Reinforcement Learning with Natural Language Signals

by

Szymon Sidor

B.A., University of Cambridge (2013)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2016

Author ..... Signature redacted
Department of Electrical Engineering and Computer Science
January 29, 2016

Certified by ..... Signature redacted
Brian C. Williams
Professor of Aeronautics and Astronautics
Thesis Supervisor

Accepted by ..... Signature redacted
Professor Leslie A. Kolodziejski
Chair, Department Committee on Graduate Theses

Reinforcement Learning with Natural Language Signals

by

Szymon Sidor

Submitted to the Department of Electrical Engineering and Computer Science
on January 29, 2016, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science and Engineering

Abstract

In this thesis we introduce a technique that allows one to use Natural Language as part of the state in Reinforcement Learning. We show that it is capable of solving Natural Language problems, similar to Sequence-to-Sequence models, but using multi-stage reasoning. We use Long Short-Term Memory Networks to parse the Natural Language input, whose final hidden state is used to compute action scores for the Deep Q-learning algorithm.

First part of the thesis introduces the necessary theoretical background, including Deep Learning approach to Natural Language Processing, Recurrent Neural Networks and Sequence-to-Sequence modeling. We consider two case studies: translation and dialogue. In addition, we provide an overview of the existing techniques for the Reinforcement Learning problems, with focus on Deep Q-learning algorithm.

In the second part of the thesis we present the multi-stage reasoning approach, and demonstrate it on solving the sentence unshuffling problem. It achieves accuracy 5% better than a Sequence-to-Sequence model, while requiring 3 times less examples to converge. Furthermore, we show that our approach is flexible and can be used with multi-modal inputs - Natural Language and agent's sensory data. We propose a system capable of understanding and executing Natural Language commands. It can be used for many different tasks with minimal engineering effort - the only required components being the reward function and example commands. We demonstrate its performance using an experiment in which an agent is required to learn to complete four types of manipulation tasks. The approach achieves nearly perfect performance on two of them and good performance in the two others.

Thesis Supervisor: Brian C. Williams
Title: Professor of Aeronautics and Astronautics

Acknowledgments

The work in this thesis would not have been possible without the incredible support of my advisor, Prof. Brian C. Williams. I would like to thank all my friends. Peng Yu for his support, contributions and helping me understand how to navigate academic environment. Jonathan Raiman for introducing me to Deep Learning and all his awesome ideas, without which this thesis would not be the same. Jaroslaw Blasiok for his invaluable insights. Jakub Pachocki for convincing me that academia can be as exciting as the industry. Probably the hardest job among all the people who supported me in those endeavor was hearing all the complaints about the amount of work I have left and for that I am very grateful to my parents.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Since the dawn of computing, it has been apparent that some problems, which people solve effortlessly, are difficult to implement in software. Artificial Intelligence (AI) is a field of study which aims to bridge this gap. AI is one of the central problems of XXI century with far reaching implications. Metamind's[1] image classification software performs medical diagnosis, allowing professionals to focus on improving patients' condition. Google, Tesla and numerous others are working on self-driving cars, which can improve road safety (In USA alone over 30000 people die in car accidents every year since 2000 [2]). Researchers at MIT have shown that intelligent robots can participate in disaster relief especially in the areas that are too far away or too dangerous for human rescuers to access [49]. Those are only a few examples of the applications of AI.

There are two main approaches to advancing AI. Symbolic AI sets out to build an elegant model of the environment using logic formulations and to make intelligent decisions using inference. Statistical AI is about learning probabilistic representations of knowledge and making decisions based on likelihood of different scenarios. Intuitively, statistical approach tends to result in systems which are fast, but often inexact or even incorrect. Symbolic approach is often slower, exploring many scenarios before providing an answer which is often optimal given the model. For those

[1] `www.metamind.io`

[2] `http://www-fars.nhtsa.dot.gov/Main/index.aspx`

readers familiar with the book "Thinking Fast and Slow" [19], Statistical AI could be compared to System 1 (fast, unconscious, automatic, error prone), while Symbolic AI is closer to System 2 (slow, conscious, effortful and reliable).

1.1 Symbolic AI

Success Stories Even though inference problems considered in Symbolic AI are NP-complete, the researchers have managed to solve problems of impressive sizes. For example in 1996, IBM has built a computer that won a chess game with grandmaster Garry Kasparov [4], which evaluated 200 million chess moves per second. Furthermore, while representing knowledge as a set of logical equations may seem restrictive, it has been successfully applied to solve diverse high-level problems, such as diagnosis and fault isolation [7], recognizing and executing tasks in human robot teams [25], or even understanding the notion of risk and discovering execution strategies with upper bounds on probability of failure [48].

Shortcomings At the time of writing (2016), the average cost of hiring a computer engineer for one year is about $100 000, while the cost of maintaining a server for a year is $1000. It means that minimizing the amount of engineering effort is at least as important as computational performance, when choosing an approach for a particular problem. One of the main shortcomings of the symbolic approach is the fact that the amount of engineering work, required to build a model, can be significant. For example, [47] demonstrates a system capable of controlling a robotic forklift based on gestures and natural language. The proposed system requires many components working in tandem: context analyzer, task planner, motion planner, navigator, object detector, and manipulation planner. A system with such a large number of components has two major disadvantages. First of all the raw complexity of the system correlates negatively with its robustness - every interaction between different components requires additional software and may introduce new bugs. Secondly, such a system is difficult to extend. For example, updating the functionality to be able to

interpret new set of commands requires significant effort from the programmers, who must have comprehensive knowledge of the system.

1.2 Statistical AI

Success Stories Recently, significant progress has been made in demonstrating the usefulness of Neural Network methods, or more generally Non-Convex Optimization [3] [4]. In the past 5 years it has been applied to solve a vast array of problems. Convolutional Neural Networks significantly surpassed previous state of the art in object recognition [22]. Recurrent Neural Networks achieved state of the art results in Machine Translation [41]. A combination of both types of networks yielded human-level accuracy in speech recognition [1]. Furthermore, Convolutional Neural Networks used as a function approximator in Bellman equation were used to learn controllers that achieve high scores on many Atari games based solely on the image of the screen and reward signal [30].

Shortcomings While Non-Convex Optimization methods rely very little on model engineering, they instead utilize vast quantities of data. For example, the object recognition network was trained using 15 million labeled images. Machine Translation experiment leveraged 12 million English-French translation pairs. Speech recognition network was trained on 11,940 hours of speech including 8 million utterances, each of which was labeled with corresponding English transcription. There have been attempts at addressing this issue, the most notable of which is Transforming Autoencoders [45], which achieved 1.74% error rate on the MNIST task (classifying handwritten digits) with only 25 labeled examples (in an active learning setting). Nevertheless such methods have never been validated on any problem of substantial

[3]The author has decided to use the name Non-Convex optimization due to the fact that some of the models presented in this thesis go well beyond the original framework of a backpropagation based computation that can be represented as a single fully differentiable computational graph. The name is overly general, but it is the best the author could come up with.

[4]Non-Convex optimization is only one of many areas of studies considered to be a part of Statistical AI. However, the author believes they are representative enough to demonstrate many of the advantages and disadvantages of this type of techniques.

Figure 1-1: A picture of a horse wearing a fly-repelling rug. When classified using state of the art 1000-class Convolutional Neural Network provided by Metamind (`http://metamind.io`) it reports Zebra as the highest probability answer, while the horse does not occur in top 5 answers. image source: `http://www.equiporium.co.uk/`

size.

Furthermore, despite the remarkable success of Non-Convex optimization methods, there still are certain qualitative differences between the predictions made by humans, and those made by even the best Neural Network models. Consider figure 1-1 - even though the picture has many features that may suggest the entity is a zebra, we can

easily refute that hypothesis, in contrast to Neural Network. Notice, that most of the people can accurately classify the image, even if it is the first instance of "horse dressed as zebra" they have seen in their life.

1.3 The renaissance of Reinforcement Learning

Reinforcement Learning (RL) is a technique that allows one to learn a strategy for choosing an action given the current knowledge, such that the current and future rewards are maximized. It is a promising approach addressing some of the shortcomings of both Symbolic and Statistical AI:

Reinforcement Learning can leverage existing and future statistical models In 1992 it was demonstrated that Reinforcement Learning can be used to achieve above-human performance on the game of TD-gammon [43]. It used Multi-Layer Perceptron as approximator of the value function. Unfortunately, due to poor convergence characteristics and computational performance, the approach was deemed impractical for problems of bigger sizes. However, in 2013 the Deep Q-learning algorithm addressed some of those issues and was applied to achieve above-human performance on many Atari games, based solely on the image of the screen and the score function [30]. It was impactful, not only because it solved a problem much larger than possible before, but also due to the fact that it used a complex statistical model as part of the Q-value approximator - Convolutional Neural Networks [24]. One of the main assumptions in this thesis is that the family of models that can be employed in that role is much wider, which we validate using Long Short-term Memory networks as an example.

Reinforcement Learning models are capable of multi-step reasoning Some problems inherently require multiple steps of reasoning, where a part of computation cannot be done before the previous one is completed. To better understand this issue, let's consider a probabilistic equivalent of a parallel processing framework (chapter 27 of [6]). Given a particular computation, one can find a critical path - the shortest

path of elementary operations, such that the next one cannot be executed before the first one finishes. The longer the critical path, the harder the algorithm is to parallelize. In a recent paper by Facebook AI [52], a set of 20 simple problems were designed to be prerequisites for any system "capable of conversing with human". One of the problems on which their Neural-Network-based system was performing particularly poorly was path finding. The critical path for this problem is as long as the length of the shortest solution. It is likely that the main reason for such a poor performance is the inability of Neural Networks to perform multi-stage reasoning. Another example is a competition organized by Kaggle, where the goal was to design an algorithm for recognizing whales (particular individuals, rather than species) based on their aerial photographs [5]. Both winning teams have reported in their blog posts [6] [7] that while naive application of Convolutional Neural Networks (CNN) failed to achieve performance higher than chance, significant improvement was obtained by breaking the problem up into pieces - one CNN identifying key feature for alignment and another classifying the aligned image. Reinforcement Learning approach could potentially address some of those problems in end-to-end fasion. It allows for multi-stage reasoning - multiple actions can be part of a single execution of the model.

1.4 Reinforcement Learning for Natural Language Processing

Natural Language Processing has two qualities that are desirable for evaluating Artificial Intelligence algorithms. Firstly, Natural Language is very diverse - the same message can be expressed in many different ways; syntax can be varied arbitrarily, while semantics remains unchanged; regardless of what I am trying to say, there are many ways to verbalize it. Secondly, understanding language often requires multi-step reasoning, e.g. "All roses are flowers. Some flowers fade quickly. Do some roses

[5] https://www.kaggle.com/c/noaa-right-whale-recognition
[6] http://deepsense.io/deep-learning-right-whale-recognition-kaggle/
[7] http://felixlaumon.github.io/2015/01/08/kaggle-right-whale.html

fade quickly?" (example from [19]), requires a few steps of logical inference to answer correctly.

While it is interesting from theoretical standpoint, Natural Language Processing has many practical applications. The most pervasive one is Information Retrieval. Google search engine performs over 3.5 billion searches per day [40]. There are also many specialized companies, that sell products capable of navigating specific types of data, such as Sumologic, Palantir etc. Another application is understanding spoken commands, which comes in two flavors - personal assistants such as Siri, Google Now or Cortana, and intelligent robotic systems, e.g. Moley (robotic kitchen). Especially in case of the later category, improving the underlying algorithms is a matter of not only convenience and efficiency, but also safety; in 2015 a manufacturing robot pressed a worker against a metal plate, leading to his death [8]. This accident could have been avoided if the machine understood spoken commands or had better comprehension of its surroundings.

Using Reinforcement Learning in Natural Language problems is not a new idea. For example, it has been demonstrated as a viable approach for teaching an agent to play Civilization II games based on game manual[3] and playing text-based games like Fantasy World[33]. However, the benefit of using Non-Convex approach is that it requires little modeling effort - it does not need part of speech tags, parse trees or mapping for words to actions.

In this thesis we show how Reinforcement Learning can be used for Natural Language Processing by incorporating language as part of the state. To accomplish that we use Long Short-Term Memory Networks to parse the Natural Language input, where the final hidden state of the network is used as the input to Multi-Layer Perceptron computing action scores.

To verify the claim that multi-step reasoning in Reinforcement Learning settings can lead to improvements on Natural Language Processing tasks, we evaluate our approach on sentence unshuffling problem. Our approach achieves performance 5% better than Sequence-to-Sequence approach, while requiring 3 times less examples

[8] `http://www.cnn.com/2015/07/02/europe/germany-volkswagen-robot-kills-worker/`

for convergence. It is worth noting that no hand-engineered features are used and the reward function is simple (the number of words put in correct positions). To further demonstrate applicability of our approach, we trained a model to perform manipulation tasks based on Natural Language commands. It showcases the flexibility of our approach by incorporating not only Natural Language, but also sensory input as part of the state available to the agent performing the task. The model jointly learns to understand and execute, which means the only necessary, domain-specific engineering is designing the reward function and providing examples of commands. It achieves perfect performance on two of the tasks, that we proposed, and good performance on the other two (significantly better than chance).

1.5 Overview of the thesis

The remainder of the thesis is structured in the following way. The second chapter describes Non-Convex methods in Natural Language Processing, including the theory and two use cases - Machine Translation and Dialogue. The third chapter lays out the theory of Reinforcement Learning with particular focus on the Deep Q-learning algorithm, as well as a simple experiment verifying its properties. The fourth chapter demonstrates how the two can be combined in two experiments. The first experiment shows how a simple NLP task - sentence unshuffling - can benefit from being posed as a Reinforcement Learning problem. Second experiment demonstrates how Reinforcement Learning agent can be thought to execute Natural Language commands based on their Natural Language description and sensory input.

Chapter 2

Natural Language Processing as a Non-Convex Optimization Problem

The aim of this chapter is to familiarize the reader with Non-Convex techniques in Natural Language Processing. The reason for that is twofold. Firstly, a family of models - Sequence-to-Sequence Recurrent Neural Networks - is currently used in the state of the art algorithms for many NLP problems, e.g. translation [41] or sentiment analysis [35]. We will use such a model as a baseline for one of the experiments in chapter 4. Secondly, we illustrate how to approach constructing Non-Convex architectures with focus on the ones useful in Natural Language Processing. This is necessary to understand the key features of our model for Reinforcement Learning using Natural Language as part of the state.

The chapter starts with a brief introduction to Non-Convex optimization techniques, including relevant architectures, error functions and optimization techniques. After that, we introduce the key insight for Non-Convex Optimization approach to Natural Language Processing - Word Vectors. To better understand how this knowledge fits together, we study two use cases - Translation and Dialogue Management. We conclude this chapter with a short experiment, hinting at the way the Non-Convex models are useful for encoding Natural Language as part of the state in Reinforcement Learning.

2.1 Background

Natural Language Processing problems have been traditionally solved by combining building blocks composed mainly of hand designed features and classifiers [27]. In recent years researchers have managed to demonstrate the usefulness of end-to-end non-convex optimization approaches, which learn to solve complex Natural Language Processing tasks based on example data. It has been demonstrated that such models can solve many of the basic NLP tasks, such as: part-of-speech tagging, chunking, named entity recognition and semantic role labeling [5]. Furthermore, such an end-to-end approach achieves competitive results on fairly complex problems like Machine Translation [41], Sentiment Analysis [38], and simple dialogue problems [53].

Using non-convex optimization for Natural Language is a relatively old idea. In 1986, Geoffrey Hinton has demonstrated this type of approach for learning the meaning of relationships in a genealogy tree from simple statements about an example family [16]. However only in recent years did the community manage to demonstrate competitive results for real-world applications. This is a result of the combination of hardware improvements and better theoretical understanding of non-convex optimization. The former is important due to the fact that those models can be very computationally expensive - for example in NLP we often encounter matrices with size (number of words in language) $\times$ (size of distributed representation), which can often have 100 000 000 or more entries. An example of the latter, a theoretical breakthrough, would be a set of recently developed techniques for training long term dependencies in Recurrent Neural Network [17].

This chapter describes the suite of algorithms used in non-convex approach for NLP followed by two case studies - Machine Translation and Dialogue Management.

2.2 Introduction to Non-Convex Optimization

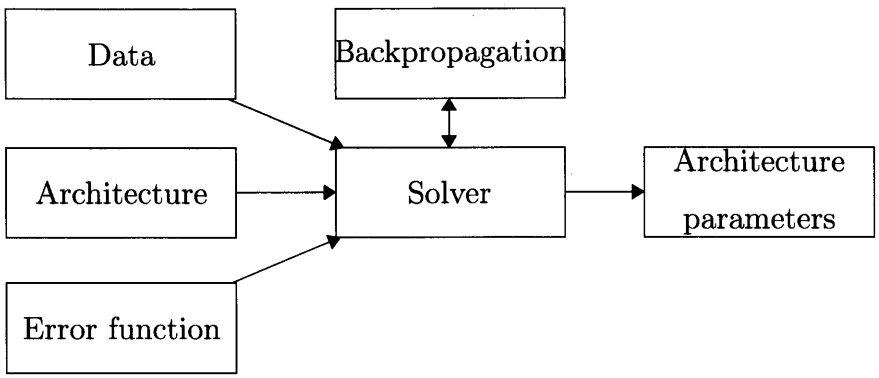


Figure 2-1: Overview of the non-convex optimization pipeline

Figure 2-1 outlines typical non-convex optimization pipeline in Machine Learning setting. For a given problem we define an architecture which depends on some parameters. We would like to tune the parameters, such that the architecture performs well on our problem subject to an error function, which estimates performance of our model on some data. The key observation is that if the architecture and error functions are differentiable, we can use backpropagation algorithm to compute a derivate of the error function. This information can be subsequently used by a solver like Stochastic Gradient Descent algorithm to find a set of parameters for the architecture that achieves low values of the error function.

2.2.1 The Data

Although theoretical analysis of non-convex optimization techniques remains incomplete as of today, many experiments have shown that performance *on unseen examples* for a given problem can always be improved in presence of more data.

Data may have many shapes and forms depending on the scenario that we are interested in, but majority of the Non-Convex models operate in the *supervised* setting, which is defined in the following way: given a set of example input and output pairs (x_1, y_1), ..., (x_n, y_n) which have been sampled from some distribution $P(X, Y)$, find a

model that approximates $P(Y|X)$. Example would be a set of sentences where each word is tagged with their part of speech.

Collecting the data can be quite a challenge and often requires certain level of creativity. For example the imagenet dataset [8] created for the purpose of 1000-class image recognition contained 3.2 million *labeled* images initially (current version exceeds 11 million). The main trick was to use publicly accessible image search engine to find candidate images and then ask humans to judge whether the label is correctly assigned, which requires much less time and effort than looking up every image separately.

There are many open problems as far as data acquisition is concerned. For example assembling a large question answering corpus is an unsolved challenge. Such corpus is of significant practical interest. For example Facebook recently (2015) invested considerable resource in deploying natural language query personal assistant, which is only partially automated, with the main goal of the project being data acquisition [1].

2.2.2 The Architecture

When choosing a non-convex optimization architecture, one must take multiple factors into account. Is it capable of expressing the desired relationship - does the "shape" of data fit the inputs/outputs of the architecture? Does it encode meaningful relationship that we expect to see in data - for example location invariance in case of Convolutional Neural Networks? Finally, can its parameters be optimized using SGD or some other algorithm; does it suffer from exploding and vanishing gradient problem or some other problems? Due to the fact that some of the first non-convex optimization architectures were neurobiologically inspired, they are often called Neural Networks. In this section we present a few architectures that have been successfully employed to solve real-life problems.

[1] `https://www.facebook.com/Davemarcus/posts/10156070660595195`

Multi-layer Perceptron

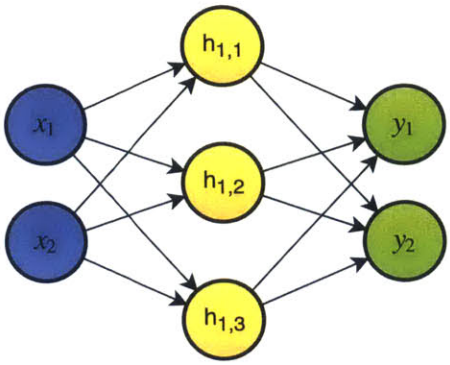


Figure 2-2: Example of a Multi-Layer Perceptron with input vector of size 2, hidden state vector of size 3 and output vector of size 2.

The simplest non-convex model is a Multi-Layer Perceptron (MLP). Simply put, it is a sequence of linear mapping interleaved with nonlinearities (nonlinear functions). Example nonlinearities include sigmoid function (σ), $tanh$, or more recently rectified linear units (often abbreviated ReLU). An example of a single hidden layer MLP is presented on figure 2-2. Formally MLP is defined by the set of equations:

$$h_1 = \tau_1(W_1 x + b_1)$$
$$h_2 = \tau_2(W_2 h_1 + b_2)$$
$$...$$
$$y = \tau_n(W_n h_{n-1} + b_n)$$

Where x is the input vector, y is the output vector and $h_1, ..., h_{n-1}$ are the intermediate hidden vectors (also known as hidden layers). Additionally network has a set of n linear transformation defined by $(W_1, b_1), ..., (W_n, b_n)$ which are learned. The number and sizes of layers n and the type of nonlinearities $\tau_1, ..., \tau_n$ used varies by the network and is usually tuned for particular application. The nonlinearities are often designed to resemble a step function while maintaining differentiability. One can think about a MLP as if every number in a hidden vector and the output was a Logistic Regression classifier, using previous vector as a set of input features. This is why MLP is often

called a hierarchical classifier. Example use case for MLP would be one where the input is set of frequencies of word occurrences in a document and the output is that document's topic. As we will find out in chapter **??** MLP is the key building block of the Deep Q-learning algorithm.

Recurrent Neural Network

MLPs can be useful for fixed size inputs, but what about the cases where input size varies by example such as natural language sentences? One solution would be to break the problem into pieces and consider each piece independently using MLP. However in many cases it is the interaction between subsequent elements of a sequence that carries the most information. This is particularly important for Natural Language Applications. For example, if a verb is preceded by the word *not*, its meaning dramatically changes. Recurrent Neural Network (RNN) is a type of non-convex model that address those concerns. RNN is essentially a sequence of MLPs, but when computing a hidden vector it takes into account not only the input, but also the value of hidden vector preceding it in the sequence. An example RNN is presented in figure 2-3.

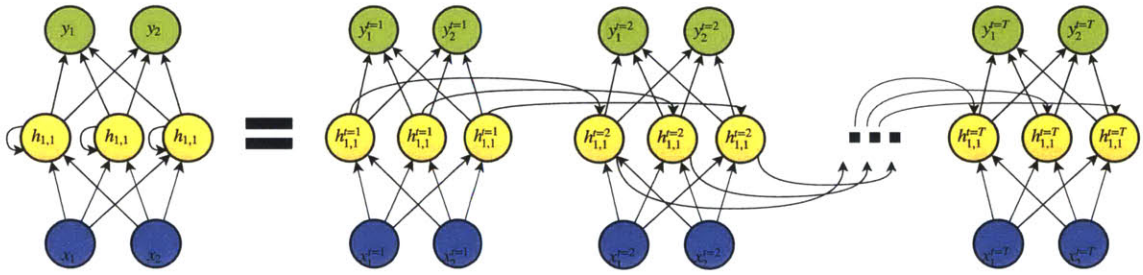


Figure 2-3: Example recurrent neural network. The input to the network is a vector sequence. The output of the network is also a vector sequence. Both input and output vectors are two numbers each. For every step of a sequence we use a vector of three numbers to represent the hidden state (memory). Crucially the weights (represented by arrows) stay the same between the time steps.

To formally define an RNN, let's consider a sequence of input vectors $x^{t=1}, ..., x^{t=T}$ and a sequence of output vectors $y^{t=1}, ..., y^{t=T}$. A single time step i of computation is defined by:

$$
\begin{aligned}
h_1^{t=i} &= \tau_1(W_1 x^{t=i} + V_1 h_1^{t=(i-1)} + b_1) \\
h_2^{t=i} &= \tau_2(W_2 h_1^{t=i} + V_2 h_2^{t=(i-1)} + b_2) \\
&\dots \\
y^{t=i} &= \tau_n(W_n h_{n-1}^{t=i} + V_n y^{t=(i-1)} + b_n)
\end{aligned}
$$

The main difference between this set of equations and equations for MLP is the addition of a set of linear transformations $V_1, ..., V_n$. This means that $h_j^{t=i}$ depends on $h_j^{t=(i-1)}$, from the previous time step, which results in $y^{t=i}$ being (indirectly) dependent on $x^{t=i}, x^{t=(i-1)}, ..., x^{t=1}$.

Long Short-Term Memory Network

As we will see in the remainder of this thesis RNNs are great tools suitable for variety of tasks. However, they do have one serious flaw: When trained over sequences longer than a few items they suffer from vanishing and exploding gradient problems. That is gradient values are all either 0 or ∞ for early time steps [17]. This was initially a major obstacle preventing successfully applying RNNs for real world problems. In 1997 the first solution to that problem was presented in the form of Long Short-Term Memory Network (LSTM) [18]. It is quite interesting to point out that the solution was not really algorithmic, but architectural - by manipulating the set of equations defining the network's architecture the authors of the paper managed to ensure that there's always a direct (constant number of nonlinearities) path from the networks output to any input, which alleviates the vanishing/exploding gradients problem.

Nonstandard Architectures

There are many interesting attempts at creating models inspired by above ideas, but more suitable for specific problems. Examples include Neural Turning Machine [12], Memory Networks [54], Hierarchical LSTMs [36], Tree-LSTMs [42] and many more.

2.2.3 Backpropagation and the Solver

Once we have defined our error function $E(D, A(\theta))$, given data D and architecture A parametrized by θ, we can use the backpropagation algorithm [24] [51] to compute derivative of E w.r.t θ (leveraging the fact that A is differentiable). Then we can use that derivative to find a local minimum of E using Stochastic Gradient Descent [9]:

$$\theta' = \theta - \alpha \frac{\partial E}{\partial \theta} \tag{2.1}$$

Where α is a problem dependent parameter. Equation 2.1 is repeatedly applied until convergence, or until performance on validation set stops improving.

When deciding how to apply a non-convex solver to a particular problem, it is important to understand that that optimizing the performance on a particular set of examples does not guarantee similar results on unseen examples. Because of that when designing experiments, it may be beneficial to consider the following modifications to the simple scheme described above.

- Rather than training error function on the entire dataset at once, we train it on a small batches of examples from the dataset to improve generalization and computational performance.

- Certain non-differential architectures can still be successfully optimized - for example, it has been demonstrated neural networks augmented with Dropout [39] are less likely to *overfit*.

Moreover, SGD is only guaranteed to find a local minimum, when the learning rate is low enough. However, low learning rates are undesirable, because they increase the total number of iterations before convergence. Fortunately it is sometimes possible to improve one without sacrificing the other by changing the update equations (replacing eq. 2.1), e.g. momentum methods, RMSprop, Adadelta [55] etc.

2.2.4 The Error Function

Appropriate choice of error function can drastically affect the properties of model that we learn. Let's consider for example loss function for multi-class classification problem with n classes $c_1, ..., c_n$. Assume a scenario where we want to compute an error for single example where correct class is c^* and model assigned scores $s_1, ..., s_n$ to $c_1, ..., c_n$ respectively (if s_i's are positive and add up to 1, they can be thought of as probabilities). If there are multiple examples, we sum the error on every one of them. Let's define a posteriori distribution over different classes, for which we know the correct label p^* as:

$$p_i^* = \begin{cases} 1 & if\ i = i^* \\ 0 & otherwise \end{cases} \tag{2.2}$$

Where i^* is such that $c* = c_i$ Below we present three alternative formulations of error function:

(a) **Squared loss**

The error is simply

$$\sum_{i=1}^{n} (s_i - p_i^*)^2 \tag{2.3}$$

(b) **Cross-entropy loss** tries to minimize the distance between the score distribution predicted by the model and p^*.

$$-\sum_{i=1}^{n} p_i^* \log s_i \tag{2.4}$$

It is therefore required that s_i's satisfy the constraints of a discrete probability distribution:

$$\sum_{i=1}^{n} s_i = 1 \qquad\qquad \forall_i . s_i \geq 0$$

Cross-entropy loss has a beautiful mathematical interpretation - it is the minimum number of bits required to compress the output, given the model and the input.

This bound is tight and can be achieved by Arithmetic Coding [28], which means that given a prediction model, there exists an algorithm that can make optimal use of the model for compression. Hutter prize [2] is a competition with $50000 prize pool for compressing the first 100MB of Wikipedia, argumenting that *"Being able to compress well is closely related to intelligence [...] The intention of this prize is to encourage development of intelligent compressors/programs as a path to Artificial General Intelligence."*

(c) **Margin ranking loss** [54] is perhaps the simplest and quite surprisingly the most recent of the three loss functions presented here. This is perhaps due to the fact that it contains a non-differentiable point around 0. For a given value of margin γ (typically around 0.1) it is defined as

$$\sum_{i=1}^{n} max(s_{i^*} - s_i + \gamma, 0) \tag{2.5}$$

It has a desirable property that target class score s_{i^*} is bigger than score for some other class s_i by at least γ, then the derivative w.r.t both of those scores is zero.

Sometimes it helps to simultaneously optimize multiple objectives. For example in case of Dynamic Memory Networks [23] and Hierarchical LSTM [36] it is critical to simultaneously optimize prediction performance and fact selection performance. Furthermore auxiliary functions like Occam's Gates [36] can improve model generalization and interpretability.

2.3 Word Vectors - The Core Idea behind Modern NLP.

In his ACL 2015 keynote Chris Manning [3] called word vectors the most enabling invention in past decade of Natural Language Processing research. To understand

[2] http://prize.hutter1.net/

[3] Professor from Stanford specializing in Natural Language Processing

what word vectors are, one must answer the following question: what is a statistically sound way of representing a single word? The simplest idea is one-hot encoding - for n words that are known, we represent the i-th word by a vector $v \in \mathbb{R}^n$ such that $v_i = 1$ and $\forall_{j \neq i}.v_j = 0$. Unfortunately, this type of representation has three main disadvantages:

1. it is very high dimensional and therefore computing over it can be inefficient.
2. no meaningful relations between the words can be inferred from it.
3. it is hard to extend it with new words (and vocabularies evolve all the time - think of words like *google*, *tweet*).

Word vectors are designed to address those concerns. The main idea is very simple. Assume a representation exists, where we have vectors representing words in dictionary. We will denote the vectors for words $w_1, ..., w_n$ by $v_1, ..., v_n \in \mathbb{R}^d$ for some choice of d. The vectors are such that v_i encodes the semantics of word w_i. Under that assumption, we could use those them to solve NLP tasks, like estimating probability that two words will co-occur in the same passage of text, or classifying sentiment of a sentence. We could formalize this condition by defining it as exceeding some performance score P on a particular dataset D for a given problem. One can think of it as a necessary condition for word vectors. But is it also sufficient, i.e. if we satisfy it do we obtain a word vector representation, that carries meaning? This seems to be the case; even though we don't have a complete formal theory yet, all the experiments seem to support that statement. Armed with that piece of knowledge we can provide a simple algorithm for finding word vectors - take a machine learning model that takes word vectors as its input for solving a particular NLP problem (co-occurrence, sentiment etc.). During optimization word vectors are treated as model parameters - after the model converges one can hope that the optimal set of parameters yields meaningful word vectors.

Such an experiment has been carried out independently by many researchers. All the resulting representations satisfy three properties mentioned:

1. It is fairly low dimensional. Glove [34] vectors are between 50 and 300. Word2vec [29] is of size between 50 and 600. Additionally, for most tasks where word vectors are used the classification performance improves with dimensionality, which makes for desirable computational efficiency versus performance trade-off.

2. Most word vector representations have very useful properties. Here are some examples of properties satisfied by Glove [34] vectors:

   - Euclidean distance is small for semantically similar objects and large for different objects. For example for Glove the four closest vectors to v_{frog} are $v_{toad}, v_{litoria}, v_{leptodactylidae}, v_{rana}$ (where the last two words are species of frog).
   - Similar type of analogies are represented by vectors close to each other:

$$v_{king} - v_{queen} \approx v_{man} - v_{woman}$$

$$v_{MIT} - v_{Massachusetts} \approx v_{Stanford} - v_{California}$$

   Those properties have far reaching consequences - for example it has been shown that we can use a simple MLP to map between brain activity of person thinking about a word and v_{word} [50]

3. Adding new words is easy - one just needs to find a way to project them on the linear space in meaningful way. This can be done, for example, by retraining the word vectors with extended vocabulary, while keeping the previously trained vectors constant. In fact we can go one step further. It has been shown that for two sets of word embeddings trained independently for two different languages, we can project one onto the other with a simple linear projection, while preserving most of the semantic relationships across the languages [10].

Word vectors are strongly related to diffusion maps [13]. One can imagine that words normally reside in some high dimensional manifold, which we try to unfold onto linear

space. It turns out that even though diffusion maps were not an inspiration for word vectors, the idea behind word vector computation can be generalized to a very efficient algorithm for computing diffusion maps [13].

2.4 Case study: Translation

It has been shown that LSTM networks can be used to translate sentences with performance comparable to or better than state of the art approaches based on engineered features [41].

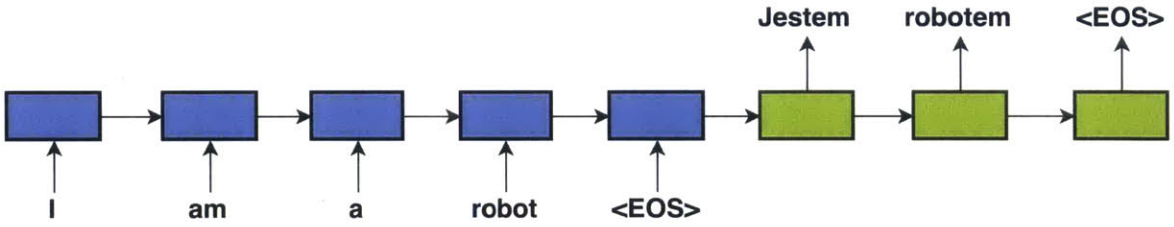


Figure 2-4: Depiction of architecture used for Sequence-to-Sequence machine translation.

The data There are multiple machine translation datasets, but perhaps the most cleverly obtained one is the opensubtitles dataset, which uses the publicly available movie subtitles, and aligns sentences in different languages [44]. Taking English-French translation as an example, today there exists about 1 000 000 000 tokens of translation data in all the revisions of opensubtitles corpus.

The architecture Figure 2-4 is a high-level depiction of network's architecture. The blue network is a Multi-Layer LSTM and it serves as an encoder network. The green network is of the same architecture and dimensions as the blue network, but uses different set of parameters. Its purpose is to decode hidden representation obtained by the blue network, to produce a translation. The `<EOS>` symbol indicates ends of sequence, which allows extra computation to be performed after having read entire sentence. The inputs to the encoder network are English words which are encoded

using word vectors. Decoder network outputs a probability distribution over all the words in the dictionary. This is done by projecting the hidden state to a vector of size equal to the number of words in the dictionary. Such vector is subsequently normalized to form a probability distribution using softmax function:

$$\text{softmax}(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{dim(x)} e^{x_j}} \tag{2.6}$$

Due to the fact that projection matrix between hidden state and vocabulary can be quite big ($1000 \times 80\,000$ in the original paper [41]), researchers have come up with more sophisticated ways of modeling probability distributions over words, like hierarchical softmax [32]. Interestingly the original paper isolated the projection operation and distributed it over 4 Graphical Processing Units (GPU), in addition to other 4 GPUs optimizing the rest of the network.

The error function To compute the error, the network is evaluated for input and output sentences (including the `<EOS>`). For every output word distribution we apply cross entropy error function between the output distribution and the target word (including `<EOS>`). The sum of errors for every word is the total error for that example.

Note that we can evaluate error because during training time we know exactly for how many steps to run the decoder network. When using a trained model we don't know the resulting sentence length and we therefore run the network until the most probable output token is `EOS`.

2.5 Case study: Dialogue Management

The term Natural Language Dialogue is used to refer to a broad class of problems, some of which we can efficiently solve using Non-Convex Optimization. Perhaps the most impressive work to date is the Neural Conversational Model [46], where the model learns, based on real world data, to answer open-ended questions, like "What

is the usual color of a leaf?", or "What is morality?". In this section, we chose to describe a model, capable of answering questions based on context information. The architecture for this problem is a good example of a complex NLP architecture.

The data Facebook AI Research recently proposed a set of 20 tasks designed to be "prerequisites" for any system "capable of conversing with human" [52]. The dataset for each task is a set of stories, each composed of *many facts*, some of which are marked as *relevant*, a *question* and the correct *answer*.

1 Daniel and Sandra journeyed to the office. 2 Then they went to the garden. 3 Sandra and John traveled to the kitchen. 4 After that they moved to the hallway. Where is Daniel? **A: garden** Relevant facts: 1, 2	1 The football fits in the suitcase. 2 The suitcase fits in the cupboard. 3 The box of chocolates is smaller than the football. Will the box of chocolates fit in the suitcase? **A:yes** Relevant facts: 3, 1

The tasks are synthetic and lack complex nature of the real-world Natural Language, which makes them easy to solve with hand-engineered systems. However, the challenge proposed in [52], is to create a general model capable of solving all of these tasks, without any manual, problem-specific feature engineering.

The architecture The architecture is presented in figure 2-5. There are four LSTM networks used in that model:

1. **Question Reader Network**. Colored in Red. It is used to read the question and compute its hidden representation (which is hidden state at the last time step).

2. **Fact Reader Network**. Colored in Blue. It is used to precess facts from the context story and compute their hidden representation. It is executed for every fact independently.

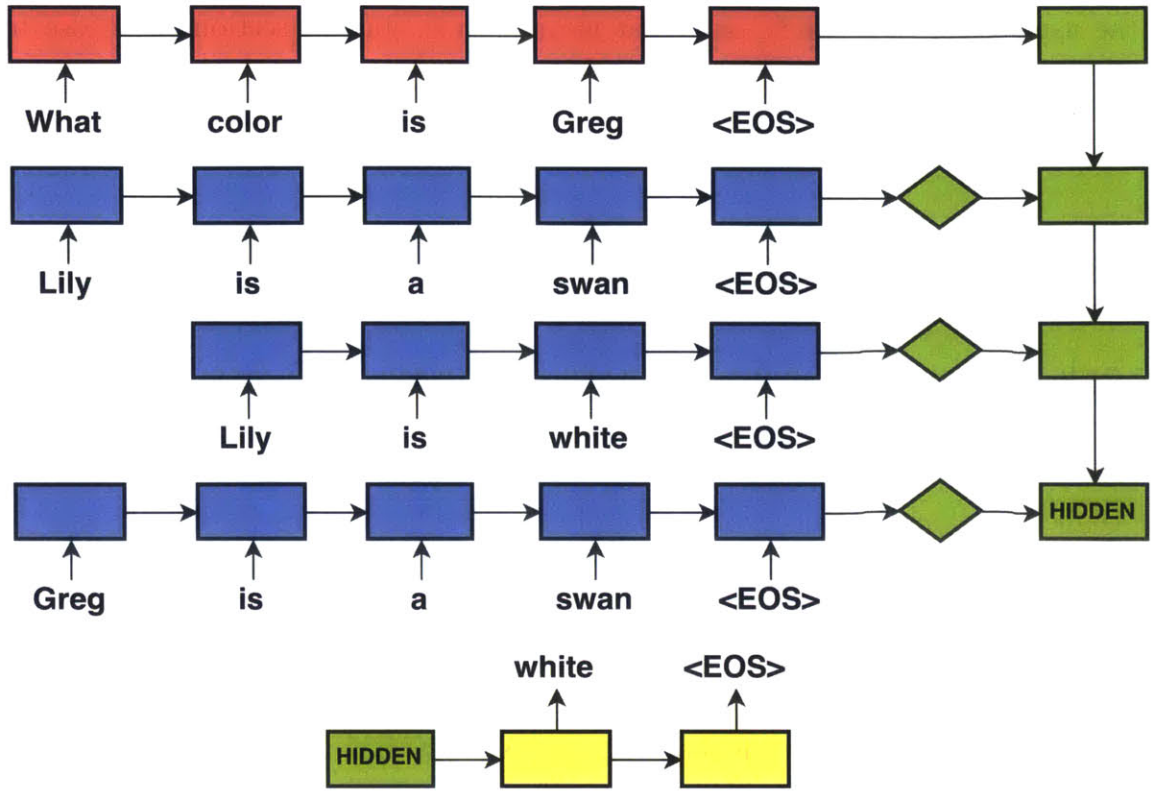


Figure 2-5: Depiction of the hierarchical LSTM model for question answering with context and deduction.

3. **High-level Network**. Colored in Green. It takes question hidden as input in the first step and in subsequent steps it takes fact hiddens. The last node denoted by `HIDDEN`, encapsulates all the information from question, context and deductive reasoning that was performed. The rhombi denote gates. They are simple linear functions of high-level network hidden at a time step and a hidden representation of corresponding fact. That linear function is fed through sigmoid nonlinearity to obtain a single number g_i in range $[0, 1]$ - the degree to which a given fact is considered relevant. The fact hidden is multiplied by g_i before being used by input to a network. In some cases that leads to improved performance, as network learns to compute over only the relevant facts. This idea is explored further in [36].

4. **Decoder Network**. Colored in Yellow. It takes `HIDDEN` as initial hidden and

decodes it into output sequence of words.

The error function The error function is regular cross entropy error on predicted sequence (just like in the case of translation). However, this time we have auxiliary objective to selecting correct values of g_i, which we can be included in our error function, because the training data explicitly includes information about which facts are relevant. The actual error function is therefore weighted sum of the two objectives - fact selection error and prediction error.

This model is a great example of how inductive bias encoded in network's architecture can make a huge difference. When we solve this problem by naively applying LSTM (using context and the question as one long sequence read by the network), the model answers questions with average 50% accuracy. In contrast model based on idea described above achieves 94% accuracy [23].

2.6 Bridging Natural Language and Reinforcement Learning

In order to apply ideas learned from this chapter to Reinforcement Learning, we use statistical signal obtained in the distributed representation. To illustrate, let's consider the Skip-Thought Vectors model [21]. It has been trained to read a big corpus of 8290 books [56], and based on each sentence in isolation, it predicts two sentences: one before and one after. The intermediate hidden vector that is the output of the reader network and input to prediction networks constitutes a good semantic representation of a sentence read. For example, consider the problem of designing Siri-like automated assistant for a smart house. Figure 2-6 presents example queries and distance-preserwing projection of their hidden vectors onto 2D plane. Even though skip-thoughts were not designed for this type of problem, they provide a good separation already.

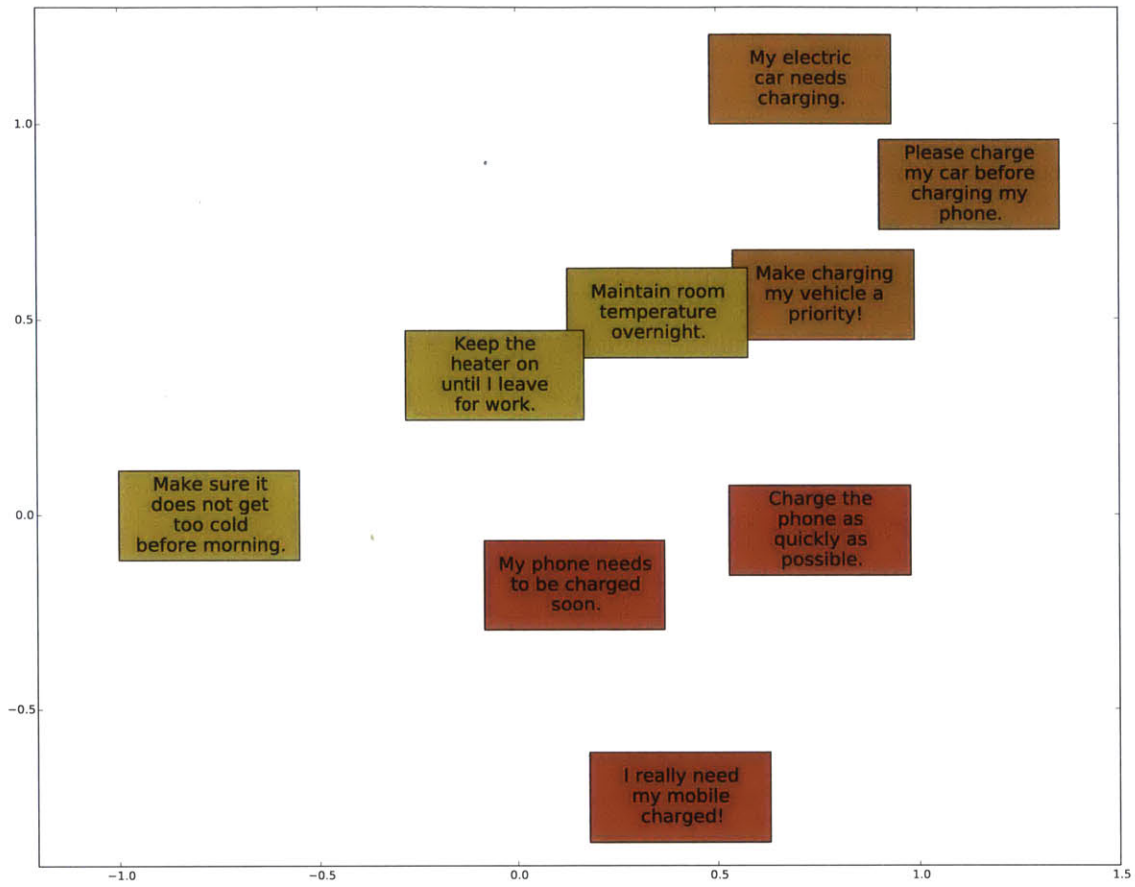


Figure 2-6: Skip thought vectors for various Natural Language queries which could be used as state for a problem of scheduling activities in smart house. The original dimension was 4800, which was projected onto 2D plane using using Spectral Embedding with a Gaussian kernel, which is known to preserve cluster pairwise distances.

2.7 Conclusion

In this chapter we described Non-Convex Optimization techniques, including relevant architectures, error functions and algorithms. We discussed in detail Word Vectors, their properties and ways of computing them. To ground the theory in examples we presented two use cases - Translation and Dialogue Management. We concluded the chapter with an experiment that demonstrates how Sequence-to-Sequence models are capable of learning semantically meaningful representations, which are of interest in the Reinforcement Learning applications.

Chapter 3

Reinforcement Learning as a Non-convex Optimization Problem

In this chapter we provide an overview of Reinforcement Learning theory. First, we give the formal definition of a Markov Decision Process, as well as the simulator setting which is a formulation better aligned with real-world problems. Next, we state the Bellman Equation which is the fundamental building block of the majority of Reinforcement Learning approaches. Continuing on, we describe a few major approaches focusing on their relation to Bellman Equation, as well as the issues addressed by each of them. In particular, we present in detail the Deep Q-learning algorithm, as it is the main component of our approach for Reinforcement Learning with Natural Language as part of the state. We conclude the chapter with a demonstration of performance on marble collection game, which allows one to better understand the effect of changes to the discount rate and introduces the a reinforcement learning problem very similar to the one used in the manipulation experiment in section 4.2, but without Natural Language as part of the state.

3.1 Introduction

Control of autonomous agents, in particular robotic agents, is still largely an unsolved problem (fig. 3-1). Thanks to the advancements in Control Theory, Trajectory Plan-

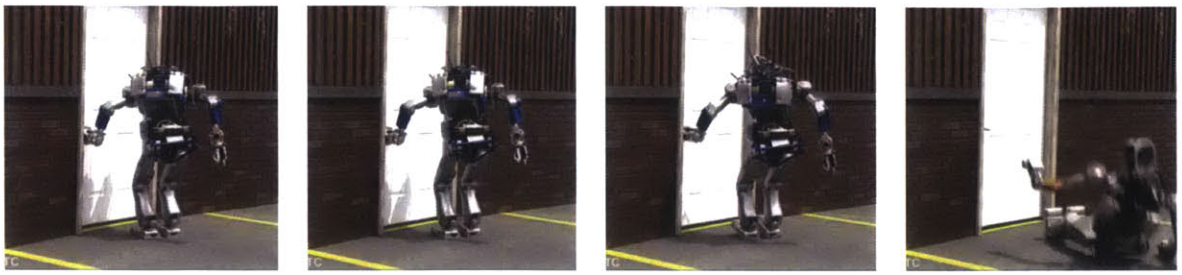


Figure 3-1: A robot participant in Darpa Robotics Challenge 2015 attempting to perform door opening sequence.

ning and Integer Optimization, we are now able to optimize trajectories and stabilize them [31]. Those methods allowed for some impressive results, such as landing robotic bird on a perch [31], and high speed (9 m/s) control of UAVs in cluttered environments [2]. However, they do have two significant disadvantages:

1. **A model of state transition is required** - in order to optimize a trajectory we need to understand how the actions affect future states precisely (in the case of robotics this information is broadly referred to as dynamics). Obtaining such models can be a very time-consuming process - a lot of person hours for human designed model or large number of examples for a learned model. This is especially relevant when building a model that concerns not only the agent, but also the environment.

2. **Low degree of adaptation to unforeseen scenarios** - any significant deviation from planned trajectory can result in control policy being unable to recover. Although it has been successfully demonstrated that by constructing high level models and leveraging execution monitoring techniques some failure cases can be detected and recovered from, it comes at the cost of greater model complexity [25].

A promising direction in addressing those concerns is Reinforcement Learning (RL). Instead of using a model, the agent has access to the environment, where it is free to interact and learn. The learning process is guided by the reward signal, which increases when agent performs well. The reward signal can be as simple as distance

to goal location, or a number of collected objects. One must understand the difference between RL and optimization over a learned model. In the former case we learn the model implicitly and jointly with optimizing the control policy. In the latter case the algorithms can be smart about which parts of the model to learn and which to ignore, such that the discovered control policy is the best possible, given the representational capacity of the model. Due to high complexity of learning task at the core of RL, it used to be impractical to use this approach for real world applications. Recently there has been a renaissance in RL related to advances in non-convex optimization. In particular, in the paper published by DeepMind, an algorithm (Deep Q-learning) was proposed that leverages Non-Convex Optimization as Q-value approximator [30] in Reinforcement Learning. They managed to teach a model to play Atari games based solely on the image of the screen and game score as a reward signal. This approach is compelling because the Q-value approximator was not an ordinary MLP, but a Convolutional Neural Network. This is a piece of evidence that the Deep Q-learning algorithm can leverage many different types of Non-Convex models. This key insight is the base of our approach for incorporating Natural Language as part of the state in Reinforcement Learning.

3.2 Problem statement

Markov Decision Process (MDP) is specified by the following items:

- set of states S
- set of actions A
- transition function $T : S \times A \rightarrow S$ (for simplicity we assume deterministic transitions)
- instantaneous reward function $r : S \times A \rightarrow \mathbb{R}$

Given a MDP, the challenge is to come up with a policy $\pi : S \rightarrow A$. Given a starting state $s_0 \in S$ a policy induces a trajectory of states $s_t = \pi(s_{t-1})$, actions $a_t = \pi(s_t)$

and rewards $r_t = r(s_t, a_t)$. The value of a trajectory is

$$V_\pi(s_0) = \sum_{t=0}^{\infty} \gamma^t r_t \tag{3.1}$$

Here $0 \leq \gamma < 1$ is a discounting factor which is chosen for a particular problem. The higher γ is, the more future rewards influence the value of a trajectory. In particular $\gamma = 0$ means that greedy strategy is optimal - one has to always pick action with highest instantaneous reward. We say that a policy is optimal if for any given starting state it induces a trajectory of maximum possible value, we denote any policy with that property π^*.

In this chapter we will consider MDPs where we don't know T and r explicitly, but instead we have access to simulator $\Delta = (S, A, s, \delta)$ where S is the state space, A is the action space, $s \in S$ is a variable representing the current state and δ is simulator advancement procedure. It takes action $a \in A$ as a parameter and returns the reward and the next state $\delta(a) = (r(s,a), T(s,a))$. Furthermore, it has a side effect of updating the current state, so that s becomes $T(s,a)$, which makes this model more restrictive than general MDP, because we cannot find out what reward we would have gotten if we chosen different action when at state s (unless not until the simulator revisits that state). Nevertheless, this formulation is more realistic, especially if one does not have accurate computer simulation of outside environment, but rather wishes to use a real-world environment, which, to the best of the author's knowledge, does not support traveling back in time.

3.3 Reinforcement learning theory and algorithms

3.3.1 Bellman equation

Bellman equation was proposed by Richard Bellman as part of his work on Dynamic Programming. Almost any RL algorithm depends on some formulation of that equation. In the simplest form it states that the following is a necessary condition for π^*

to be optimal:

$$\forall_{s \in S} V_{\pi^*}(s) = \max_{a \in A}[r(s,a) + \gamma V_{\pi^*}(T(s,a))] \tag{3.2}$$

Notice that the condition is not straightforward to verify. Firstly computing V_{π^*} requires evaluating infinite sum. Secondly the state space S can be huge or infinite (e.g. all the possible configurations of joints of a robot). Fortunately, in the case of the first problem the solution is simple and has been successfully used for over 60 years - Dynamic Programming. Solving the second problem is one of the main topics in this chapter.

3.3.2 Curse of Dimensionality

The famous term *Curse of Dimensionality* was coined by Richard Bellman when considering Reinforcement Learning and other Dynamic Programing problems. Many RL algorithms require construction of table of size $|S|$ or even $|S| \times |A|$. For example, if the state space is a d dimensional binary vector, that means one would need to create a table of size 2^d to represent all the possible trajectories. For example if an input is 17×17 black and white photograph then the number of states that we need to memorize exceeds the estimated number of atoms in the observable universe.

3.3.3 Value Iteration

In Value Iteration we try to find V explicitly by applying the following update function repeatedly for all the states $s \in V$ until convergence:

$$V(s) = \max_{a \in A} r(s,a) + V(T(s,a)) \tag{3.3}$$

Optimal policy (described in eq. 3.2) is a fixed point of the update equation. The disadvantage of this method is that ones needs to know the transition function T in order to pick the optimal action during execution, which means that it does not work in the simulator setting.

3.3.4 SARSA

In SARSA we try to estimate the action-value function (denoted by Q) which is expected to ultimately (at convergence) represent the utility of choosing action a while at state s, i.e. if we denote Q at convergence by Q^* we would like it to satisfy:

$$Q^*(s,a) = R(s,a) + \gamma V_{\pi^*}(T(s,a)) \tag{3.4}$$

which can be also written as

$$Q^*(s_t,a_t) = r_t + \gamma \max_{a' \in A} Q^*(s_{t+1},a') \tag{3.5}$$

The name of the SARSA algorithm comes from the data used for every update to the Q function which is $s_t, a_t, r_t, s_{t+1}, a_{t+1}$:

$$Q(s_t,a_t) = Q(s_t,a_t) + \alpha\left(r_t + \gamma Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)\right) \tag{3.6}$$

Where α is a parameter used to control the learning. One way to think about SARSA is that used to slowly bring $Q(s_t,a_t)$ closer to satisfying the Bellman equation. Notice that at convergence we have

$$Q^*(s_t,a_t) = r_t + \gamma Q^*(s_{t+1},a_{t+1}) \tag{3.7}$$

which is a fixed point of equation 3.6, but only if $\pi^*(s_{t+1}) = a_{t+1}$, which may not be the case.

3.3.5 Q-learning

Q learning is very similar to SARSA with a key difference that rather than considering a_{t+1} we consider the action that maximizes the estimate of Q value at s_{t+1}:

$$Q(s_t,a_t) = Q(s_t,a_t) + \alpha\left(r_t + \gamma \max_{a' \in A} Q(s_{t+1},a') - Q(s_t,a_t)\right) \tag{3.8}$$

The advantage of this approach is optimal Q function (eq. 3.5) is unconditionally a fixed point of the update equation. The disadvantage is that it is biased towards more "optimistic" solutions (higher Q values than the solution) as evidenced in [14].

3.3.6 Temporal Difference Learning[43]

All of the methods described above suffer form the Curse of Dimensionality. One of the early successes at addressing this issue was the work on TD-Gammon, which is an RL based agent playing Backgammon. To address the issue of large state space S, instead of representing the value function directly, the authors of [43] used a Multi-Layer Perceptron to learn a compressed representation of V. The original value iteration update rule (eq. 3.3) was found to produce too large gradient values and instead a custom update rule was introduced.

3.3.7 Deep Q-learning [30]

The Deep Q-learning can be thought of as a successor of TD learning algorithm. It addresses the limitations of value iteration algorithm and further improves learning stability. It was successfully employed in [30] to learn to play Atari games based solely on the image of the screen. The algorithm uses Q learning algorithm at its heart. Deep Q-learning assumes that state can be represented as a vector of real numbers $S = \mathbb{R}^n$ and the set of possible actions is finite $A = \{A_1, ..., A_k\}$. The Q function is a Multi-Layer Perceptron, where input vector is a state from S and the output vector $O = (Q(s, A_1), ..., Q(s, A_k))$. Instead of using the Q Learning update equation (eq. 3.8), it is implicitly implemented as a Stochastic Gradient Descent (SGD) optimization in the following way: Given a transition (s_t, a_t, r_t, s_{t+1}), let the target value be $y_t = r_t + \gamma \max_{a' \in A} Q(s_{t+1}, a')$. Now we define an error function $E_t = (Q(s_t, a_t) - y_t)^2$ which can be optimized with SGD. When using vanilla SGD this is equivalent to Q learning update rule (with α being the learning rate), but using this formulation we can employ more sophisticated optimizers (e.g. [30] uses RMSprop). To increase the convergence speed the following ideas were introduced:

1. Unlike TD-Gammon we do not update the network based on subsequent transitions (s_t, a_t, r_t, s_{t+1}) , but each time we independently sample a random subset of transitions from the past. This means that the subsequent updates to the network are uncorrelated, which in theory is required for SGD correctness.

2. Some gradient updates would cause the network weights to be close to infinite, which is a state from which we cannot expect to recover when using IEEE floating point numbers. Therefore all the gradient updates with gradient norm higher than some constant value were discarded.

3. Due to the fact that the network can get stuck in a strategy that corresponds to a local minima (e.g. always move to the left, hide in the corner, etc.), we only learn based on a fixed number of recent transitions. This means that a_t's are more in line with current policy.

4. When computing y_t we use a exponentially averaged version of Q network (denote it Q^D). Each time Q is updated we set $Q^D = \tau Q + (1 - \tau)Q^D$. This decreases the oscillation of the error function and leads to more stable learning.

3.4 Experiments

3.4.1 Experimental setup

To better understand the Deep Q-learning algorithm, a simple experiment was implemented, which was inspired by work done by Andrej Karpathy [1]. The experiment is set up in the following way: There is a board with red and green marbles and a hero. The goal of the game is to collect green marbles while avoiding collecting red. Every time a marble is collected a new one is spawn of exactly the same color. Marbles are subject to rules of physics - they travel with constant velocity and they bounce of walls (the collision is perfectly elastic, so the speed remains constant). The hero is subject to the same rules, but we are able to control its acceleration. Figure 3-2

[1] `http://cs.stanford.edu/people/karpathy/reinforcejs/index.html`

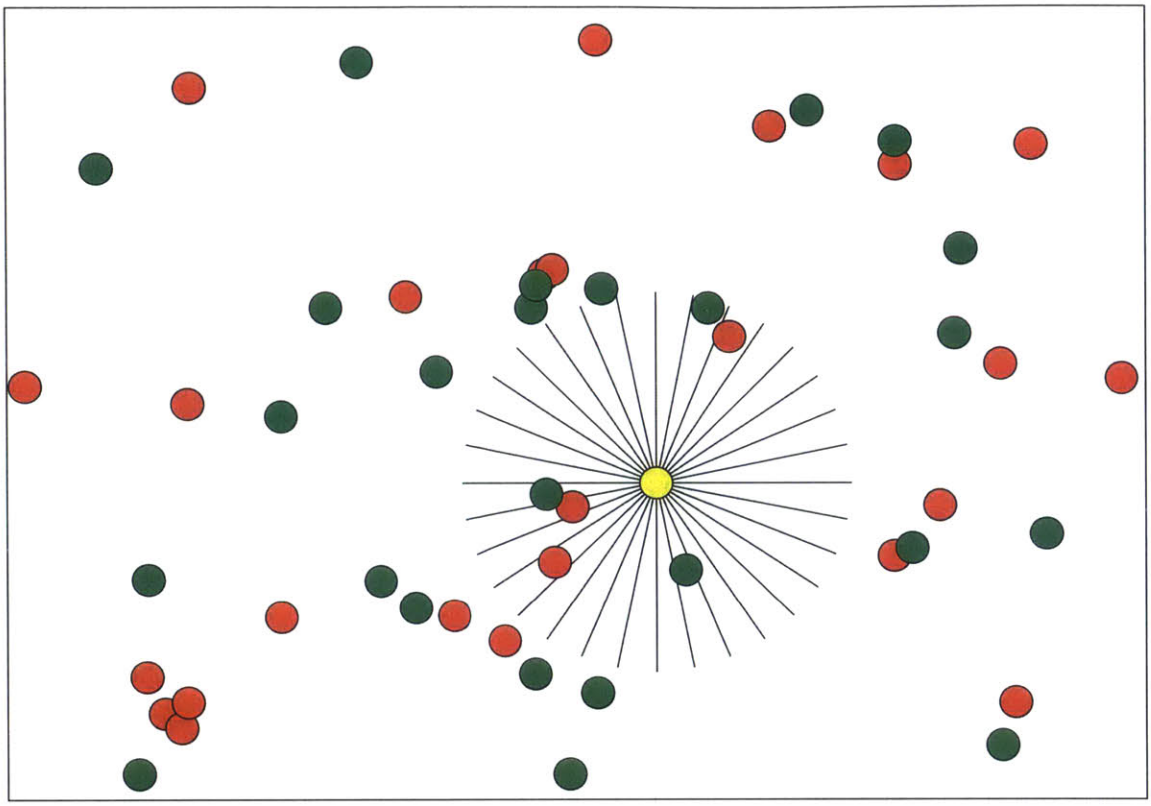
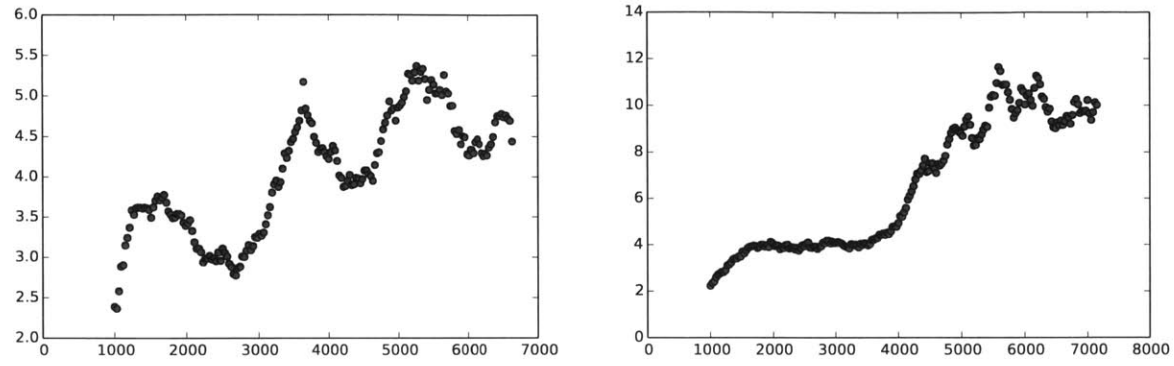


Figure 3-2: Example state of a board in the game. The goal is to collect green marbles while avoiding red. Yellow marble depicts the hero. The black lines are the visibility lines described in section 3.4.1.

shows example game state. We wish to formulate this experiment as a MDP and we therefore need to define state space, action space, transitions and rewards.

Starting at hero's position there are 32 visibility lines (VL) extending outwards. Each VL can be thought of as an "eye" of the hero - it can see 0 or 1 objects. Each VL induces 5 numbers in the state. First there numbers are distances to the interection with the green marble, red marble and a wall normalized to 1.0. If no such object intersects a visibility line then the corresponding number is one. If there are multiple objects intersecting the visibility line, only the closest distance is represented and the rest is set to 1.0. The remaining two numbers represent the velocity vector of the object intersecting a line (if that object is a wall or there is no object, then those numbers are set to 0.0). For example if a particular red marble intersects a VL in



(a) greedy ($\gamma = 0$)

(b) long-term ($\gamma = 0.99$)

Figure 3-3: Performance of two different reinforcement learning agents on the marble collection game. The number plotted on the graph is the ratio of green to red marbles collected in the past 1000 seconds. The x-axis is time in seconds. Note that scales on y-axes are not the same.

the middle of its length, and that marble travels with speed 0.7 to the right then the state induced is $[1.0, 0.5, 1.0, 0.7, 0.0]$. Aside from state induced by VLs, hero's speed is also part of the state. Total length of the state vector is therefore $32 * 5 + 2 = 162$. The state transitions are dictated by the laws of physics, and to make a valid MDP formulation we define $dt = 0.1s$ as the amount of time that passes between subsequent states.

There are for available actions - UP, DOWN, LEFT, RIGHT, and each corresponding to constant change of velocity in a given direction. The reward function is 0.1 for every green and -0.1 for every red marble collected since last state transition. Notice that the reward is often zero.

3.4.2 Results

Agent	long-term ($\gamma = 0.99$)	greedy($\gamma = 0$)	hand-coded
original	10.23	4.4	6.01
2x more marbles	5.76	3.9	5.4
hero does not bounce	7.7	3.74	5.97
2x shorter visibility	6.5	2.2	3.1

Table 3.1: Results demonstrating how well different strategies generalize when the game setting is slightly modified.

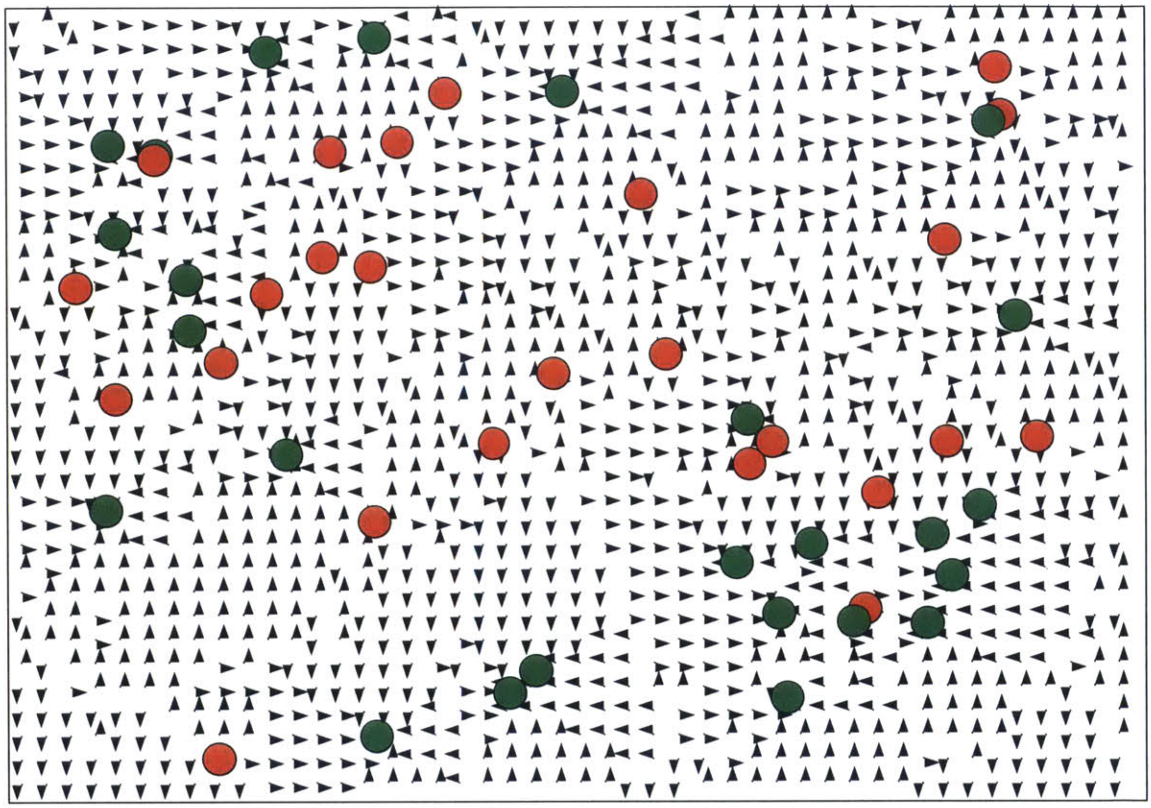


Figure 3-4: Illustration of the strategy learned by greedy ($\gamma = 0$) agent. Arrows depict preferred action for any given position.

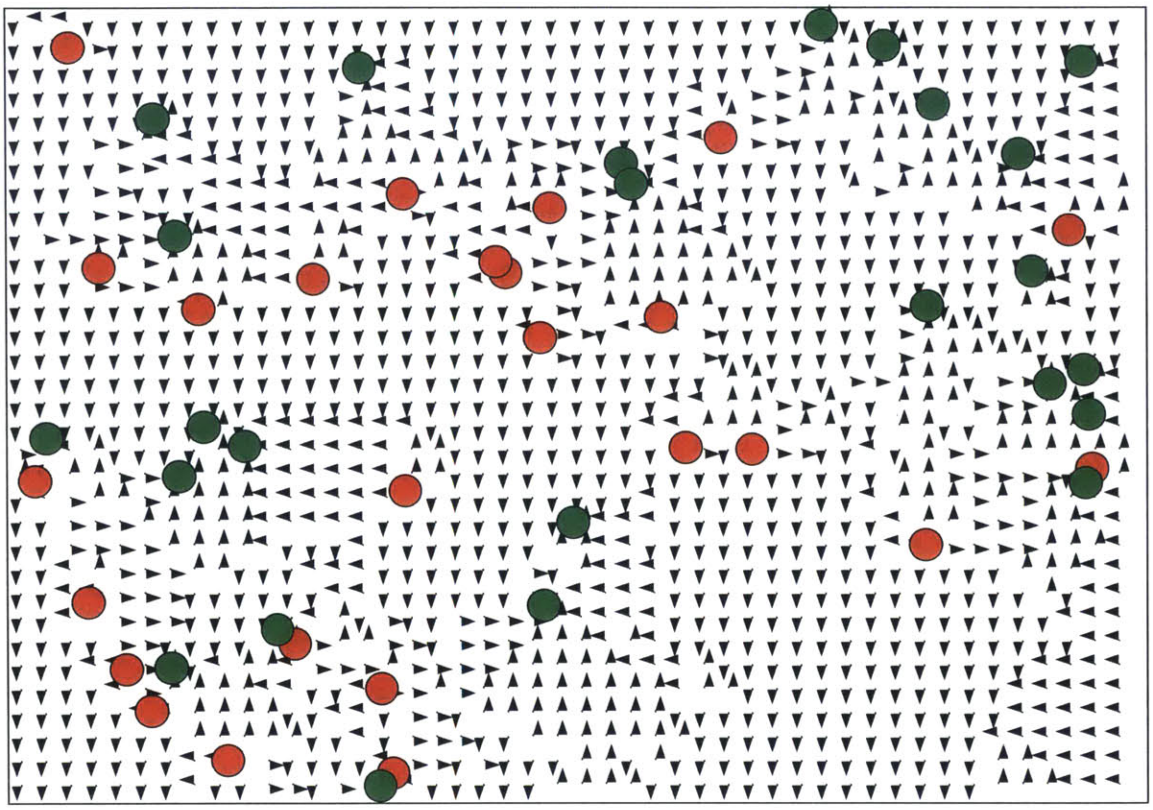


Figure 3-5: Illustration of the strategy learned by long-term rewards ($\gamma = 0.99$) agent. Arrows depict preferred action for any given position.

Three different agents were implemented for the scenario described above: two Deep Q-learning controllers with $\gamma = 0.0$ (greedy) and $\gamma = 0.99$ (long-term reward) and one hand-coded controller which always goes in the direction where the ratio of visible green/red marbles is the highest. The Deep-Q learning agents used a Multi-Layer Perceptron with two hidden layers of size 200 and tanh activation function. The gradient optimization was performed using RMSProp with learning rate 0.0001 and momentum 0.9.

The results are summarized in the top row of table 3.1. The long-term agent outperformed greedy and hand-coded. Considering the fact that greedy algorithm only learns when it is very close to the consumed marble it might seem surprising that the learned strategy is so good. Nevertheless once the network learns to go towards green balls and away from red balls at close distances, due to the way the state vector is created, one can expect similar activation to arise when the marbles are further away. Figure 3-3 shows the performance of Deep Q-learning agents over time. Both of the agents seem to achieve their peak performance around the same time, but the performance of long-term agent is over twice as good and learning curve is less oscillatory. Figures 3-4 and 3-5 depict the strategy learned by the greedy and long-term agent respectively. One can see that greedy agent sometimes makes nonsensical decisions, like going away from a green marble, especially at a slight distance.

Finally to test how well the strategy learned in the original game setting generalizes to new situations three modifications were considered:

1. **2x more marbles** - instead of 25 green and 25 red marbles, the board had 50 green and 50 red marbles - with the same board size.

2. **hero does not bounce** - the hero collides with the wall perfectly inelastically - its speed is set to zero.

3. **2x shorter visibility** - the length of the visibility lines is cut by half.

We can see that all three agents generalize well to new scenarios, with the long-term agent generalizing the poorest. This is because the long term reward strategy might be more situation specific than greedy strategy.

3.5 Conclusion

In this chapter we provided an overview of Reinforcement Learning including definition of Markov Decision Process, Bellman Equation and various approaches to finding a good policy. We focused in particular on the Deep Q-learning algorithm and verified that it is capable of finding policies that better than the greedy strategies in the marble collection game. In addition we demonstrated that the algorithm learns a robust strategy which is capable of good performance even when we perturb some of the environment parameters. In chapter 4 we will explore a similar experiment, but rather than optimizing it for a specific type of task we will show that it can achieve good performance on many different tasks, when the objective of each task is provided as a Natural Language description.

Chapter 4

Reinforcement Learning Problems with Natural Language State

In this chapter we verify our claim that multi-step reasoning in Reinforcement Learning settings can lead to improvements on Natural Language Processing problems. The insights from previous chapters are combined to build a Reinforcement Learning model that can process Natural Language. We show that our model supports the claim by evaluating our approach on sentence unshuffling task and comparing its performance to a Sequence-to-Sequence model.

In the second experiment we showcase the flexibility of our approach by incorporating multi-modal input as part of the state. We teach an agent to perform simple manipulation tasks based on objective description in Natural Language. The input to the architecture consists of not only Natural Language, but also sensory data observed by the agent.

4.1 Natural Language Processing as Reinforcement Learning

This section introduces sentence unshuffling problem, and then evaluates two approaches: Sequence-to-Sequence RNN and Reinforcement Learning.

4.1.1 The Problem

The problem is defined in the following way: given a sentence with its words shuffled, predict the original sentence. For example, given the words "a Today interesting read John very book", the correct output is "Today John read a very interesting book". The unshuffling task is useful for high-level grammar correction, for example if user types "Today I read an book excellent.", we wish to determine that it is incorrect and suggest swapping the last two words. While we use unshuffling tasks as an illustration of our approach, it could be applied to many other NLP problems. Coreference resolution problem (given a piece of text, understanding which phrases refer to the same entity) seems particularly well suited - one could define action space as adding/removing a pair of words to/from the coreference clique. Furthermore, by augmenting the action space with deletions, substitutions and swaps, we could perform translation, or improve the quality of translations output by the Sequence-to-Sequence model.

4.1.2 Evaluation Criterion

The most straightforward evaluation criterion is the number of examples for which model predicts the original sentence correctly. However, notice that the correct output sentence might be ambiguous, for example the sentence "met Daniel Sandra", could be interpreted as both "Daniel met Sandra" and "Sandra met Daniel". Fortunately, we can alleviate this issue, while avoiding the need to model this ambiguity explicitly. We will incorrectly consider only one of those options correct (the one that appears in the training set), but rather than computing accuracy as the percentage of sentences correctly predicted, we will evaluate the percentage of words in the predicted sentence that appear at the correct positions. For example predicting "Sandra met Daniel", while the original sentence was "Daniel met Sandra" results in 33% accuracy. Notice that human performance is strictly less than 100 %, using this criterion.

4.1.3 Data

The data is assembled from a subset of 5 Facebook toy problems [52]. From each of the task we extracted 120 training, 30 validation and 30 test examples, yielding the total of 600 training, 150 validation and 150 test examples. The validation was used to indicate when to stop training the network ("Early stopping"). Test set was used only once to evaluate the performance based on the set of parameters that achieved best performance on the validation set.

4.1.4 Models

Model A - Sequence-to-Sequence Approach

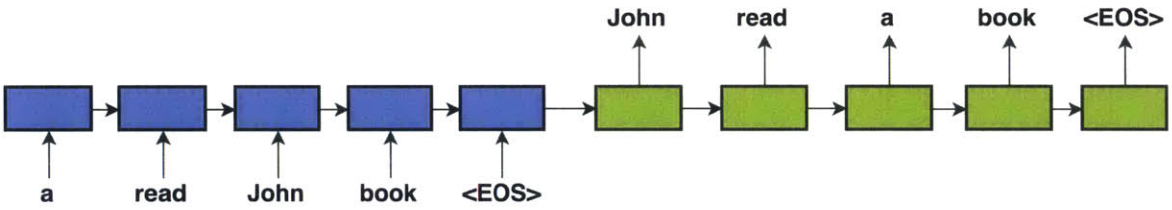


Figure 4-1: Model A: Sequence-to-Sequence model which takes a shuffled sentence as an input and outputs an unshuffled sentence. Blue network is the encoder LSTM and green network is the decoder LSTM.

This model is very similar to the translation model described in section 2.4. The only difference is that the input is a shuffled sentence and the output is the original sentence. The embedding size was 50 and both encoder network and decoder network has 3 LSTM layers of sizes 72, 62 and 50. To ensure that the results presented are the best possible for this type of model, we tried varying different parameters - solver type (vanilla stochastic gradient decent or Adam [20]), learning rate and batch size (the number of examples used to compute the gradient for each optimization step). The error function was the same as in case of the translation model - cross entropy error between the distributions of the predicted and the target words. The model is presented on figure 4-2.

Training procedure The network is optimized over as many input-output pairs as there are sentences in the training set. For each example, the original sentence is the expected output and input is its random permutation. Every epoch we use different permutations, but only one per sentence.

Model B - Reinforcement Learning Approach

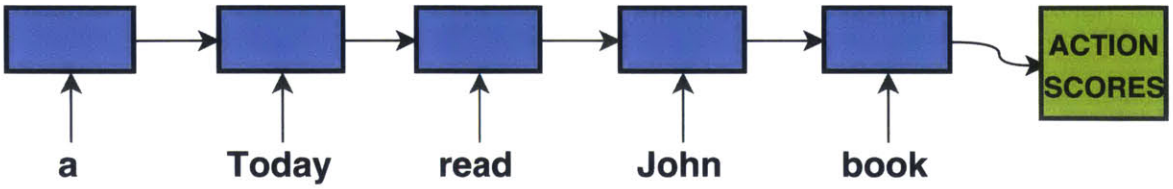


Figure 4-2: Model B: Neural Network model used for Q-value approximation in the Deep Q-learning algorithm.

To interpret the problem as an MDP, we consider the following formulation: State space consists of sequences of words. Each action modifies the order of those words. Rewards are aligned with our evaluation criterion.

Action Space We propose two alternative characterizations of actions space:

- *swaps* - action (x, y) corresponds to swapping elements on position x with element on position y. For example action $(0, 2)$ transforms "a Today read John book" into "read Today a John book".

- *moves* - action (x, y) corresponds to moving element on position x to position y-th position from the left (excluding its old position). For example action if we start with "a Today read John book" action $(2, 1)$ transforms it into "a read Today John book", while action $(2, 3)$ transforms it into a "Today John book read".

Both action spaces include a special $STOP$ action after which no further actions are taken.

Reward Function The reward was computed based on the following objective function. If the current state is composed of words in order $s = (w_1, ..., w_n)$ and the original sentence was $(g_1, ..., g_n)$, then the objective function is (where $[P]$ is the Iverson bracket):

$$o(s) = \sum_{i=1}^{n} [w_i = g_i]$$

The reward function is computed as change in the objective function as a result of an action, i.e. if we transition from state s_1 to state s_2 as a result of an action a, then the reward is $o(s_2) - o(s_1)$. This means that moving to state with higher objective value yields higher reward. Notice that in this formulation the value of the objective $o(s)$ is only equivalent to long term reward at state s as the discount rate γ increases to infinity.

The Network The Q-value network is presented on figure 4-2. The blue squares represent state parsing LSTM network with 3 hidden layers of sizes 100, 100, and 50. The final hidden state of the top layer is projected onto action space. The two most important parameters in the model are the discount rate γ and target network update rate τ (described in section 3.3.7).

Training Procedure In every epoch, the Deep-Q controller is used for unshuffling a random permutation of each of the training examples until it uses the $STOP$ action or until it executes more actions than twice the length of the input sentence.

Note on the Network Sizes

Networks in models A and B were chosen to ensure that the number of parameters is approximately the same. The total number of parameters in Model A is 178386, and in Model B it is 174022.

4.1.5 Results

Model A

Optimizer used	batch size	value of τ	epochs until convergence	validation accuracy	test accuracy
sgd	8	1	29	75	73
	8	0.1	29	62	60
	16	1	30	71	69
	16	0.1	23	49	48
adam	8	1e-2	28	82	81
	8	1e-3	26	77	76
	16	1e-2	30	79	77
	16	1e-3	30	72	71

Table 4.1: Results of model A on unshuffling task.

The results are summarized in table 4.1. The best performing model was trained with Adam using learning rate of 0.01 with batch size 8. In general lower batch sizes often improve prediction performance, but increase the total computation time. The best performing model correctly predicted 81% of words on the test set.

Model B

Actions space	value of γ	value of τ	epochs until convergence	validation accuracy	test accuracy
swaps	0.0	1e-4	16	85	86
	0.99	1e-4	13	85	86
	0.9999	1e-3	2	37	34
	0.9999	1e-4	11	85	86
moves	0.0	1e-4	17	61	62
	0.99	1e-4	12	71	72
	0.9999	1e-3	2	23	23
	0.9999	1e-4	12	83	82

Table 4.2: Results of model B on unshuffling task.

The results are summarized in table 4.2. The best result was achieved by the swap model which achieved 86% accuracy on the test set.

Effects of different parameters

- **Effect of action space** - While both actions sets achieve high performance, the swaps model achieves better performance. This is possibly because of the fact that actions in the swap model only act on two positions in the state, while moves can with a single move change elements on all the positions (when moving the last element to the front of the sentence). Therefore the latter action space has more complicated effect on state, which is harder to be encoded by the network.

- **Effect of target model update rate** - Using target network update rate is *essential* for proper training. When τ was set to 1.0, which is equivalent to using unaveraged target network, we were not able to achieve any results better than chance.

- **Effect of γ** - It turns out that in the swaps action space, greedy strategy is optimal, so varying γ has little effect on final performance. However, non-zero values of gamma improve convergence speed. In case of the moves action space, greedy strategy is no longer optimal and therefore higher values of γ yield better final performance.

4.1.6 Discussion

Performance

The best performance of the Reinforcement Learning model exceeds the best performance of Sequence-to-Sequence model by 5%. Additionally RL requires many fewer examples to converge - it converges after only 10 epochs (6000 example permutations), while the LSTM needs 30 epochs (18000 example permutations) to converge. Figure 4-3 shows a comparison of convergence speed of both models.

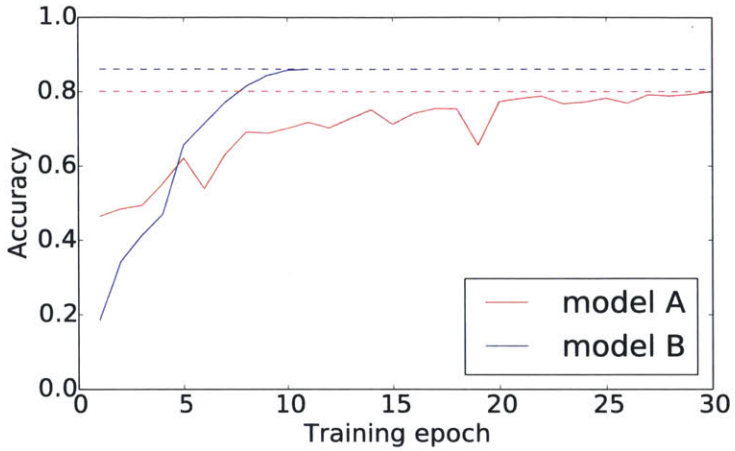


Figure 4-3: Comparison of the learning curves for the best parameters for model A and for model B. The values represent performance on the validation set.

Learned Representations

The representations learned by model B have the same desirable properties as the ones that often emerge in Sequence-to-Sequence models. Notice that all the are few separable clusters related to places, people, verbs applicable to people, verbs applicable to objects etc. Despite small number of examples used in training, the embeddings seem to be encoding semantically meaningful relations.

Error Analysis

Model A When browsing errors that the best instance of model A makes, we will observe one dominant type of an error: confusing the names of subject and objects of sentences. For example instead of "Sandra grabbed the football there", we get "Mary grabbed the apple there". This is likely due to the poor ability of LSTMs to memorize, which is studied in more detail in [12] - the publication shows that LSTM needs hundreds of thousands of examples to learn to memorize 20 8-bit vectors. Furthermore, even with that many examples, it never does better than average 1 bit error in copy task.

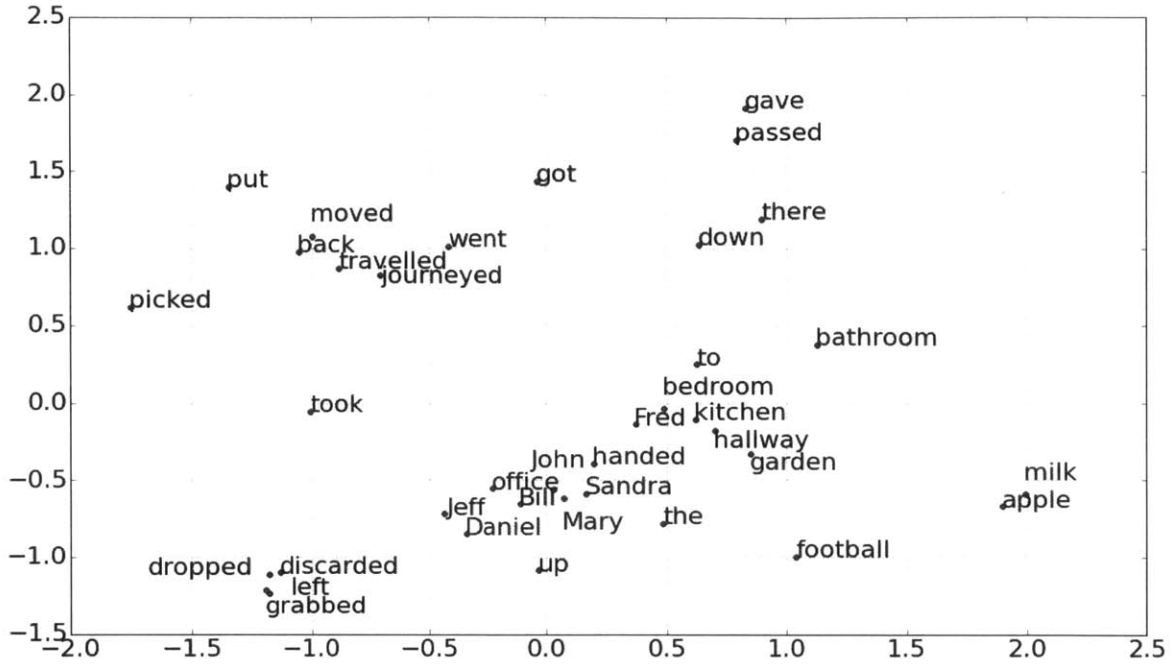


Figure 4-4: The word vectors being part of the best performing set of parameters for model B. The vectors have been projected on 2 dimensional plane using Spectral Embedding with a Gaussian kernel.

Model B Majority of the errors were unavoidable, for example "Daniel passed the apple to John" is one of the two correct answers along with "John passed the apple to Daniel". Occasionally the network "forgot" a single swap, for example "Mary put the down apple".

4.2 Learning by Doing

Learning by doing, or Experiential learning is the process of learning through experience, and is more specifically defined as "learning through reflection on doing"[11]. According to National Training Institute, retention in experimental learning (learning by doing) is significantly higher than in encyclopedic learning (learning by reading) [26]. One might wonder if this concept is in any way applicable to Machine Learning. The exercise described in this section is a stepping stone towards testing this hypothesis - we describe a framework where agent learns to understand natural language

sentence (learning) and is expected to perform a task in Reinforcement Learning setting (doing).

In this experiment we investigate a scenario where an agent needs to accomplish a mission such that its statement is written in natural language. The agent needs to choose actions based on the statement as well as the sensory information. Every mission statement is associated with a reward function which is positive if agent is getting closer to fulfilling the mission and negative otherwise. The approach described in this section can be applied to real-life robotic agents. It would be especially useful when it is non-trivial to separate the understanding from the execution, for example commands like "pass me the strange-looking book" or "neatly arrange items on the table" are not easily expressed symbolically. Furthermore, since the representations of words' meaning are obtained by doing, rather than reading, one could hope that they would amount to a qualitative jumpy in language understanding. That last claim could be further explored in the future work.

4.2.1 Experimental Setup

Figure 4-6 presents example state of the simulation. The state is composed of two parts - sensory data and natural language data. Sensory data is the same as the one described in section 3.4.1. Natural language data is a brief sentence describing the mission to be accomplished. State transitions are governed by physics, just like in section 3.4.1. The reward function is based on the task objectives. Task objectives are designed such that they are always non-negative and the mission is considered "accomplished" when their value is close to zero. The reward is based on changes in task objective and is equal to the difference in task objective between successive states. Table 4.3 describes all the types of tasks used in the simulation.

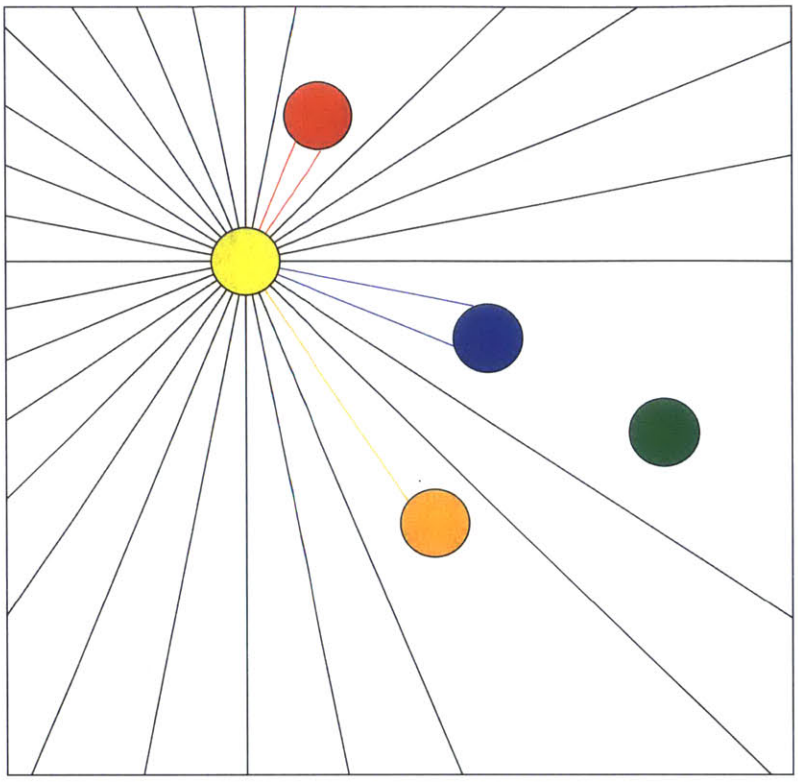


Figure 4-5: State of the simulation. The sensory data is computed based on the observation lines, and natural language data is computed based on mission statement. The yellow marble is an actor and red, green, blue and orange marbles are additional objects that participate in some tasks.

4.2.2 The Model

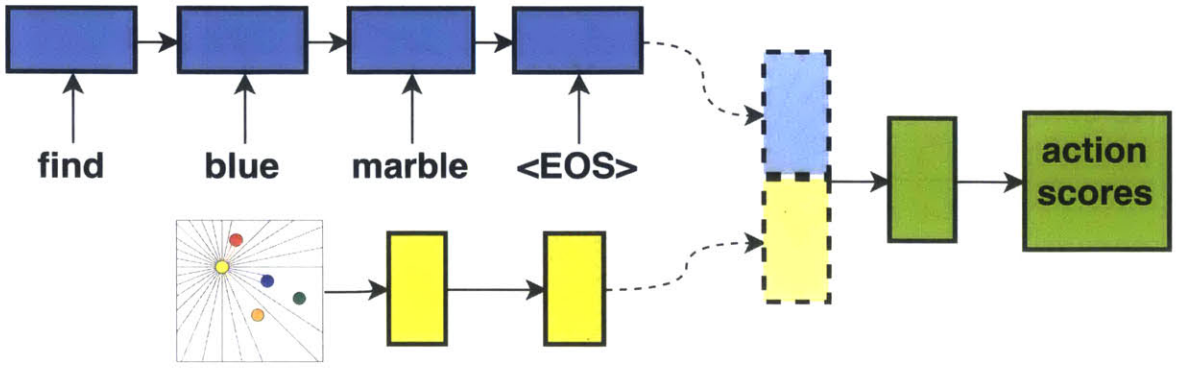
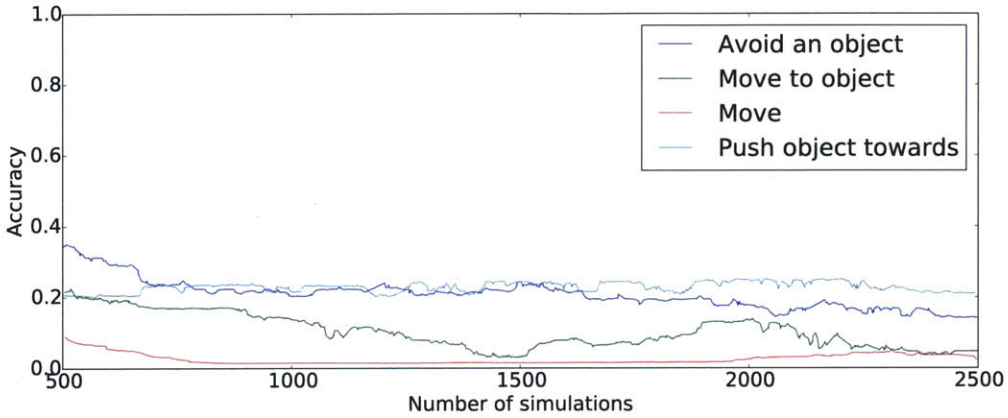


Figure 4-6: The model used in Deep Q-learning algorithm as Q value approximator. Colored in Blue is the LSTM network used to process the Natural Lanuage state (mission statement). Colored in yellow is the Multi-Layer Perceptron that processes the sensory input. Finally, colored in green is the high-level MLP which combines sensory and natural language data.

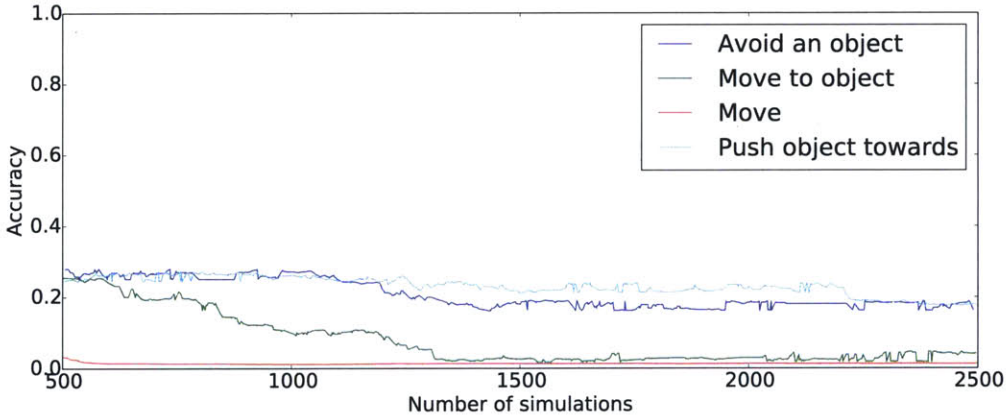
The model used to solve this problem is presented on figure 4-6. The sensory input is processed by *sensory MLP*, the Natural Language is processed by *language LSTM* and the final hidden states of both of those networks are concatenated and used as inputs to the *high-level MLP*, which outputs the action scores. Notice that concatentation can be easily handled by the backprogragtion algorithm, by splitting the gradient of the concatenated vector into two parts relevant to each of the inputs. We used RMSProp optimizer with learning rate 0.0001. We used discount rate $\gamma = 0.99$ and target network update rate $\tau = 0.0001$. *Sensory MLP* had two hidden layers of sizes 250 and 150, *language LSTM* had embedding size 50, one hidden layer of size 150, and *high-level MLP* had two hidden layers of size 200 and 100.

4.2.3 Results and Discussion

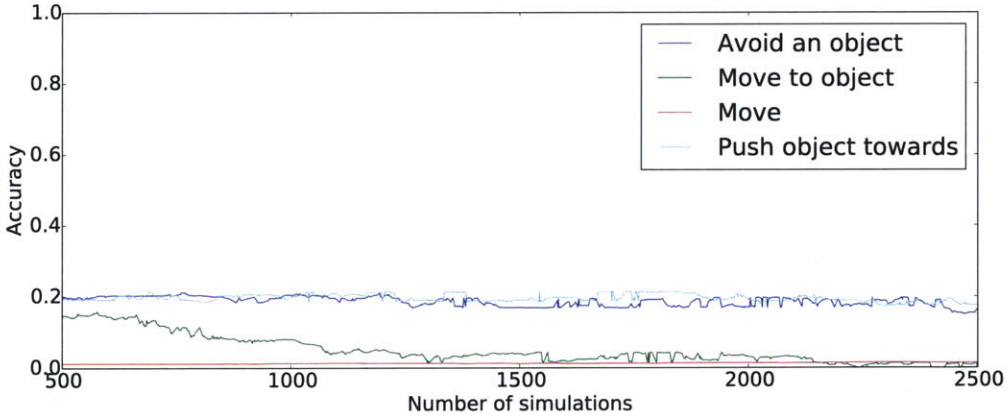
The results are summarized by the plots in figure 4-7. Our original model achieved perfect performance on the move task and got close to the same for Move to Object task. When evaluating different sets of parameters and observing the learned policies, we devised a number of optimizations summarized below.



(a) The model with the same learning rate across all the parameters and no goal state reward.



(b) The model with different learning rates for the LSTM and the rest of the model



(c) The model with different learning rates and extra goal state reward.

Figure 4-7: Performance of three different models in the manipulation problem. The accuracy which is y-axis of the plots is the value of task objective at the end of 10 second simulation (zero means perfect performance).

Reward discretization [30] suggests discretizing the reward function, i.e. all the reward increases are represented by 1, decreases by -1 and no change is represented by 0 reward. The reason for doing that is to avoid the need to hand tune learning rate for different tasks (because the scale of rewards can vary, which influences the norm of the gradient). Given the fact that in this problem we deal with multiple tasks within the same optimization process, one could argue this change would have positive impact on the overall performance. We found however that this approach leads to a very subtle bug. Because available actions correspond to acceleration, the agent was learning a strategy where it would accelerate quickly away from the goal state and then move slowly towards it. This results in high long term rewards, because there was a small number of transitions when backing away (negative reward) and many steps moving towards the target (positive reward).

Learning rate adaptation The way we set up the experiment, the Natural Language state representations are changing much less often, than the sensory data - for 10 second simulation at 30 frames per second, there will be a single value of natural language statement and 300 different sensory inputs (assuming the agent keeps moving). In order to balance that, we used different learning rate for the *language LSTM* than for the rest of the model - it was decreased to 0.00001 by a factor of ten with respect to the rest of the model. The effect on performance is presented on figure 4-7b. This optimization allows our model to confidently arrive at the near-optimal solution for the Move to Object task and increases the convergence speed.

Goal state reward Since it often takes time to discover a state of optimal solution of a particular task, we tried increasing the "region of attraction" of the correct solution by setting the reward to 1 unconditionally as soon as the task objective was zero in hope that it accelerates convergence. The effect on performance is presented on figure 4-7c. This optimization further increases the convergence speed.

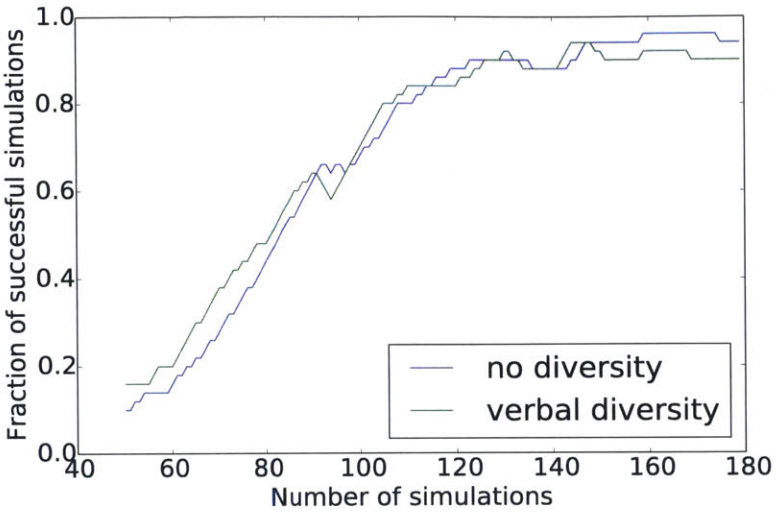


Figure 4-8: The effect of verbal diversity on convergence of the model.

Effect of Verbal Diversity

Real world language is very noisy, but nevertheless we would like the agent to be able to understand it. In order to understand how verbal diversity influences the learning, we run a single experiment where the agent was supposed to learn the *Move* task. We run the experiment in two settings:

- **no diversity** - only uses one type of statement, i.e. *move to <location> corner* where location is one of *upper left*, *upper right*, *bottom left* or *bottom right.*
- **verbal diversity** - uses multiple types of statements i.e. *<prefix> <location> <suffix>*, where prefix is one of *move to*, *arrive at*, *find* or *locate* and suffix is one of *promptly*, *now* or empty string.

The resulting convergence curves are presented on figure 4-8. There is no noticeable effect on convergence.

4.3 Conclusion

Our claim that multi-step reasoning in Reinforcement Learning settings can lead to improvements on Natural Language Processing tasks has been validated. We

have demonstrated 5% improvement in accuracy and 300% improvement in data efficiency with respect to the Sequence-to-Sequence model on sentence unshuffling problem. Furthermore, we have shown that our approach is capable of jointly learning to understand and execute Natural Language commands. For the four tasks that we proposed our model preforms near-perfectly on two of them and well on the other two.

Task	**description**	**task objective**	**example mission statements**
Move	The goal of this task is for the hero to move towards one of the corners of the board.	$\|a - c\| - \sqrt{2}r$ where c is corner location eg. $(0, 0)$ for upper left and a is agent location and r is radius of an object.	*Move to upper left corner, Apprach lower right corner promptly*
Move to object	The goal of this task is for the hero to move towards one an object of particular color.	$\|a - o\| - 2r$ where o is location of target object and a is agent location and r is radius of an object.	*Locate green marble now, Find orange thingy*
Avoid an object	The goal of this task is to move as far away as possible from a particular type of object.	$\|o - c\| - \|a - o\|$ where o is location of the object, a is location of an agent and c is location of a corner that is further away from the object o	*Move away from blue marble, Avoid blue fellow*
Push object towards	The goal of this task is to push object of one color towards an object of another color.	$\|o_1 - o_2\| - 2r$ where o_1 and o_2 are objects that we want to push towards each other and r is radius of an object.	*Push orange towards blue thingy, Push blue towards orange marble*

Table 4.3: Summary of tasks in learning by doing experiment.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

In this thesis we show how Reinforcement Learning can be used for Natural Language Processing by incorporating language as part of the state. We used Long Short-Term Memory Networks to parse the Natural Language input, where the final hidden state of the network is used as the input to Multi-Layer Perceptron computing action scores. We demonstrated a system capable of solving Natural Language problems, similar to Sequence-to-Sequence models, but capable of multi-stage reasoning. We used the approach to solve the sentence unshuffling problem and showed that it achieves accuracy 5% better than Sequence-to-Sequence model and that it requires 3 times less examples to converge.

Furthermore, we have shown that our approach is flexible and can be used with multi-modal inputs. We evaluated a system capable of understanding and executing Natural Language commands, which can be used for many different tasks with minimal engineering effort - the only required components being the reward function and example commands. We demonstrated that it is capable of achieving nearly perfect performance on two of the proposed tasks (Move to Corner and Move to Object) and good performance on the two others (Avoid Object and Push Object toward Another). Finally, we described a set of optimizations that improves convergence speed and the final performance.

5.2 Future Work

The work done for this thesis has inspired many ideas for future work, some of which are application-driven and others could further the capabilities of AI.

5.2.1 Beyond Deep Q Learning

The success of Deep Q learning is encouraging not only because of its performance, but also because, as evidenced by recent literature in Non-Convex optimization those methods allow for modeling flexibility, which encourages experimentation. One such idea is combining RL with NLP, but there are many other interesting ideas to purse. Particularly noteworthy examples include:

- **Relaxing Markovian assumption** - One of the major problems preventing the Deep Q-learning algorithm from achieving human-level performance on some of the Atari games is the fact that the policy chooses an action based only on current image on the screen, ignoring the past. An interesting approach to address this issue is presented in [15], in which the authors used LSTM to combine observations from fixed number of past timesteps. However considering the entirety of the past states is still an open problem. The author of this thesis proposes three approaches to this problem:
  - consider only the "surprising" past states (i.e. poor autoencoder performance or a result of a drastic change in state encoding)
  - construct a K clusters of past states and only consider one most recent state from each cluster the clusters are periodically recomputed during training
  - explicitly train choice function about which states to remember and which to discard, by subsampling candidate states to remember during training.
- **Reward attribution** - Consider the following scenario - given a state s, one needs to choose n actions before receiving reward: first one from set A_1, second

one from set A_2 and so on. It is an interesting problem to devise an approach that can efficiently navigate such an environment. One approach is to simply consider one "super-action" from set $A_1 \times A_2 \times ... \times A_n$, but the action space grows exponentially. Another approach is to consider an RL scenario where first $n-1$ decisions receive 0 reward, but the n-th decision receives the cumulative reward. This is also unsatisfactory, because the errors in Q-value estimate at first action increase exponentially with n. An interesting approach was demonstrated by Deepmind [37], where they shown a very simple algorithm which seems to be able to perform reward attribution for execution traces of go games.

5.2.2 Deep Q learning and NLP intersection

Chapter 4 merely laid out the foundations for what could be a part of much greater piece of research impacting Natural Language Processing and Autonomous Agent Control. The two directions that are interesting to pursue are:

- **Using RL for different NLP problems** - understanding for which problems, other than sentence unshuffling, the RL approach is beneficial. In particular, coreference resolution seems particularly well suited - one could define action space as adding/removing a pair of words to/from the coreference clique. Furthermore, by augmenting the action space with deletions, substitutions and swaps, we could perform translation, or improve the quality of translations output by the Sequence-to-Sequence model.

- **Further experimentation in robot manipulation based on NLP commands** - Two experiments that could be interesting are: building a bigger simulation with larger number of commands, and replicating the results on actual robots. In case of the latter it was already demonstrated that Deep Q-learning can be applied to real-world robots [1], but it would be even more impressive to demonstrate robot performing tasks, based on input spoken by a person.

[1] `https://www.youtube.com/watch?v=-YMfJLFynmA`

- **Question answering by imagining** - one could use the architecture similar to the one used in manipulation scenario, but instead of commands the network would receive questions and it would be asked to jointly simulate (use the "imagination") and output the answer to the question. First step would be to demonstrate the capability on a narrow domain, like path finding.

Bibliography

[1] D. Amodei, R. Anubhai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, J. Chen, M. Chrzanowski, A. Coates, G. Diamos, E. Elsen, J. Engel, L. Fan, C. Fougner, T. Han, A. Hannun, B. Jun, P. LeGresley, L. Lin, S. Narang, A. Ng, S. Ozair, R. Prenger, J. Raiman, S. Satheesh, D. Seetapun, S. Sengupta, Y. Wang, Z. Wang, C. Wang, B. Xiao, D. Yogatama, J. Zhan, and Z. Zhu. Deep Speech 2: End-to-End Speech Recognition in English and Mandarin. *ArXiv e-prints*, December 2015.

[2] Andrew J Barry and Russ Tedrake. Pushbroom stereo for high-speed navigation in cluttered environments. *arXiv preprint arXiv:1407.7091*, 2014.

[3] SRK Branavan, David Silver, and Regina Barzilay. Learning to win by reading manuals in a monte-carlo framework. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 268–277. Association for Computational Linguistics, 2011.

[4] Murray Campbell, A Joseph Hoane, and Feng-hsiung Hsu. Deep blue. *Artificial intelligence*, 134(1):57–83, 2002.

[5] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *The Journal of Machine Learning Research*, 12:2493–2537, 2011.

[6] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition.* The MIT Press, 3rd edition, 2009.

[7] Johan De Kleer and Brian C. Williams. Diagnosing multiple faults. *Artificial intelligence*, 32(1):97–130, 1987.

[8] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[9] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12:2121–2159, 2011.

[10] Manaal Faruqui and Chris Dyer. Improving vector space word representations using multilingual correlation. Association for Computational Linguistics, 2014.

[11] Patrick Felicia. *Handbook of Research on Improving Learning and Motivation through Educational Games: Multidisciplinary Approaches.* IGI Global, 2011.

[12] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[13] Tatsunori B. Hashimoto, David Alvarez-Melis, and Tommi S. Jaakkola. Word, graph and manifold embedding from markov processes. *CoRR*, abs/1509.05808, 2015.

[14] Hado V Hasselt. Double q-learning. In *Advances in Neural Information Processing Systems*, pages 2613–2621, 2010.

[15] Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. *arXiv preprint arXiv:1507.06527*, 2015.

[16] Geoffrey E Hinton. Learning distributed representations of concepts. In *Proceedings of the eighth annual conference of the cognitive science society*, volume 1, page 12. Amherst, MA, 1986.

[17] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, and Jürgen Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

[18] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[19] Daniel Kahneman. *Thinking, fast and slow.* Macmillan, 2011.

[20] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[21] Ryan Kiros, Yukun Zhu, Ruslan Salakhutdinov, Richard S Zemel, Antonio Torralba, Raquel Urtasun, and Sanja Fidler. Skip-thought vectors. *arXiv preprint arXiv:1506.06726*, 2015.

[22] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[23] Ankit Kumar, Ozan Irsoy, Jonathan Su, James Bradbury, Robert English, Brian Pierce, Peter Ondruska, Ishaan Gulrajani, and Richard Socher. Ask me anything: Dynamic memory networks for natural language processing. *arXiv preprint arXiv:1506.07285*, 2015.

[24] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[25] Steven J. Levine and Brian C. Williams. Concurrent plan recognition and execution for human-robot teams. In *ICAPS-14*, 2014.

[26] Saranne Magennis and Alison Farrell. Teaching and learning activities: Expanding the repertoire to support student learning. *Emerging issues in the practice of university learning and teaching*, 1, 2005.

[27] Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. The Stanford CoreNLP natural language processing toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60, 2014.

[28] G Nigel N Martin, Glen G Langdon Jr, and Stephen JP Todd. Arithmetic codes for constrained channels. *IBM Journal of Research and Development*, 27(2):94–106, 1983.

[29] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[30] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[31] Joseph Moore, Rick Cory, and Russ Tedrake. Robust post-stall perching with a simple fixed-wing glider using lqr-trees. *Bioinspiration & biomimetics*, 9(2):025013, 2014.

[32] Frederic Morin and Yosh2001ua Bengio. Hierarchical probabilistic neural network language model. In *Proceedings of the international workshop on artificial intelligence and statistics*, pages 246–252. Citeseer, 2005.

[33] Karthik Narasimhan, Tejas Kulkarni, and Regina Barzilay. Language understanding for text-based games using deep reinforcement learning. *arXiv preprint arXiv:1506.08941*, 2015.

[34] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. *Proceedings of the Empiricial Methods in Natural Language Processing (EMNLP 2014)*, 12:1532–1543, 2014.

[35] Jonathan Raiman. Building blocks for the mind. Master's thesis, Massachusetts Institute of Technology, 2015.

[36] Jonathan Raiman and Szymon Sidor. Occam's gates. *arXiv preprint arXiv:1506.08251*, 2015.

[37] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray

Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, January 2016.

[38] Richard Socher, Jeffrey Pennington, Eric H. Huang, Andrew Y Ng, and Christopher D Manning. Semi-supervised recursive autoencoders for predicting sentiment distributions. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 151–161. Association for Computational Linguistics, 2011.

[39] Nitish Srivastava. *Improving neural networks with dropout.* PhD thesis, University of Toronto, 2013.

[40] Danny Sullivan. Google: 100 billion searches per month, search to integrate gmail, launching enhanced search app for ios. *Search Engine Land. http://searchengineland. com/google-search-press-129925*, 2012.

[41] Ilya Sutskever, Oriol Vinyals, and Quoc VV Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[42] Kai Sheng Tai, Richard Socher, and Christopher D Manning. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*, 2015.

[43] Gerald Tesauro. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[44] Jörg Tiedemann. Improved sentence alignment for movie subtitles.

[45] Tijmen Tieleman. *Optimizing Neural Networks that Generate Images.* PhD thesis, University of Toronto, 2014.

[46] Oriol Vinyals and Quoc Le. A neural conversational model. *arXiv preprint arXiv:1506.05869*, 2015.

[47] Matthew R Walter, Matthew Antone, Ekapol Chuangsuwanich, Andrew Correa, Randall Davis, Luke Fletcher, Emilio Frazzoli, Yuli Friedman, James Glass, Jonathan P How, et al. A situationally aware voice-commandable robotic forklift working alongside people in unstructured outdoor environments. *Journal of Field Robotics*, 32(4):590–628, 2015.

[48] Andrew J. Wang and Brian C. Williams. Chance-constrained scheduling via conflict-directed risk allocation. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, January 2015.

[49] Maxwell B Wang, Albert Chu, Lawrence A Bush, and Brian C Williams. Active detection of drivable surfaces in support of robotic disaster relief missions. In *Aerospace Conference, 2013 IEEE*, pages 1–13. IEEE, 2013.

[50] Leila Wehbe, Ashish Vaswani, Kevin Knight, and Tom Mitchell. Aligning context-based statistical models of language with brain activity during reading.

[51] Paul J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[52] Jason Weston, Antoine Bordes, Sumit Chopra, and Tomas Mikolov. Towards ai-complete question answering: A set of prerequisite toy tasks. *arXiv preprint arXiv:1502.05698*, 2015.

[53] Jason Weston, Antoine Bordes, Sumit Chopra, and Tomas Mikolov. Towards ai-complete question answering: a set of prerequisite toy tasks. *arXiv preprint arXiv:1502.05698*, 2015.

[54] Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. *CoRR*, abs/1410.3916, 2014.

[55] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[56] Yukun Zhu, Ryan Kiros, Richard Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *arXiv preprint arXiv:1506.06724*, 2015.