INDIAN INSTITUTE OF TECHNOLOGY MADRAS

Learning Language And Expanding Vocabulary Using Reinforcement Learning

by

Ghulam Ahmed Ansari

(EE12B134)

A thesis submitted in partial fulfillment for the degree of

BACHELOR OF TECHNOLOGY & MASTER OF TECHNOLOGY

in the

Electrical Engineering Department

under the guidance of

Dr. B. Ravindran

&

Dr. Kaushik Mitra

May 2017

THESIS CERTIFICATE

This is to certify that the thesis titled **Learning Language And Expanding Vocabulary Using Reinforcement Learning**, submitted by **Ghulam Ahmed Ansari**, to the Indian Institute of Technology, Madras, for the award of the degree of **Bachelor of Technology** and **Master of Technology**, is a bona fide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. B. Ravindran
Research Guide
Associate Professor
Dept. of Computer Science
IIT-Madras, 600 036

Dr. Kaushik Mitra
Research Guide
Assistant Professor
Dept. of Electrical Engineering
IIT-Madras, 600 036

Place: Chennai
Date: May 2017

INDIAN INSTITUTE OF TECHNOLOGY MADRAS

Abstract

Learning Language And Expanding Vocabulary Using Reinforcement Learning

by Ghulam Ahmed Ansari

Text-based games are suitable test-beds for designing agents that can learn by interacting with the environment through natural language text. Very recently, deep reinforcement learning based agents have been successfully applied for playing text-based games. In this paper, we explore the possibility of designing a single agent to play several text-based games and of expanding the agent's vocabulary using the vocabulary of agents trained for different games separately. To this extent, we explore the application of recently proposed policy distillation method for video games to the text-based game setting and show that this method is able to learn rich word representations.

Acknowledgements

I would like to express my thanks to the people who have helped me most throughout my project. I am extremely grateful to my research adviser, **Prof. B. Ravindran**, for supporting and guiding me throughout the project.

I would also like to offer my special thanks to **Sarath Chandar**, (Ph.D student , Department of Computer Science, University of Montreal) for his constant support, guidance and motivation as a friend. I am also grateful to his valuable suggestions and feedback in designing the right set experiments and understanding the results.

I wish to thank **J.P Sagar**, (Dual Degree Student, Department of Computer Science IIT Madras) for designing the game environments, being a great companion during the literature survey and also for assisting in interfacing the lua based game environment with our python based core module.

I am grateful to the members of the reviewing committee of **ACL**, a premier conference of the field of computational linguistics, who provided interesting insights and suggestions with regard to a short paper submission of this work titled **Language Expansion In Text-Based Games**.

I wish to thank **Prof. Mitesh Khapra**, Department of Computing Science, IIT Madras, for his suggestions and views.

I also wish to thank **Prof. Kaushik Mithra**, Department of Electrical Engineering, IIT Madras, for his guidance in helping me become a competent researcher.

Finally, I wish to thank my parents, my friends, and the faculty of Department of Computer Science and Department of Electrical Engineering, IIT Madras for their kind support and assistance.

Ghulam Ahmed Ansari

Contents

List of Figures

List of Tables

Abbreviations

RL	**R**einforcement **L**earning
MDP	**M**arkov **D**ecision **P**rocess
DQN	**D**eep **Q** **N**etwork
LSTM	**L**ong **S**hort **T**erm **M**emory
MUD	**M**ulti **U**ser **D**ungeon
t-SNE	**t**-Distributed **S**tochastic **N**eighbor **E**mbedding

Notations

$\mathcal{S}$	State space of a MDP
$\mathcal{A}$	Action space of an MDP
$P(.)$	Conditional probability distribution of next state and reward obtained on state transition
γ	Discount factor
θ	A variable weight parameter
$\pi(.)$	A policy on a MDP
Q^{π}	State-Action value function of policy π
$length(.)$	length of an episode e
$\mathbb{E}[.]$	Expectation of a random variable
$\mathcal{D}$	Set of demonstrations on M following expert's policy

Chapter 1

Introduction

In this section, we will give a brief introduction to the concepts in reinforcement learning and machine learning that are needed to have a better understanding of the problem that we are interested in. The technical preliminaries are postponed to a later section. We will begin by discussing the full reinforcement learning framework in section 1.1 following which in section 1.4 we will describe the concept of imitation learning. Finally, in section 1.5, we briefly describe the problem that forms the goal of this work.

1.1 Reinforcement Learning

Reinforcement learning (RL) is the branch of machine learning that can be used to model sequential decision making problems. It considers an agent situated in an environment: each time-step, the agent takes an action, and it receives an observation and reward. An RL algorithm seeks to maximize the agents total reward called the return, given a previously unknown environment, through a trial-and-error learning process.

As can be seen in Figure 1.1, the execution of an action results in two kinds of feedback from the environment. First, the agent receives a reward. Second, the environment makes a transition to another state. Both the above effects are dependent only on the state the agent was in when it took the action, and the action itself - but not any event that happened further back in time. The above system of the agent, the environment, states, actions, transitions and rewards is encapsulated as a Markov Decision Process.

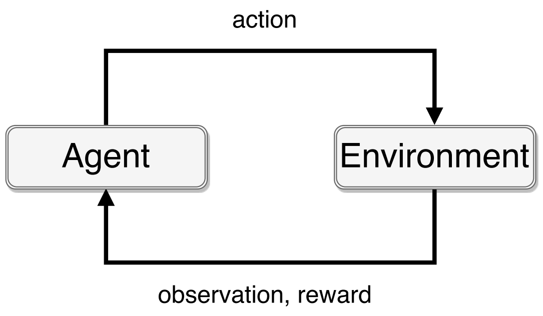


Figure 1.1: A general reinforcement learning setting

When an agent is thus introduced to an unknown environment it has to learn through experience what is the best action to be taken at every state. The quality of the action can be understood to correspond to the amount of return that is expected by taking that action. We will call this state-action mapping that is to be learned as the optimal policy. This process of learning will involve exploration - that enables to the agent to understand the environment better - and also exploitation that ensures that the agent makes best use of what it has learned. Thus, when an agent explores and receives a high reward, the reward is a means of reinforcement which encourages the agent to believe that the steps it took recently are good and can be exploited later. Chapter 2 provides a more detailed description of the mathematical formulation of reinforcement learning.

With this basic idea of the reinforcement learning problem, we will first briefly discuss the area of deep reinforcement learning and then move on to describing the concept of imitation learning.

1.2 Deep Learning

Modern machine learning is mostly concerned with the problem of learning a function to represent the given data. In the recent years Deep learning has revolutionized the way we do machine learning. It revolves around finding a good set of parameters to a deep neural network (function approximator). A loss function has to be chosen depending upon the problem at hand, and then the parameters are optimized using gradient descent or its variants. Deep Learning has been successfully applied and outperformed the existing methods in a wide range of problems like object detection (Girshick, 2015), object recognition (Krizhevsky et al., 2012), and speech recognition (Dahl et al., 2010).

We will now give a brief introduction on the combination of reinforcement learning and deep neural networks.

1.3 Deep Reinforcement Learning

Deep reinforcement learning is the study of reinforcement using neural networks as function approximators. The integration of reinforcement learning and neural networks dated back to 1990s- Tesauros TD-Gammon (Tesauro, 1995), developed in the early 1990s, used a neural network value function and played at the level of top human players. Lins 1993 thesis (Lin, 1992) explored the combination of various reinforcement learning algorithms with neural networks, with application to robotics. With recent advances in deep learning (Lecun et al., 2015; Goodfellow et al., 2014), benefiting from powerful computational resources and new algorithmic techniques, we have been witnessing an immense growth in the area of deep reinforcement learning, i.e., a combination of reinforcement learning and deep neural networks. This rise of interest in deep reinforcement learning primarily occurred following the results from Mnih et al. (2015), who demonstrated learning to play a collection of Atari games, using only feed from the screen as input, using a variant of Q-learning. Since then, there have been many interesting results like AlphaGo (Silver et al., 2016) and differentiable neural computer (Graves et al., 2016); and novel architectures and applications, like asynchronous methods (Mnih et al., 2016) etc.

1.4 Imitation Learning

Recall that the goal of the reinforcement learning problem is to find the optimal policy i.e., a mapping that tells us what is the best action to perform at a given state in a sequential decision making process. The imitation learning problem gives an interesting twist to this problem: we no longer have a notion of an optimal policy; instead we want to learn a policy that best mimics the behavior of another agent whom we choose to call the teacher or expert. Imitation learning is extremely useful when we are dealing with an environment with no access to accurate reward signals. In such a case where there is no reinforcement in form of rewards, manually generating reward signals and tuning them until the task is learnt is a ineffective apprach. Instead, having an expert or teacher to demonstrate its optimal policy is a better option. Next, we talk about how imitation learning plays an important role in multi-task learning.

1.5 Multi-task learning

The ability to act in multiple environments and transfer previous knowledge to new situations can be considered a critical aspect of any intelligent agent. Thus it is important for an agent to learn how to behave in multiple tasks simultaneously, and then generalize its knowledge to new domains. The ability to perform multiple task simultaneously is referred to as multi-task learning. Our work is built upon one method of multi-task learning namely, policy distillation (Rusu et al., 2015). Policy distillation was introduced as an approach for designing single agent that can play multiple video-based games (atari games). The basic recipe of policy distillation is to extract expert trajectories from teachers trained on various games and train a new agent to play all the games by mimicking the expert trajectories of the teachers.

Given that (Narasimhan et al., 2015) demonstrated that it is possible to learn language by interaction with a text-based game environment, our goal is to be able to combine these language representations learned by playing different text-based games.

1.6 Summary

The rest of the document is organized as follows. Chapter 2 formally introduces the terms and definitions that will be used for discussing the problem. In particular, we will introduce the reader to the notations that are common in reinforcement learning. We will also define and explain the LSTM-DQN framework and policy distillation method, which are important for understanding our work. In Chapter 3, we detail our motivation in solving this problem in the light of earlier work and then proceed to discussing our solution and insights. Finally, in Chapter 4 we summarize our contributions.

Chapter 2

Background Theory

2.1 Markov Decision Processes (MDP)

A reinforcement learning problem can often be modelled as a Markov Decision Process (MDP). The interactions of the agent with the environment are described by the following components:

- $\boldsymbol{\mathcal{S}}$: ***state space***

  A set of all the states in the environment

- $\boldsymbol{\mathcal{A}}$: ***action space***

  A set of actions available to the agent at every time-step

- $\boldsymbol{P(r, s'|s, a)}$: ***Transition probability***

  Given that the agent is in a state s and an action a is taken, $P(r, s'|s, a)$ specifies the probability that the environment will transition to a state s' and output a reward r

The final goal here is to find a policy **policy** π, which maps states to actions. Policies can be either stochastic or deterministic. A stochastic policy corresponds to a conditional distribution given as $\pi(s|a)$, where as deterministic policies are generally written as $a = \pi(s)$.

2.2 The Episodic Reinforcement Learning Problem

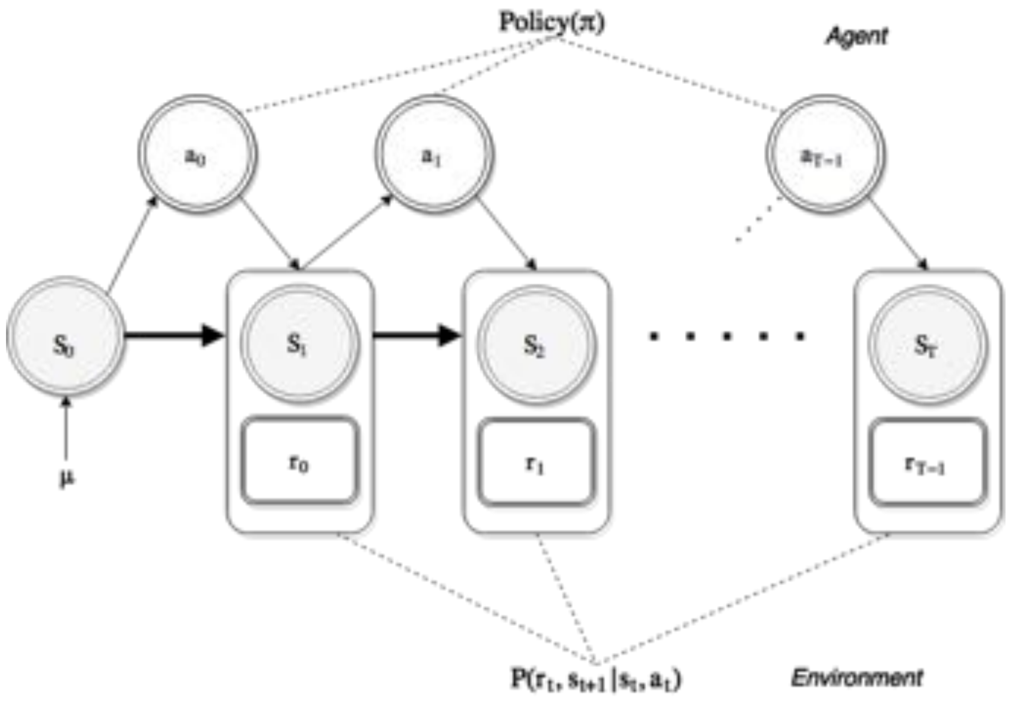


Figure 2.1: Episodic Reinforcement Learning Schematic

In this thesis we will consider text-based game environments that are episodic in nature, where the agents experience is broken up into a series of episodes with a finite number of states, actions and rewards. Episodic reinforcement learning in the fully-observed setting is defined by the following process. Each episode begins by sampling an initial state of the environment, s_0 from a probability distribution $\mu(s_0)$. At each time-step $t = 0, 1, \ldots$ the agent chooses an action $a_t \in \mathcal{A}$, sampled from the distribution $\pi(a_t|s_t)$. Here, the distribution π is known as the policy, which is the probablity distribution that the agent uses to sample it's actions from. After taking the action, the environment generates the next state and reward, according to some distribution $P(r_t, s_{t+1}|s_t, a_t)$. Finally, the episode ends when a terminal state s_T is reached. This process is described in Figure 2.1.

The goal is to find a policy π that optimizes the expected total reward per episode.

$$\max_{\pi} \mathbb{E}_{\phi}(R|\pi)$$

The expectation is taken over a set of trajectories ϕ which are generated by using policy π to take actions. Here R is known as the Return. For a given episode, the return R is defined as

$$R = r_0 + r_1 + \ldots +_{length(episode)-1}$$

2.3 State Action Value Function (Q-function)

A state action value function helps us understand how desirable is an action under a policy. It can be defined as,

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}[R|s_0 = s, a_0 = a]$$

Where the agent follows the policy π to pick actions at every state except s_0.

An optimal policy π^* is such that $\forall s \in \mathcal{S}$,

$$Q^{\pi^*}(s, a) = \max_{\pi} Q^{\pi}(s, a)$$

2.4 Q-learning

Q-Learning (Watkins and Dayan, 1992) is a model-free technique which can be used to learn an optimal $Q(s,a)$ for the agent. In a state s the agent takes an action a by consulting a state-action value function $Q(s,a)$, which is a measure of the action's expected long-term reward. In Q-Learning, the Q-function is initialized randomly. Then, by playing the game and obtaining rewards, the agent continuously updates its Q-values according to Equation 2.1. The iterative updates are derived from the Bellman equation (Sutton and Barto, 1998):

$$Q_{i+1}(s,a) = \mathrm{E}[r + \gamma \max_{a'} Q_i(s',a') \mid s,a] \tag{2.1}$$

where γ is a discount factor for future rewards and the expectation is over all game transitions that involved the agent taking action a in state s.

In a state s the agent uses the Q-values to choose an action $a^{'}$, given by:

$$a^{'} = \arg\max_{a} Q(s,a) \tag{2.2}$$

In practice, it is necessary to take non greedy actions and explore the state space. This can achieved by following an ϵ-greedy policy (Sutton and Barto, 1998), where the agent performs a random action with probability ϵ.

2.5 Deep Q Networks (DQN)

In most real world environments the state representation is very high dimensional. In such state spaces it is difficult for an agent to experience all the states even once. Thus we need some form of generalization over the state action pairs. Moreover as in Q-learning it is infeasible to store all the state action pairs, thus we cannot use any tabular methods like Q-learning. A solution to this problem, as pointed out by many function approximation methods, is to use a parametrized state action value function $Q(s, a; \theta)$. However, creating a good parametrization requires knowledge of the state and action spaces. One way to bypass this feature engineering is to use a Deep Q-Network (DQN) (Mnih et al., 2015). The DQN approximates the Q-value function with a deep neural network to predict $Q(s, a)$ for all possible actions a simultaneously, given the current state s.

2.6 LSTM-DQN

LSTM-DQN is a reinforcement learning framework for learning to play MUD[1] games. The action space in this architecture was restricted to be of one action word and one object word (e.g. *go east*). LSTM-DQN agent proposed in (Narasimhan et al., 2015) differs from the standard DQN agent (Mnih et al., 2015) in two ways as can be seen in Figure 2.2.

Firstly, since the state is a sequence of words in the case of text-based games, LSTM-DQN uses an LSTM layer for state representation instead of a convolutional layer. This LSTM layer will have different vocabulary for different games. Following (Narasimhan et al., 2015) a whole sentence is passed through the LSTM word by word and an output is generated by it for every word. The mean of all these outputs of LSTM is passed to the next fully connected layer of the network. Secondly, MUD-games use multi-word textual commands as actions. For each state in the game, Q-values of all possible actions and all

[1]http://mudstats.com/

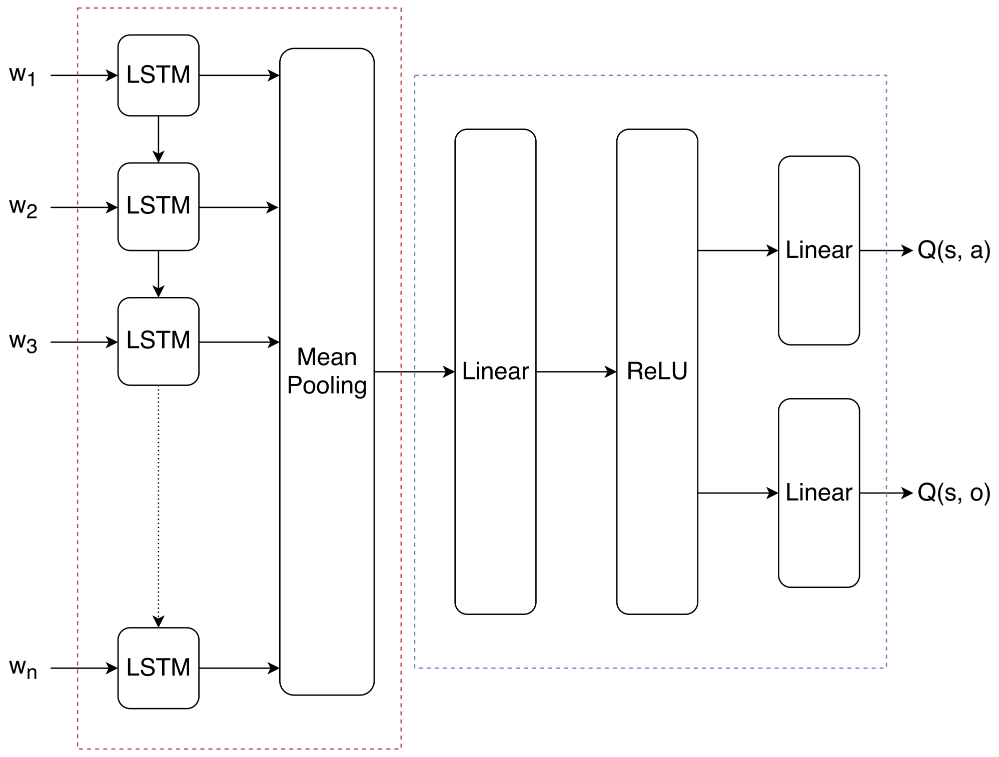


Figure 2.2: LSTM-DQN Architecture

available objects are predicted using the same network. An average of the Q-values of the action a and the object o is used as a measure of the Q-value of the entire command (a, o). The action and object with the highest predicted Q-value is chosen by the agent, to maximize the expected long-term rewards.

2.7 Multi-Task Policy Distillation

In Multi-Task learning the goal is to learn a single agent that can learn to perform multiple tasks. In our work we employ a method known as policy distillation (Rusu et al., 2015), one among few methods for Multi-Task Learning.

Distillation is a method to transfer knowledge from a teacher model T to a student model S. This technique is generally employed to learn a simpler classification network from a complex one. The distillation targets are typically obtained by passing the weighted sums of the last network layer of the classification network, through a softmax function. Generally, a temperature parameter τ is also used in the softmax, in-order to control the amount of knowledge transfer.

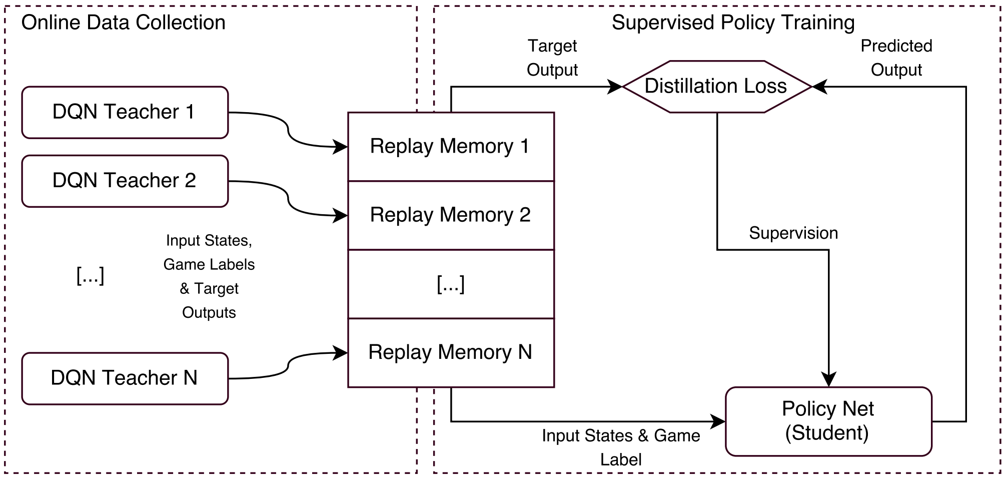


Figure 2.3: Multi-Task Policy Distillation architecture

Rusu et al. (2015) describes Multi-Task *Policy distillation* method for training a single agent that can play multiple Atari games that have similar structure and dynamics. The approach for multi-task policy distillation, illustrated in Figure 2.3, is straightforward. It uses n DQN single-game experts, each trained separately. These agents produce inputs and targets, just as with single-game distillation, and the data is stored in separate memory buffers. The distillation agent then learns from the n data stores sequentially, switching to a different one every episode. Since different tasks often have different action sets, a separate output layer (called the controller layer) is trained for each task and the ID of the task is used to switch to the correct output during both training and evaluation.

Chapter 3

Learning Language And Expanding Vocabulary Using Reinforcement Learning

3.1 Motivation

The exact process of language learning in human is still unclear, but it is certain that interaction and feedback from the environment plays a pivotal role in shaping the language centers of the human brain. As far as learning language by machines is considered, we generally follow a frequentist approach where wherein thousands of labeled textual examples are used to learn a representation of language. Learning language in this fashion undermines the essence of the human communication, i.e., understanding. There has been very little progress towards language understanding, as even the current metrics for evaluating representations of language do not explicitly penalize or reward based on the closeness of the representations to human language. Our goal is to narrow the gap between machine and human language learning. One way to achieve this as proposed by Narasimhan et al. (2015) is to allow an agent to interact with an environment using natural language commands. In this way, we can intelligently tailor environments to impart language information to an agent.

An important measure of an agent's language representation is the size of it's vocabulary. In the interaction based learning framework, the vocabulary of an agent is only so good as the size of the vocabulary of the environment. The biggest problem that arises with the interaction based approach is that it is not feasible to build an environment encompassing an entire language.

3.2 Contribution

In our work, we propose a solution to the practical limits on environments size by proposing a framework to constructively combine knowledge gained by different agents trained in nonidentical environments irrespective of the scale of the environments. This way we not only expand the vocabulary learned, but we also bypass the problem of environment creation as there are a plethora of text-based games that can be used for this purpose.

By playing different games simultaneously and generating a representation of language based upon all the games, we learn a representation that is better than each of the individual representations learned by playing a single game. In our experiments, the games are designed in a way that mastering one game helps the learning of another game when the context of the games are similar.

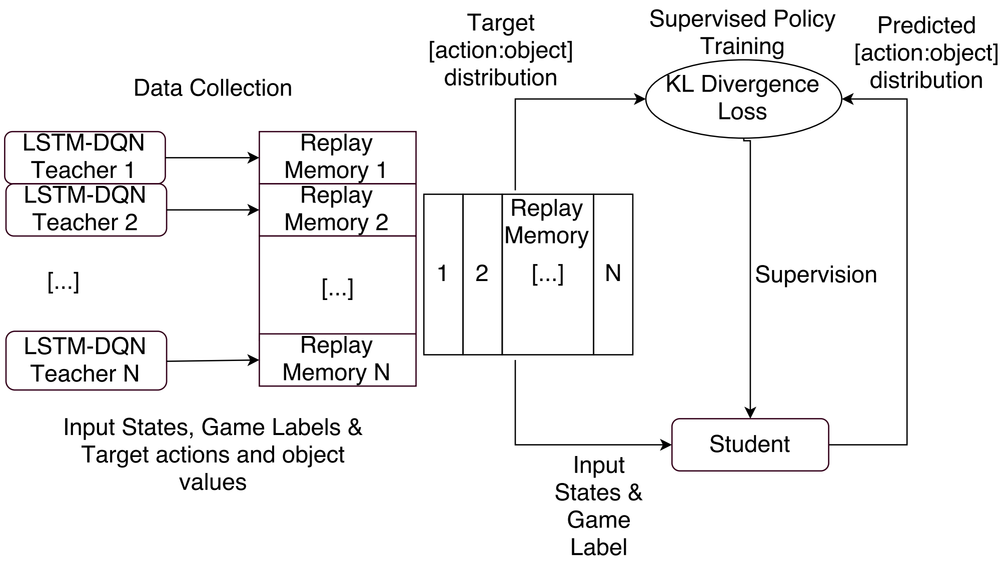
3.3 Multi-task Distillation Agent for Learning Representations from Multiple Sources

Now we describe our method to learn language representations from multiple games with stochastic textual descriptions. First we train n single-game LSTM-DQN teachers (T) separately. We want a single agent to be able to play all n games, therefore we train a separate student neural network (S) to learn the optimal policies of each of these games. Each of the n expert teachers produce a set of state-action and state-object values which we store along with inputs and game labels in a memory buffer. A separate memory buffer is maintained for each game as shown in Figure 3.1a and training happens sequentially, i.e training is done one by one, on samples drawn from a single game buffer at a time.

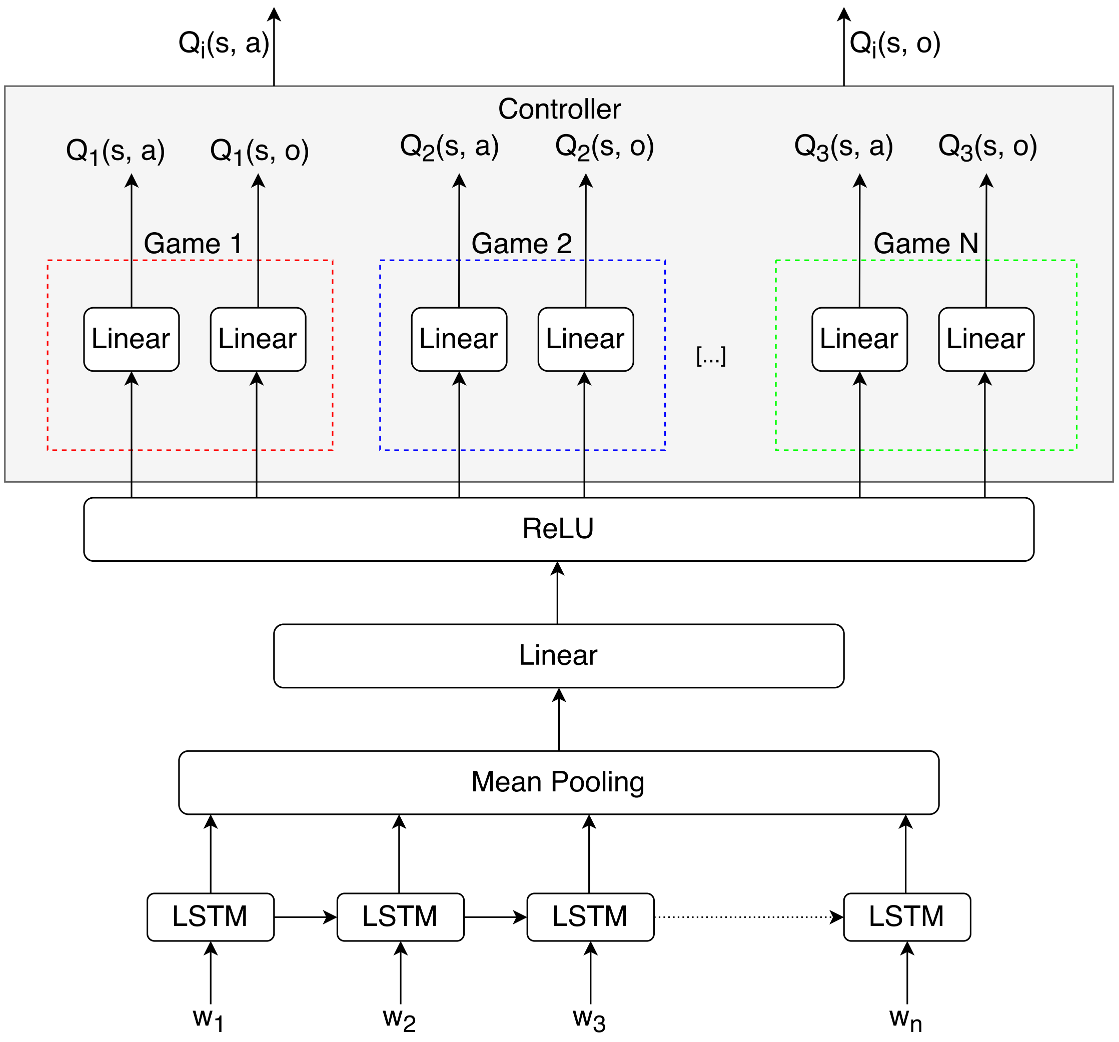
As we have n different games and each game has to output an action and object at each time-step, the student network S has a controller which can shift between n different action, object output modules as shown in Figure 3.1b. The student network uses the input game label to switch between the corresponding modules.

The outputs of the final layer of the teachers are used as targets for the student S, after passing through a softmax function with a temperature parameter (τ). The targets to the student S are thus given by softmax($\frac{\mathbf{q}^T}{\tau}$), where $\mathbf{q}^T$ is the vector of Q-values of Teacher T. The outputs of the final layer of the student S are also passed through a softmax function and can be given by softmax($\mathbf{q}^S$), where $\mathbf{q}^S$ is the vector of Q-values of Student S. KL divergence loss function is used to train the student network S.

$$\mathcal{L}(X, \theta_S) = \sum_{i=1}^{|X|} softmax(\frac{\mathbf{q}_i^T}{\tau}) ln \frac{softmax(\frac{\mathbf{q}_i^T}{\tau})}{softmax(\mathbf{q}_i^S)}$$


Data Collection
LSTM-DQN Teacher 1
LSTM-DQN Teacher 2
[...]
LSTM-DQN Teacher N
Replay Memory 1
Replay Memory 2
[...]
Replay Memory N
Input States, Game Labels & Target actions and object values
Target [action:object] distribution
1
2
Replay Memory
[...]
N
Supervised Policy Training
KL Divergence Loss
Predicted [action:object] distribution
Supervision
Student
Input States & Game Label

(a) Multi Task Policy Distillation for Text-based games



(b) Student Network Architecture

Figure 3.1: Learning representations from multiple text-based games

where dataset $X = \{(s_i, \mathbf{q}_i)\}_{i=0}^{N}$ is generated by Teacher T, where each sample consists of a short observation sequence s_i and a vector $\mathbf{q}_i$ of un-normalized Q-values with one value per action and one value per object. In simpler terms, $\mathbf{q}_i$ is a concatenation of the output action and object value function distributions. We call this agent as multi-task distillation agent. This is a straight-forward extension of distillation agent for video games proposed in Rusu et al. (2015) to text-based games.

We also train a multi-task LSTM-DQN agent as a baseline. For multi-task LSTM-DQN, the approach is similar to single-game learning. The network is optimized to predict the average discounted return of each possible action given a small number of continuous observations. The training is similar to multi-task distillation. The current game is switched after every episode, separate replay memory buffers are maintained for each task, and training is evenly interleaved between all tasks. The game label is used to switch between different output modules as in multi-task policy distillation, thus enabling a different output layer, or controller, for each game. The multi-task LSTM-DQN loss function remains identical to single-task learning (Narasimhan et al., 2015).

Chapter 4

Experiments and Analysis

In this section, we do an extensive analysis of policy distillation applied to LSTM-DQN. Specifically, we are interested in answering the following questions:

1. How to learn multiple games with contradicting state dynamics?
2. What is the mechanics of multi-task policy distillation?
3. What can we say from the visualization of word embeddings?
4. How to do transfer learning using policy distillation?
5. How is the performance of policy distillation when compared to multi-task LSTM-DQN?

We first explain the text-based game environment used in the experiments before reporting our findings.

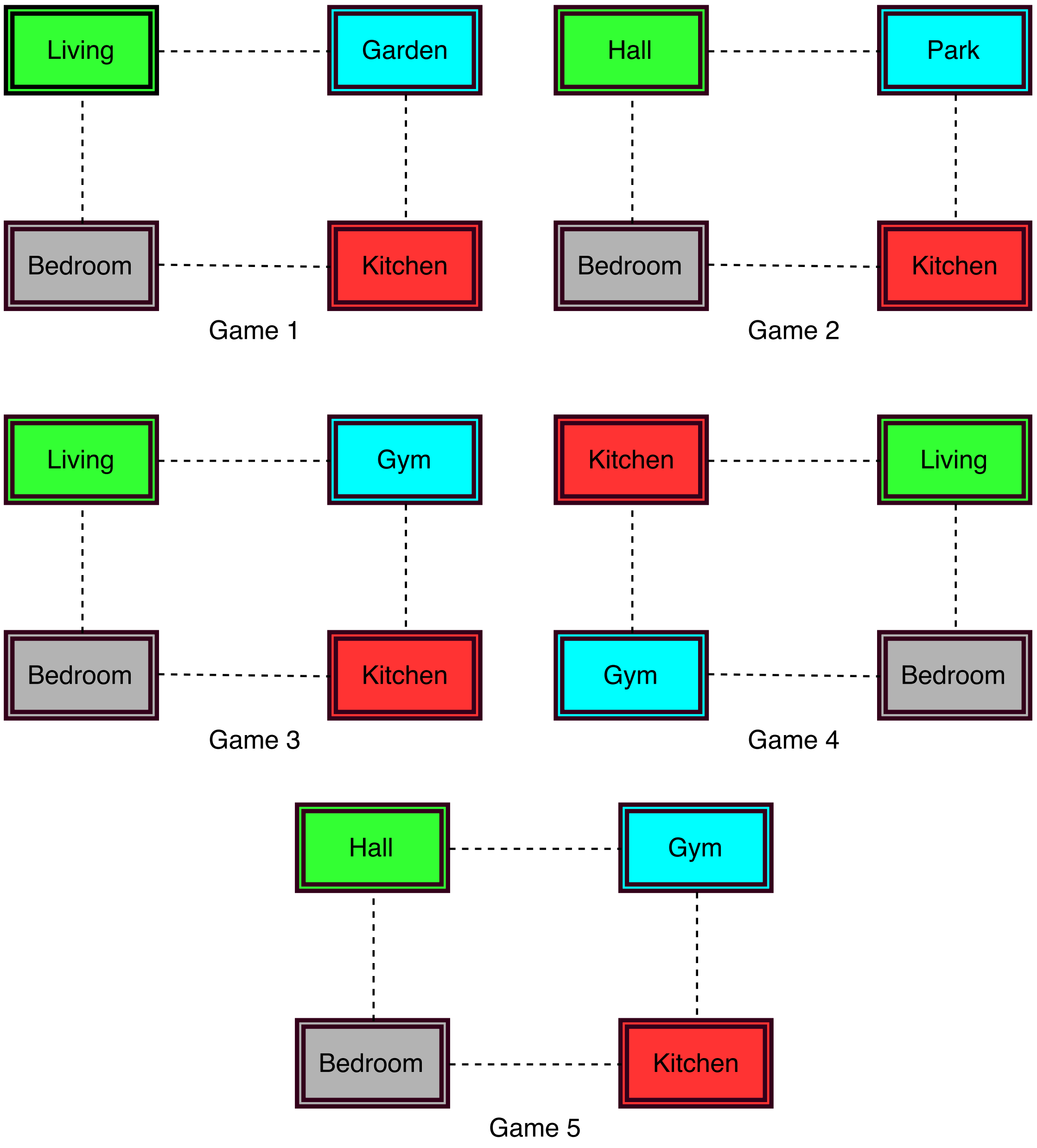


Figure 4.1: Different game layouts

4.1 Game Environment

We conduct experiments on 5 worlds which we have built upon the *Home World* created by Narasimhan et al. (2015)[1]. For our vocabulary expansion experiments, we have created multiple worlds, such that no two of the worlds have identical vocabulary. Out

[1]using Evennia (`http://www.evennia.com/`)

of the 5 worlds we created, only game-4 has a different layout. The games have a vocabulary of 90 words on an average. Every room has a variable set of textual descriptions among which one is randomly provided to the agent on entering that room. The quests (tasks given to the agent) and room descriptions are structured in a similar fashion to Narasimhan et al. (2015) in order to constrain an agent to understand the underlying information from the state so as to achieve higher rewards.

Each of the 5 worlds consists of four rooms – a bedroom, a living room, a garden, and a kitchen – that can be connected in different ways as shown in Figure 4.1. Each room has an object that the player can interact with. The player has to figure out the right room and the object to interact with based upon the given quest. "eat pizza", "go west" etc. are few examples of possible interactions of the player with the environment. There is no stochasticity in transitions between the rooms. The start states are random and the player can spawn into any random room with a random quest provided to it. The state representation that the player has, contains a textual description of its current state and quest. The action set and quests that are given to the agent in all 5 games are the same. We have also added alternate descriptions for all the rooms in different games.

We use the same evaluation metrics and procedure as used in Narasimhan et al. (2015). To measure an agent's performance we use the cumulative reward acquired per episode averaged over multiple episodes and the fraction of episodes in which the agent was able to complete the quest within 20 steps during evaluation. The best value for average reward a player can get is 0.98, since each step incurs a penalty of -0.01. In the worst case scenario the goal room can be diagonally opposite to the start room and so in order to obtain the best score the agent should be completing the given task within 2 to 3 steps without selecting any wrong command. Giving a negative reward for every step also enables the agent to optimize the number of steps.

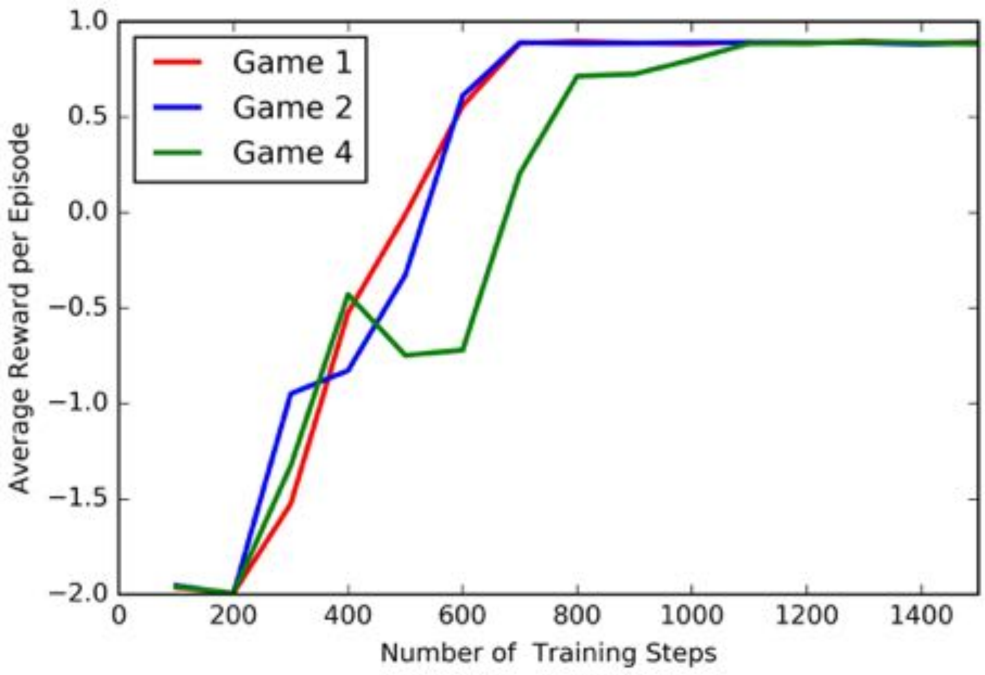
4.2 Learning games with contradicting state dynamics

In this section we analyze the performance of multi-task policy distillation agent (student) when the game state dynamics are contradicting. We use expert trajectories from games $1, 2, 4$ as teachers to our student network. Game $1, 2$ have same layout and game 4 has a different layout.

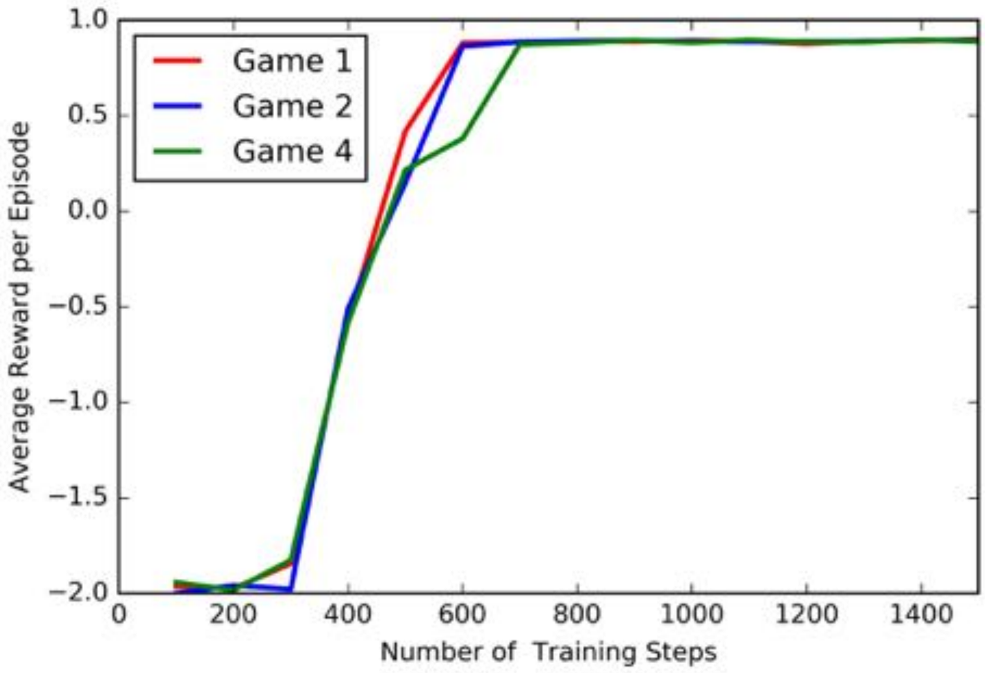
In Figure 4.2 we see that after around 1000 training steps, the student is not only able to complete 100% of the tasks given to it but additionally it also is able to get the best average reward possible on all the 3 games. Note that the size of the 1^{st} linear layer for the teacher networks is 100 which is double of that of the smaller student network used here.

An interesting observation that can be made from Figure 4.2a is that, initially for all the games the average reward increased but around 400 training steps the average reward for the game 4 started decreasing and after a while it started increasing again. One reason for this we suspect is that, the layout of game 4 is different compared to the other games, and thus the optimal policies will also be different. Due to this factor, an update in the network weights in the direction of game $1, 2$ could negatively affect it's performance on game 4. To strengthen our belief, we performed the same experiments with doubled size of the 1^{st} Linear layer which can be found in the Figure 4.2b.

From the training curves for Game 4 in Figure 4.2b, we observe that the learning is much more steadier and faster compared to the smaller student network experiments. Making the network wider enhances the modelling capability of the student network. The results in Figure 4.2 indicate that, with more modelling power, the student can learn much faster on contrasting domains.



(a) Student with 1st linear layer size as 50



(b) Student with 1st linear layer size as 100

Figure 4.2: Training curves of multi-task policy distillation agent (student)

4.3 What is the mechanics of multi-task policy distillation?

Intrigued by the positive results obtained on simply widening a single layer in the student network, we went a step further to interpret the intricacies of the student network. In this experiment, we analyze how inputs from different games affect different parts of the student network. In order to do this, we compare jacobians evaluated for different combinations of network layers for different games. 100 states are sampled from a single game at once, and the jacobians are calculated with these states as inputs to the network. Subsequently, we take an average over the jacobians evaluated over the 100 sampled states. This process is repeated for the next game and once we have the average jacobian matrices for the required games, we normalize the corresponding jacobian matrices and then scale the absolute values of elements in the resultant matrices to 255. Final heat-maps are generated for the scaled matrices and can be seen in the Figure 4.3 & 4.4.

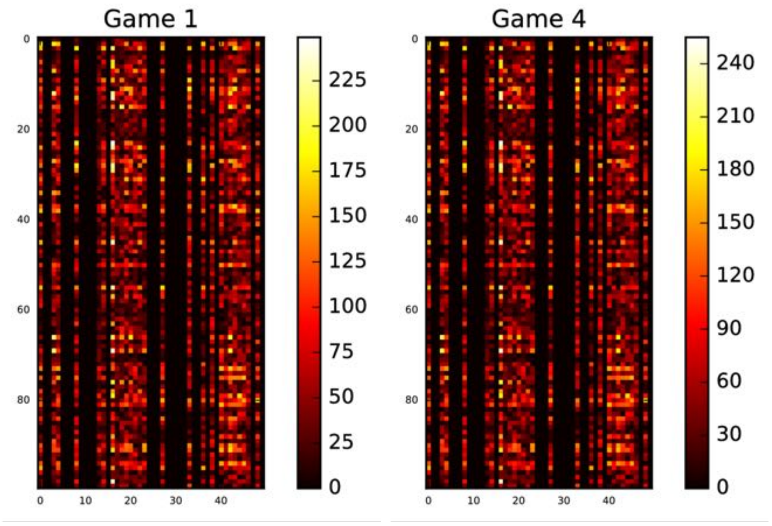
We have chosen the following combinations of layers for evaluating the jacobians for a game i, where $i \in \{1, 4\}$:

- combination 1: jacobian of ReLU layer w.r.t mean-pool layer outputs
- combination 2: jacobian of i^{th} action value layer w.r.t ReLU layer
- combination 3: jacobian of i^{th} object value layer w.r.t ReLU layer

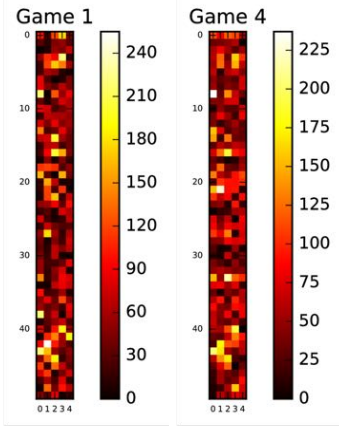
In both Figure 4.3 & 4.4 we compare the heat-maps between game 1 & 4. We see that in case of both the smaller and larger student network, the heat-maps for combination 1 are very similar where as most of the contrast can be visualized only in the heat-maps for combination 2 & 3. In the Table 4.1 we numerically quantify the similarities and dissimilarities between the heat-maps. For this we take an average of the absolute values of difference between the maps for game 1 & 4 for different combinations of network layers.

Table 4.1: Average absolute difference between heat-maps of game 1, 4 for different combinations of network layers using the two different sized versions of student network

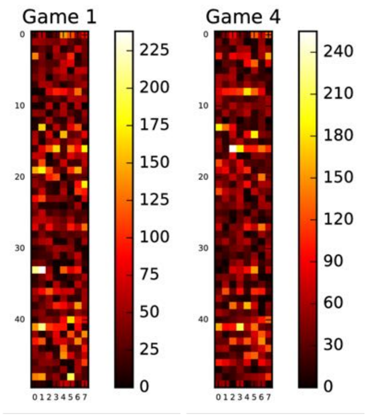
Layer Combination	1st linear layer size 50	1st linear layer size 100
ReLU vs mean-pool	23.58	23.11
Action value vs ReLU	138.02	123.56
Object value vs ReLU	124.43	129.53



(a) ReLU layer vs mean-pool layer

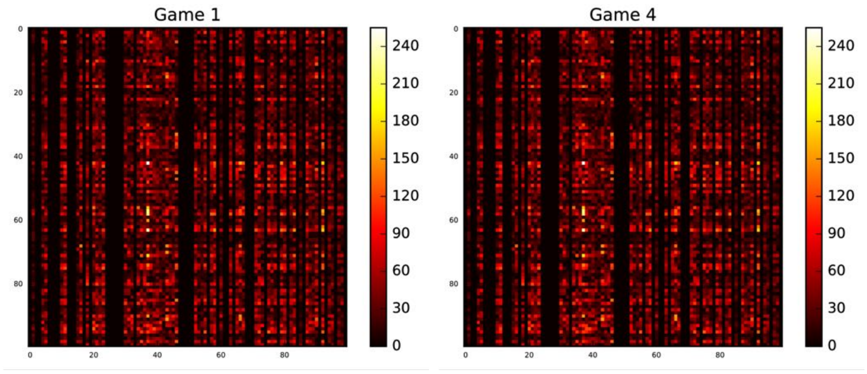


(b) Action value layer vs ReLU layer

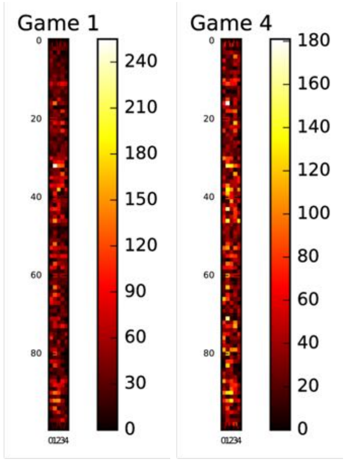


(c) Object value layer vs ReLU layer

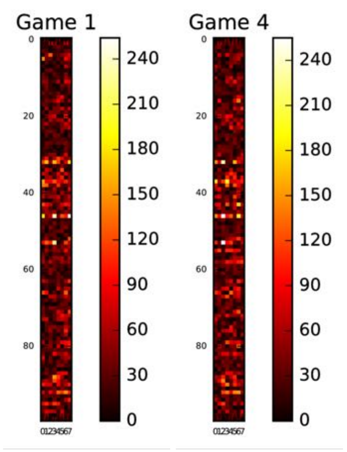
Figure 4.3: Heat-maps of different layer combinations for **smaller** student network trained on game 1, 2, 4 with size of 1st linear layer as 50. The heat-maps in (a) are very similar for the two games. All the diversity can be seen in the heat-maps in (b) & (c)



(a) ReLU layer vs mean-pool layer



(b) Action value layer vs ReLU layer



(c) Object value layer vs ReLU layer

Figure 4.4: Heat-maps of different layer combinations for **larger** student network trained on game $1, 2, 4$ with size of 1^{st} linear layer as 100. The heat-maps in (a) are very similar for the two games. All the diversity can be seen in the heat-maps in (b) & (c)

These results indicate that the learning in the multi-task policy distillation is two-fold. Firstly, a part of the network, the portion between input and ReLU Layer, is specializing to learning something universal and generalizable from different game sources. Secondly, the portions of network between the common ReLU layer and different output layers are learning the corresponding game specific information. This especially is the reason why making this part of the network deeper or wider will help get better performance on the complex games as can be seen in Figure 4.2.

4.4 Transfer learning using policy distillation

Going a step further in comparing the utility of representations learnt by the student with that of a single game teacher, we created game 5 which has parts of its vocabulary common with each of the games 1, 2, 3. We used single game teachers T_1, T_2, T_3 trained on games 1, 2, 3 respectively. The student which we will denote by S is trained using these expert teachers T_1, T_2, T_3. For this experiment we take the Game 5 and train 4 agents A_1, A_2, A_3, A_4 by initializing the embeddings of possible words with the embeddings learnt by T_1, T_2, T_3 and S respectively. Here, by possible words we mean those words that are common between the already learnt network's game source and game-5. The embeddings for the words that are not common are initialized randomly just as in the case of a normal LSTM-DQN teacher.

We also train two more agents, one is A_5 that has all the word embeddings initialized randomly, and other is A_6 which has all the possible word embeddings from game-5 initialized using all possible embeddings from teachers T_1, T_2, T_3 at the same time. In the case of A_6 when there is more than one possible embedding source for a word, the source is chosen randomly. Agent A_5 an A_6 serve as baselines for this experiment.

From the Figure 4.5 we infer that the teacher A_4 that had embeddings initialized from the student S is able to learn much faster than the others. Most importantly, we see that it performs better than A_6, which is the usual method for language expansion. This strengthens our belief that student is able to learn useful representations of language from multiple game sources.

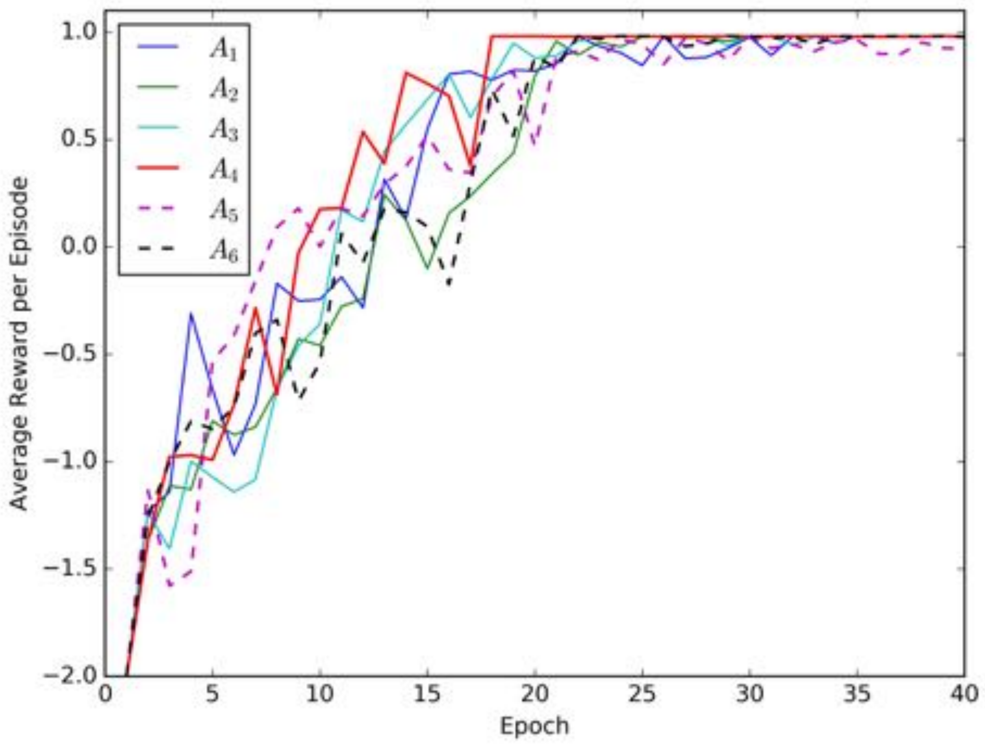


Figure 4.5: Training curves for LSTM-DQN agents A_1, A_2, A_3, A_4, A_5, A_6 training on game 5. Agent A_1, A_2, A_3, are initialized with learnt embeddings from LSTM-DQN teachers T_1, T_2, T_3 respectively. Agent A_4 is initialized with learnt embeddings from student S. Agent A_5 has randomly initialized embeddings. Agent A_6 has all possible embeddings initialized from all the LSTM-DQN teachers T_1, T_2, T_3 combined.

4.5 What can we say from the visualization of word embeddings?

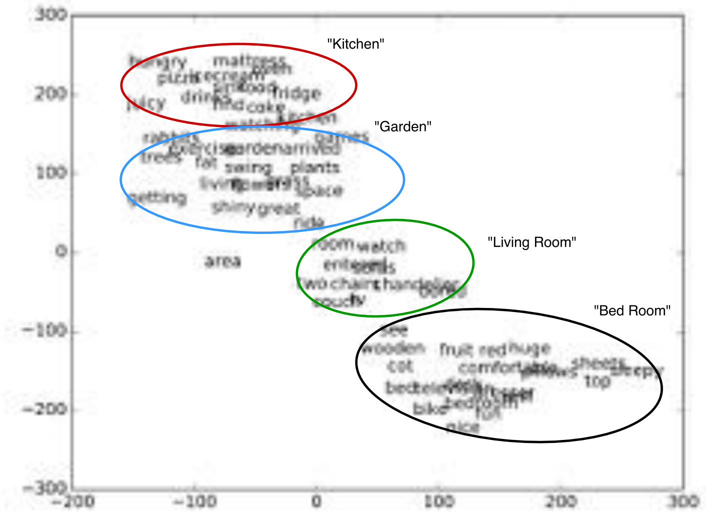
In this section we analyze the t-SNE (Maaten and Hinton, 2008) embeddings learnt by the multi-task policy distillation method on learning games 1, 2 and 3 combined and compare them with the embeddings learnt by corresponding single game teachers. We train a new student network for this experiment. Games 1, 2, and 3 all have the same layout but have different vocabularies. For this we first passed every word in the vocabulary of a game through the LSTM layer and took its output and then projected it onto a 2 dimensional space using t-SNE algorithm (Maaten and Hinton, 2008) and plotted the representations. This was done for all the teacher networks and the student network. In this experiment we used a student network of same size as that of corresponding teacher networks. From the Figure 4.6 and Figure 4.7, we can infer that the word embeddings learnt by a single student network are much more compact and better clustered as compared to those learnt by multiple single game teachers. Also, the number of outliers in the clusters are much less in the case of student embeddings as can be seen in the Figure 4.6.

We believe that the student network is able to associate words with the rooms and the tasks (quests) better than the teacher networks, by benefiting from textual descriptions from different games.

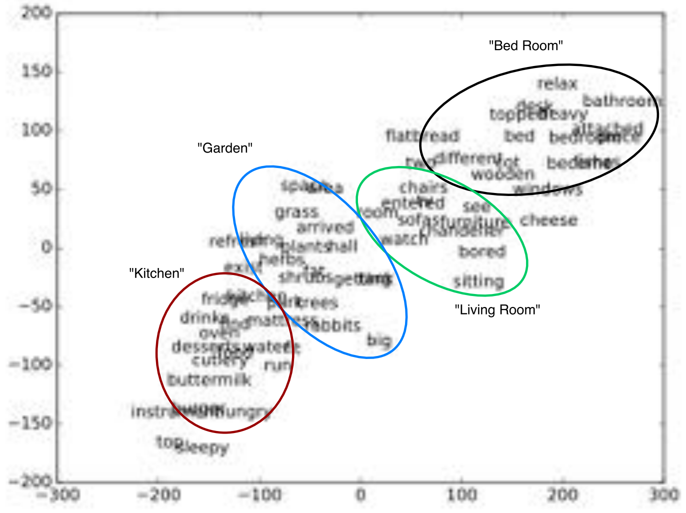
One more factor that could enable student to learn better language representations is that, in the process of policy distillation only the final optimal policies of the teacher are learnt by the student network. These stable targets to the student network could help the network to learn better, unlike the case of LSTM-DQN teachers training where the network has to represent a series of policies that are obtained in the policy iteration cycle of the Q-learning Algorithm.

4.6 Policy distillation Vs. Multi-Task LSTM-DQN?

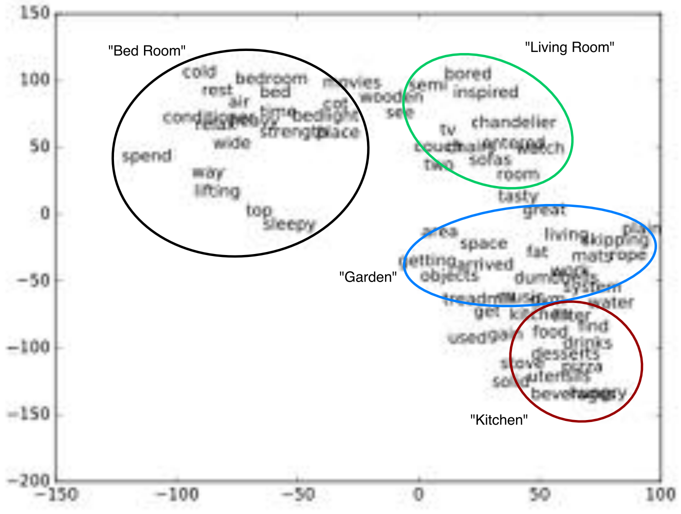
In this section we analyze the performance of the multi-task LSTM-DQN. We trained a multi-task LSTM-DQN agent on games 1, 2, 3. The performance of this method is very bad as compared to the multi-task policy distillation method as can be seen in Figure 4.8a. The multi task LSTM-DQN agent is able to solve only about 30% of the quests on an average, given that the episodes are ended after 20 steps, whereas multi-task policy distillation was able to solve 100% of the quests.



(a) game 1

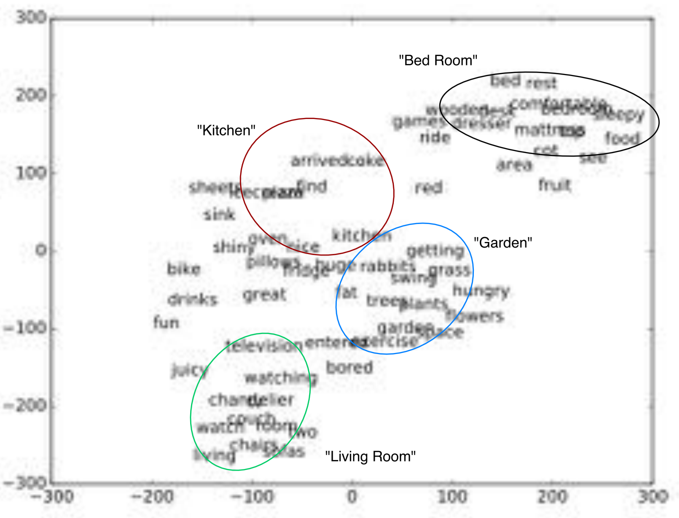


(b) game 2

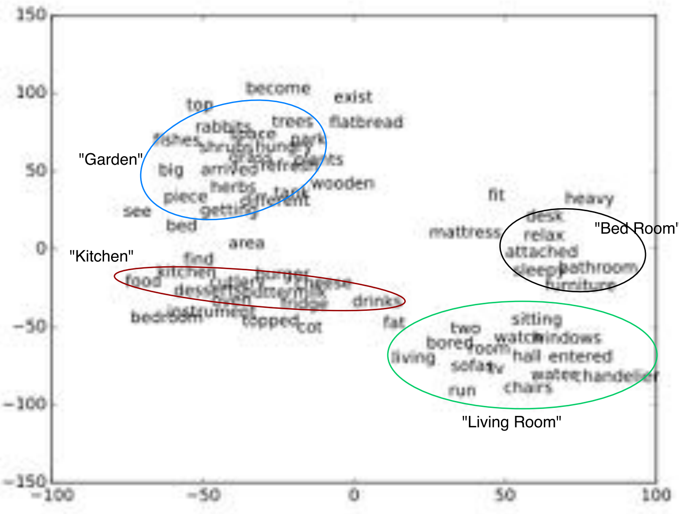


(c) game 3

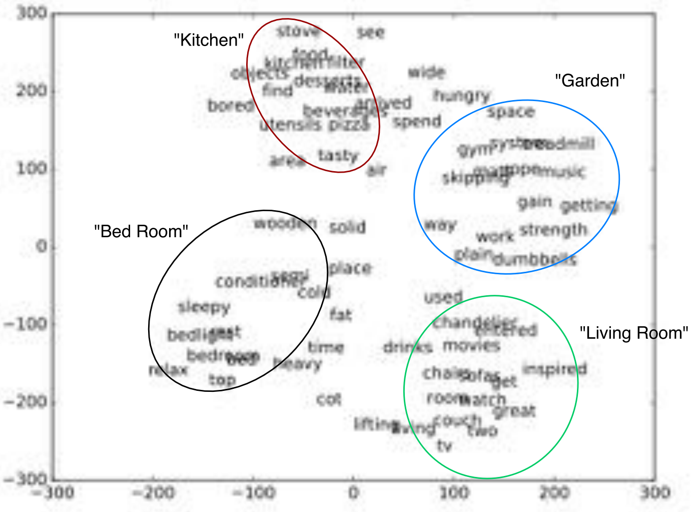
Figure 4.6: t-SNE plots of embeddings learnt by multi-task distillation agent (student) trained on games 1, 2, 3



(a) game 1

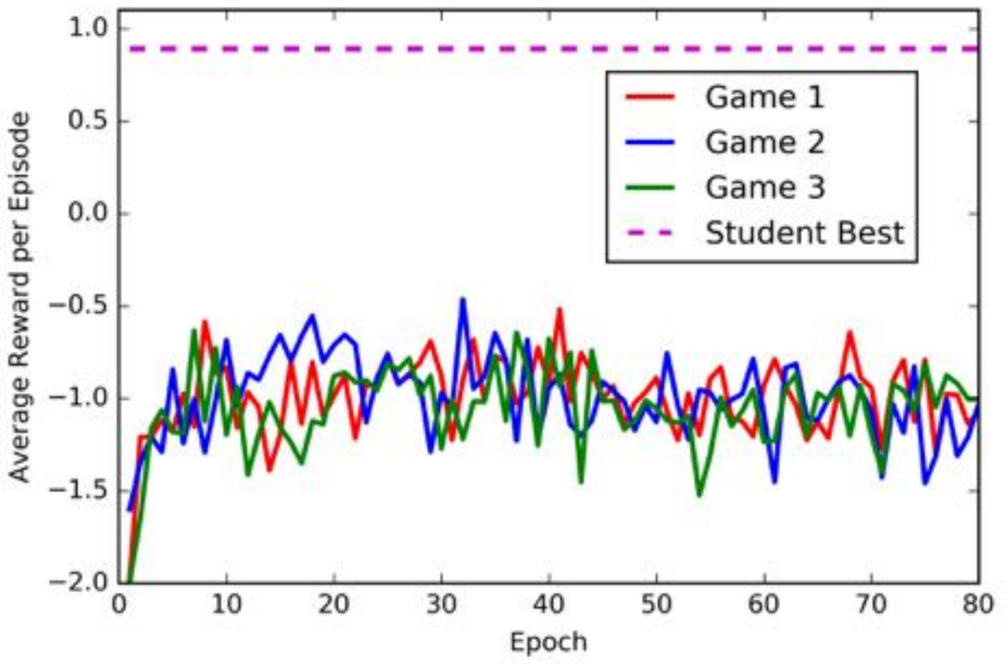


(b) game 2

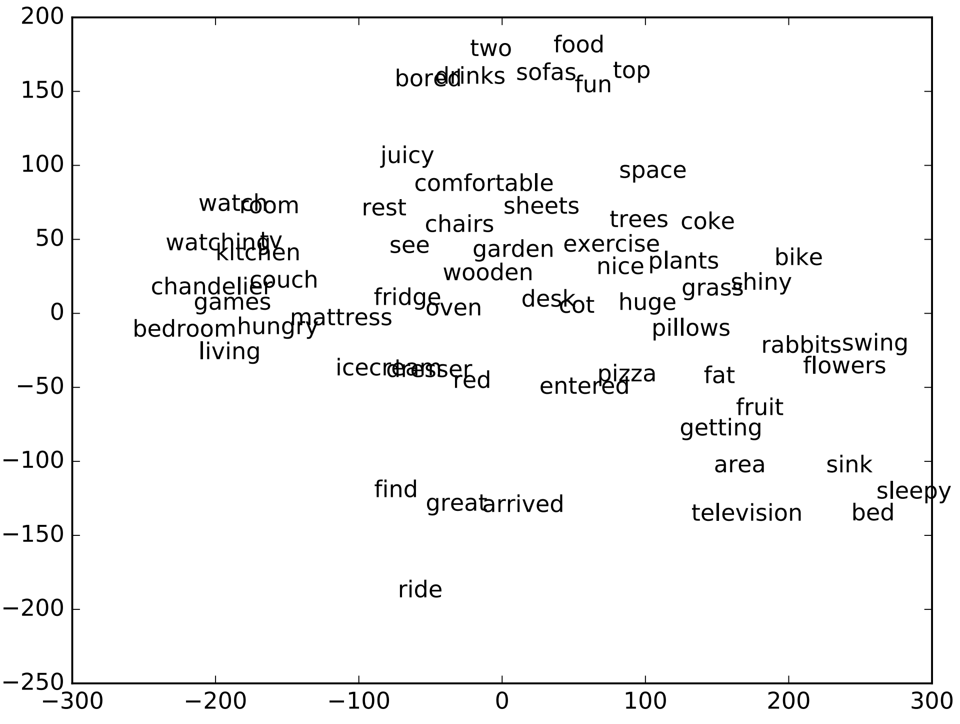


(c) game 3

Figure 4.7: t-SNE plots of embeddings learnt by LSTM-DQN teachers trained on games 1, 2, 3 respectively



(a) Training curves of multi-task LSTM-DQN



(b) t-SNE plot of the multi-task LSTM-DQN word embeddings on game 1

Figure 4.8: Experimental results of multi-task LSTM-DQN

Analysis of the embeddings learnt by the agent for words in vocabulary of games $1, 2, 3$ showed that the embeddings learnt by this method are almost random. This could happen because the errors are not propagating steadily throughout network, as the policy that the agent is learning keeps changing due to the policy iteration cycle of the Q-Learning algorithm (Watkins and Dayan, 1992). Since, there are 3 sets of games we are learning here, the effective targets to the network are even more unstable.

Chapter 5

Conclusion

In this work, we applied the recently proposed policy distillation method (Rusu et al., 2015) to text-based games. Our experiments verify that policy distillation is an effective approach even for LSTM-DQN where each game might have different vocabulary. We also provided an extensive analysis on how distillation works when games have contradicting state dynamics. It will be interesting to see if the same analysis will hold for games with larger vocabulary space. A crucial problem we faced is that evaluating the closeness of learned embeddings to that of human based learning is difficult, and there is no globally accepted evaluation metric and standard metrics do not focus on this scenario. We believe that better evaluation metrics are needed to accelerate the progress in this direction. Finally, we believe that our work is an important step in the direction of designing agents that can expand their language in a completely online fashion (aka continual learning).

LIST OF PAPERS BASED ON THESIS

1. Ghulam Ahmed Ansari, Sagar J P, Sarath Chandar, Balaraman Ravindran "Language Expansion In Text-Based Games", *Deep Reinforcement Learning Workshop, Neural Information Processing Systems (NIPS)*, 2016 (Accepted).
2. Ghulam Ahmed Ansari, Sagar J P, Sarath Chandar, Balaraman Ravindran "Language Expansion In Text-Based Games", *Emperical Methods In Natural Language Processing (EMNLP)*, 2017 (Submitted).

Bibliography

Ross B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015. URL http://arxiv.org/abs/1504.08083.

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.

George Dahl, Dong Yu, Li Deng, and Alex Acero. Context-dependent pre-trained deep neural networks for large vocabulary speech recognition. *Audio, Speech, and Language Processing, IEEE Transactions on*, (99):1–1, 2010.

G. Tesauro. Temporal difference learning and TD-Gammon. *Communications of the ACM*, 38(3):58–68, 1995.

Long-Ji Lin. *Reinforcement Learning for Robots Using Neural Networks*. PhD thesis, Pittsburgh, PA, USA, 1992. UMI Order No. GAX93-22750.

Yann Lecun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553): 436–444, 5 2015. ISSN 0028-0836. doi: 10.1038/nature14539.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets.

In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014. URL http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

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. doi: 10.1038/nature16961.

Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, AdriàPuigdomènech Badia, Karl Moritz Hermann, Yori Zwols, Georg Ostrovski, Adam Cain, Helen King, Christopher Summerfield, Phil Blunsom, Koray Kavukcuoglu, and Demis Hassabis. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, 10 2016. URL http://dx.doi.org/10.1038/nature20101.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy P Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, 2016.

Andrei A. Rusu, Sergio Gomez Colmenarejo, Çaglar Gülçehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation. *CoRR*, abs/1511.06295, 2015. URL `http://arxiv.org/abs/1511.06295`.

Karthik Narasimhan, Tejas D. Kulkarni, and Regina Barzilay. Language understanding for text-based games using deep reinforcement learning. *CoRR*, abs/1506.08941, 2015. URL `http://arxiv.org/abs/1506.08941`.

Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4): 279–292, 1992.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning : An Introduction*. MIT Press, 1998.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008.

Ji He, Mari Ostendorf, Xiaodong He, Jianshu Chen, Jianfeng Gao, Lihong Li, and Li Deng. Deep reinforcement learning with a combinatorial action space for predicting and tracking popular discussion threads. *arXiv preprint arXiv:1606.03667*, 2016.

Rodrigo Nogueira and Kyunghyun Cho. End-to-end goal-driven web navigation. *arXiv preprint arXiv:1602.02261*, 2016.

Sainbayar Sukhbaatar, Arthur Szlam, Gabriel Synnaeve, Soumith Chintala, and Rob Fergus. Mazebase: A sandbox for learning from games. *arXiv preprint arXiv:1511.07401*, 2015.

Kumpati S Narendra and Kannan Parthasarathy. Identification and control of dynamical systems using neural networks. *IEEE Transactions on neural networks*, 1(1):4–27, 1990.