 2016.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Aaron Courville, Ruslan Salakhutdinov, Richard Zemel, and Yoshua Bengio. Show, attend and tell: neural image caption generation with visual attention. In *Proceedings of the International Conference on Machine Learning*, 2015.

Zichao Yang, Xiaodong He, Jianfeng Gao, Li Deng, and Alex Smola. Stacked attention networks for image question answering. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 2017.

Pengcheng Yin, Zhengdong Lu, Hang Li, and Ben Kao. Neural enquirer: Learning to query tables. In *Proceedings of the International Joint Conference On Artificial Intelligence*, 2017.

Licheng Yu, Hao Tan, Mohit Bansal, and Tamara L Berg. A joint speaker-listener-reinforcer model for referring expressions. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 2017.

Matthew D Zeiler. ADADELTA: An adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *Proceedings of the European Conference on Computer Vision*, 2014.

Bolei Zhou, Yuandong Tian, Sainbayar Sukhbaatar, Arthur Szlam, and Rob Fergus. Simple baseline for visual question answering. *arXiv preprint arXiv:1512.02167*, 2015.