Language Understanding for Text-based Games using Deep Reinforcement Learning

Karthik Narasimhan[*]
CSAIL, MIT
karthikn@csail.mit.edu

Tejas D Kulkarni[*]
CSAIL, BCS, MIT
tejask@mit.edu

Regina Barzilay
CSAIL, MIT
regina@csail.mit.edu

Abstract

In this paper, we consider the task of learning control policies for text-based games. In these games, all interactions in the virtual world are through text and the underlying state is not observed. The resulting language barrier makes such environments challenging for automatic game players. We employ a deep reinforcement learning framework to jointly learn state representations and action policies using game rewards as feedback. This framework enables us to map text descriptions into vector representations that capture the semantics of the game states. We evaluate our approach on two game worlds, comparing against baselines using bag-of-words and bag-of-bigrams for state representations. Our algorithm outperforms the baselines on both worlds demonstrating the importance of learning expressive representations. [1]

State 1: The old bridge
You are standing very close to the bridge's eastern foundation. If you go east you will be back on solid ground ... The bridge sways in the wind.

Command: **Go east**

State 2: Ruined gatehouse
The old gatehouse is near collapse. Part of its northern wall has already fallen down ... East of the gatehouse leads out to a small open area surrounded by the remains of the castle. There is also a standing archway offering passage to a path along the old southern inner wall.
Exits: Standing archway, castle corner, Bridge over the abyss

Figure 1: Sample gameplay from a Fantasy World. The player with the quest of finding a secret tomb, is currently located on an *old bridge*. She then chooses an action to *go east* that brings her to a *ruined gatehouse* (State 2).

1 Introduction

In this paper, we address the task of learning control policies for text-based strategy games. These games, predecessors to modern graphical ones, still enjoy a large following worldwide.[2] They often involve complex worlds with rich interactions and elaborate textual descriptions of the underlying states (see Figure 1). Players read descriptions of the current world state and respond with natural language commands to take actions. Since the underlying state is not directly observable, the player has to understand the text in order to act, making it challenging for existing AI programs to play these games (DePristo and Zubek, 2001).

In designing an autonomous game player, we have considerable latitude when selecting an adequate state representation to use. The simplest method is to use a bag-of-words representation derived from the text description. However, this scheme disregards the ordering of words and the finer nuances of meaning that evolve from composing words into sentences and paragraphs. For instance, in State 2 in Figure 1, the agent has to understand that going *east* will lead it to the castle whereas moving *south* will take it to the standing archway. An alternative approach is to convert text descriptions to pre-specified representations using annotated training data, commonly used in

[*]Both authors contributed equally to this work.
[1]Code is available at http://people.csail.mit.edu/karthikn/mud-play.
[2]http://mudstats.com/

Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pages 1–11,
Lisbon, Portugal, 17-21 September 2015.

language grounding tasks (Matuszek et al., 2013; Kushman et al., 2014).

In contrast, our goal is to learn useful representations in conjunction with control policies. We adopt a reinforcement learning framework and formulate game sequences as Markov Decision Processes. An agent playing the game aims to maximize rewards that it obtains from the game engine upon the occurrence of certain events. The agent learns a policy in the form of an action-value function $Q(s, a)$ which denotes the long-term merit of an action a in state s.

The action-value function is parametrized using a deep recurrent neural network, trained using the game feedback. The network contains two modules. The first one converts textual descriptions into vector representations that act as proxies for states. This component is implemented using Long Short-Term Memory (LSTM) networks (Hochreiter and Schmidhuber, 1997). The second module of the network scores the actions given the vector representation computed by the first.

We evaluate our model using two Multi-User Dungeon (MUD) games (Curtis, 1992; Amir and Doyle, 2002). The first game is designed to provide a controlled setup for the task, while the second is a publicly available one and contains human generated text descriptions with significant language variability. We compare our algorithm against baselines of a random player and models that use bag-of-words or bag-of-bigrams representations for a state. We demonstrate that our model LSTM-DQN significantly outperforms the baselines in terms of number of completed quests and accumulated rewards. For instance, on a fantasy MUD game, our model learns to complete 96% of the quests, while the bag-of-words model and a random baseline solve only 82% and 5% of the quests, respectively. Moreover, we show that the acquired representation can be reused across games, speeding up learning and leading to faster convergence of Q-values.

2 Related Work

Learning control policies from text is gaining increasing interest in the NLP community. Example applications include interpreting help documentation for software (Branavan et al., 2010), navigating with directions (Vogel and Jurafsky, 2010; Kollar et al., 2010; Artzi and Zettlemoyer, 2013; Matuszek et al., 2013; Andreas and Klein, 2015) and playing computer games (Eisenstein et al., 2009; Branavan et al., 2011a).

Games provide a rich domain for grounded language analysis. Prior work has assumed perfect knowledge of the underlying state of the game to learn policies. Gorniak and Roy (2005) developed a game character that can be controlled by spoken instructions adaptable to the game situation. The grounding of commands to actions is learned from a transcript manually annotated with actions and state attributes. Eisenstein et al. (2009) learn game rules by analyzing a collection of game-related documents and precompiled traces of the game. In contrast to the above work, our model combines text interpretation and strategy learning in a single framework. As a result, textual analysis is guided by the received control feedback, and the learned strategy directly builds on the text interpretation.

Our work closely relates to an automatic game player that utilizes text manuals to learn strategies for Civilization (Branavan et al., 2011a). Similar to our approach, text analysis and control strategies are learned jointly using feedback provided by the game simulation. In their setup, states are fully observable, and the model learns a strategy by combining state/action features and features extracted from text. However, in our application, the state representation is not provided, but has to be inferred from a textual description. Therefore, it is not sufficient to extract features from text to supplement a simulation-based player.

Another related line of work consists of automatic video game players that infer state representations directly from raw pixels (Koutník et al., 2013; Mnih et al., 2015). For instance, Mnih et al. (2015) learn control strategies using convolutional neural networks, trained with a variant of Q-learning (Watkins and Dayan, 1992). While both approaches use deep reinforcement learning for training, our work has important differences. In order to handle the sequential nature of text, we use Long Short-Term Memory networks to automatically learn useful representations for arbitrary text descriptions. Additionally, we show that decomposing the network into a representation layer and an action selector is useful for transferring the learnt representations to new game scenarios.

3 Background

Game Representation We represent a game by the tuple $\langle H, A, T, R, \Psi \rangle$, where H is the set of

all possible game states, $A = \{(a, o)\}$ is the set of all commands (action-object pairs), $T(h' \mid h, a, o)$ is the stochastic transition function between states and $R(h, a, o)$ is the reward function. The game state H is *hidden* from the player, who only receives a varying textual description, produced by a stochastic function $\Psi : H \rightarrow S$. Specifically, the underlying state h in the game engine keeps track of attributes such as the player's location, her health points, time of day, etc. The function Ψ (also part of the game framework) then converts this state into a textual description of the location the player is at or a message indicating low health. We do not assume access to either H or Ψ for our agent during both training and testing phases of our experiments. We denote the space of all possible text descriptions s to be S. Rewards are generated using R and are only given to the player upon completion of in-game quests.

Q-Learning Reinforcement Learning is a commonly used framework for learning control policies in game environments (Silver et al., 2007; Amato and Shani, 2010; Branavan et al., 2011b; Szita, 2012). The game environment can be formulated as a sequence of state transitions (s, a, r, s') of a Markov Decision Process (MDP). The agent takes an action a in state s by consulting a state-action value function $Q(s, a)$, which is a measure of the action's expected long-term reward. Q-Learning (Watkins and Dayan, 1992) is a model-free technique which is used to learn an optimal $Q(s, a)$ for the agent. Starting from a random Q-function, the agent continuously updates its Q-values by playing the game and obtaining rewards. The iterative updates are derived from the Bellman equation (Sutton and Barto, 1998):

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

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.

Using these evolving Q-values, the agent chooses the action with the highest $Q(s, a)$ to maximize its expected future rewards. In practice, the trade-off between exploration and exploitation can be achieved following an ϵ-greedy policy (Sutton and Barto, 1998), where the agent performs a random action with probability ϵ.

Deep Q-Network In large games, it is often impractical to maintain the Q-value for all possible state-action pairs. One solution to this problem is to approximate $Q(s, a)$ using a parametrized function $Q(s, a; \theta)$, which can generalize over states and actions by considering higher-level attributes (Sutton and Barto, 1998; Branavan et al., 2011a). 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. The non-linear function layers of the DQN also enable it to learn better value functions than linear approximators.

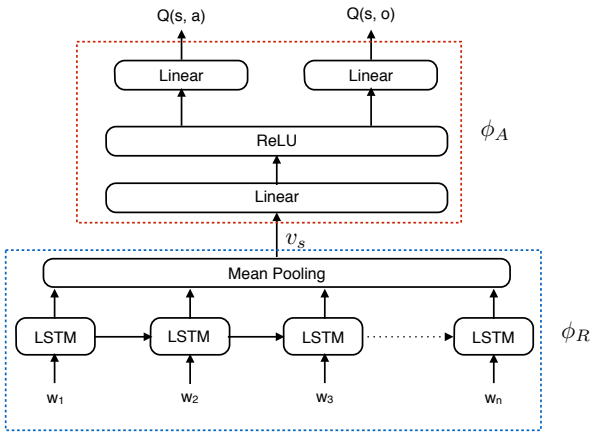


Figure 2: Architecture of LSTM-DQN: The Representation Generator (ϕ_R) (bottom) takes as input a stream of words observed in state s and produces a vector representation v_s, which is fed into the action scorer (ϕ_A) (top) to produce scores for all actions and argument objects.

4 Learning Representations and Control Policies

In this section, we describe our model (DQN) and describe its use in learning good Q-value approximations for games with stochastic textual descriptions. We divide our model into two parts. The first module is a *representation generator* that converts the textual description of the current state into a vector. This vector is then input into the second module which is an *action scorer*. Figure 2 shows the overall architecture of our model. We learn the parameters of both the representation generator and the action scorer jointly, using the in-game reward feedback.

Representation Generator (ϕ_R) The representation generator reads raw text displayed to the agent and converts it to a vector representation v_s. A bag-of-words (BOW) representation is not sufficient to capture higher-order structures of sentences and paragraphs. The need for a better semantic representation of the text is evident from the average performance of this representation in playing MUD-games (as we show in Section 6).

In order to assimilate better representations, we utilize a Long Short-Term Memory network (LSTM) (Hochreiter and Schmidhuber, 1997) as a representation generator. LSTMs are recurrent neural networks with the ability to connect and recognize long-range patterns between words in text. They are more robust than BOW to small variations in word usage and are able to capture underlying semantics of sentences to some extent. In recent work, LSTMs have been used successfully in NLP tasks such as machine translation (Sutskever et al., 2014) and sentiment analysis (Tai et al., 2015) to compose vector representations of sentences from word-level embeddings (Mikolov et al., 2013; Pennington et al., 2014). In our setup, the LSTM network takes in word embeddings w_k from the words in a description s and produces output vectors x_k at each step.

To get the final state representation v_s, we add a *mean pooling* layer which computes the element-wise mean over the output vectors x_k.[3]

$$v_s = \frac{1}{n}\sum_{k=1}^{n} x_k \tag{2}$$

Action Scorer (ϕ_A) The action scorer module produces scores for the set of possible actions given the current state representation. We use a multi-layered neural network for this purpose (see Figure 2). The input to this module is the vector from the representation generator, $v_s = \phi_R(s)$ and the outputs are scores for actions $a \in A$. Scores for all actions are predicted simultaneously, which is computationally more efficient than scoring each state-action pair separately. Thus, by combining the representation generator and action scorer, we can obtain the approximation for the Q-function as $Q(s,a) \approx \phi_A(\phi_R(s))[a]$.

An additional complexity in playing MUD-games is that the actions taken by the player are multi-word natural language *commands* such as *eat apple* or *go east*. Due to computational constraints, in this work we limit ourselves to consider commands to consist of one action (e.g. *eat*) and one argument object (e.g. *apple*). This assumption holds for the majority of the commands in our worlds, with the exception of one class of commands that require two arguments (e.g. *move red-root right*, *move blue-root up*). We consider all possible actions and objects available in the game and predict both for each state using the same network (Figure 2). We consider the Q-value of the entire command (a, o) to be the average of the Q-values of the action a and the object o. For the rest of this section, we only show equations for $Q(s,a)$ but similar ones hold for $Q(s,o)$.

Parameter Learning We learn the parameters θ_R of the representation generator and θ_A of the action scorer using stochastic gradient descent with *RMSprop* (Tieleman and Hinton, 2012). The complete training procedure is shown in Algorithm 1. In each iteration i, we update the parameters to reduce the discrepancy between the predicted value of the current state $Q(s_t, a_t; \theta_i)$ (where $\theta_i = [\theta_R; \theta_A]_i$) and the expected Q-value given the reward r_t and the value of the next state $\max_a Q(s_{t+1}, a; \theta_{i-1})$.

We keep track of the agent's previous experiences in a *memory* $\mathcal{D}$.[4] Instead of performing updates to the Q-value using transitions from the current episode, we sample a random transition $(\hat{s}, \hat{a}, s', r)$ from $\mathcal{D}$. Updating the parameters in this way avoids issues due to strong correlation when using transitions of the same episode (Mnih et al., 2015). Using the sampled transition and (1), we obtain the following loss function to minimize:

$$\mathcal{L}_i(\theta_i) = \mathrm{E}_{\hat{s},\hat{a}}[(y_i - Q(\hat{s}, \hat{a}; \theta_i))^2] \tag{3}$$

where $y_i = \mathrm{E}_{\hat{s},\hat{a}}[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) \mid \hat{s}, \hat{a}]$ is the target Q-value with parameters θ_{i-1} fixed from the previous iteration.

The updates on the parameters θ can be performed using the following gradient of $\mathcal{L}_i(\theta_i)$:

$$\nabla_{\theta_i}\mathcal{L}_i(\theta_i) = \mathrm{E}_{\hat{s},\hat{a}}[2(y_i - Q(\hat{s}, \hat{a}; \theta_i))\nabla_{\theta_i} Q(\hat{s}, \hat{a}; \theta_i)]$$

For each epoch of training, the agent plays several episodes of the game, which is restarted after every terminal state.

[3] We also experimented with considering just the output vector of the LSTM after processing the last word. Empirically, we find that mean pooling leads to faster learning, so we use it in all our experiments.

[4] The memory is limited and rewritten in a first-in-first-out (FIFO) fashion.

Algorithm 1 Training Procedure for DQN with prioritized sampling

Initialize experience memory $\mathcal{D}$
Initialize parameters of representation generator (ϕ_R) and action scorer (ϕ_A) randomly
for $episode = 1, M$ **do**
    Initialize game and get start state description s_1
    for $t = 1, T$ **do**
        Convert s_t (text) to representation v_{s_t} using ϕ_R
        if $random() < \epsilon$ **then**
            Select a random action a_t
        else
            Compute $Q(s_t, a)$ for all actions using $\phi_A(v_{s_t})$
            Select $a_t = \text{argmax}\ Q(s_t, a)$
        Execute action a_t and observe reward r_t and new state s_{t+1}
        Set priority $p_t = 1$ if $r_t > 0$, else $p_t = 0$
        Store transition $(s_t, a_t, r_t, s_{t+1}, p_t)$ in $\mathcal{D}$
        Sample random mini batch of transitions $(s_j, a_j, r_j, s_{j+1}, p_j)$ from $\mathcal{D}$, with fraction ρ having $p_j = 1$
        Set $y_j = \begin{cases} r_j & \text{if } s_{j+1} \text{ is terminal} \\ r_j + \gamma \max_{a'} Q(s_{j+1}, a'; \theta) & \text{if } s_{j+1} \text{ is non-terminal} \end{cases}$
        Perform gradient descent step on the loss $\mathcal{L}(\theta) = (y_j - Q(s_j, a_j; \theta))^2$

Mini-batch Sampling In practice, online updates to the parameters θ are performed over a mini batch of state transitions, instead of a single transition. This increases the number of experiences used per step and is also more efficient due to optimized matrix operations.

The simplest method to create these mini-batches from the experience memory $\mathcal{D}$ is to sample uniformly at random. However, certain experiences are more valuable than others for the agent to learn from. For instance, rare transitions that provide positive rewards can be used more often to learn optimal Q-values faster. In our experiments, we consider such positive-reward transitions to have higher *priority* and keep track of them in $\mathcal{D}$. We use *prioritized sampling* (inspired by Moore and Atkeson (1993)) to sample a fraction ρ of transitions from the higher priority pool and a fraction $1 - \rho$ from the rest.

5 Experimental Setup

Game Environment For our game environment, we modify Evennia,[5] an open-source library for building online textual MUD games. Evennia is a Python-based framework that allows one to easily create new games by writing a batch file describing the environment with details of rooms, objects and actions. The game engine keeps track of the game state internally, presenting textual descriptions to the player and receiving text commands from the player. We conduct experiments on two worlds - a smaller *Home world* we created ourselves, and a larger, more complex *Fantasy world* created by Evennia's developers. The motivation behind Home world is to abstract away high-level planning and focus on the language understanding requirements of the game.

Stats	Home World	Fantasy World
Vocabulary size	84	1340
Avg. words / description	10.5	65.21
Max descriptions / room	3	100
# diff. quest descriptions	12	-
State transitions	Deterministic	Stochastic
# states (underlying)	16	≥ 56
Branching factor (# commands / state)	40	222

Table 1: Various statistics of the two game worlds

Table 1 provides statistics of the game worlds. We observe that the Fantasy world is moderately sized with a vocabulary of 1340 words and up to 100 different descriptions for a room. These descriptions were created manually by the game developers. These diverse, engaging descriptions are designed to make it interesting and exciting for human players. Several rooms have many alternative descriptions, invoked randomly on each visit by

[5] http://www.evennia.com/

the player.

Comparatively, the Home world is smaller: it has a very restricted vocabulary of 84 words and the room descriptions are relatively structured. However, both the room descriptions (which are also varied and randomly provided to the agent) and the quest descriptions were adversarially created with negation and conjunction of facts to force an agent to actually understand the state in order to play well. Therefore, this domain provides an interesting challenge for language understanding.

In both worlds, the agent receives a positive reward on completing a quest, and negative rewards for getting into bad situations like falling off a bridge, or losing a battle. We also add small deterministic negative rewards for each non-terminating step. This incentivizes the agent to learn policies that solve quests in fewer steps. The supplementary material has details on the reward structure.

Home World We created *Home world* to mimic the environment of a typical house.[6] The world consists of four rooms - a living room, a bedroom, a kitchen and a garden with connecting pathways. Every room is reachable from every other room. Each room contains a representative object that the agent can interact with. For instance, the kitchen has an *apple* that the player can *eat*. Transitions between the rooms are deterministic. At the start of each game episode, the player is placed in a random room and provided with a randomly selected quest. The text provided to the player contains both the description of her current state and that of the quest. Thus, the player can begin in one of 16 different states (4 rooms $\times$ 4 quests), which adds to the world's complexity.

An example of a quest given to the player in text is *Not you are sleepy now but you are hungry now*. To complete this quest and obtain a reward, the player has to navigate through the house to reach the kitchen and eat the apple (i.e type in the command *eat apple*). More importantly, the player should interpret that the quest does not require her to take a nap in the bedroom. We created such misguiding quests to make it hard for agents to succeed without having an adequate level of language understanding.

[6] An illustration is provided in the supplementary material.

Fantasy World The Fantasy world is considerably more complex and involves quests such as navigating through a broken bridge or finding the secret tomb of an ancient hero. This game also has stochastic transitions in addition to varying state descriptions provided to the player. For instance, there is a possibility of the player falling from the bridge if she lingers too long on it.

Due to the large command space in this game,[7] we make use of cues provided by the game itself to narrow down the set of possible objects to consider in each state. For instance, in the MUD example in Figure 1, the game provides a list of possible exits. If the game does not provide such clues for the current state, we consider all objects in the game.

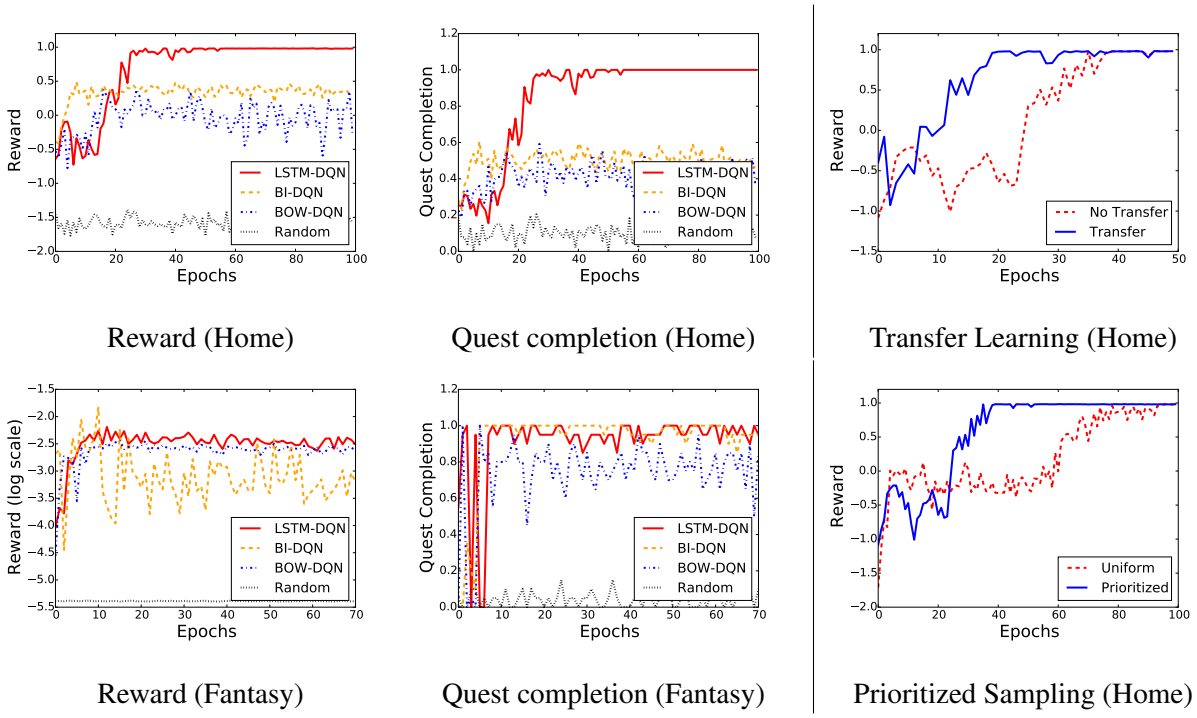
Evaluation We use two metrics for measuring an agent's performance: (1) the cumulative reward obtained per episode averaged over the episodes and (2) the fraction of quests completed by the agent. The evaluation procedure is as follows. In each epoch, we first train the agent on M episodes of T steps each. At the end of this training, we have a testing phase of running M episodes of the game for T steps. We use $M = 50, T = 20$ for the Home world and $M = 20, T = 250$ for the Fantasy world. For all evaluation episodes, we run the agent following an ϵ-greedy policy with $\epsilon = 0.05$, which makes the agent choose the best action according to its Q-values 95% of the time. We report the agent's performance at each epoch.

Baselines We compare our LSTM-DQN model with three baselines. The first is a *Random* agent that chooses both actions and objects uniformly at random from all available choices.[8] The other two are BOW-DQN and BI-DQN, which use a bag-of-words and a bag-of-bigrams representation of the text, respectively, as input to the DQN action scorer. These baselines serve to illustrate the importance of having a good representation layer for the task.

Settings For our DQN models, we used $\mathcal{D} = 100000, \gamma = 0.5$. We use a learning rate of 0.0005 for RMSprop. We anneal the ϵ for ϵ-greedy from 1 to 0.2 over 100000 transitions. A mini-batch gradient update is performed every 4 steps of the gameplay. We roll out the LSTM (over words) for

[7] We consider 222 possible command combinations of 6 actions and 37 object arguments.

[8] In the case of the Fantasy world, the object choices are narrowed down using game clues as described earlier.



Reward (Home) | Quest completion (Home) | Transfer Learning (Home)

Reward (Fantasy) | Quest completion (Fantasy) | Prioritized Sampling (Home)

Figure 3: **Left:** Graphs showing the evolution of average reward and quest completion rate for BOW-DQN, LSTM-DQN and a Random baseline on the Home world (**top**) and Fantasy world (**bottom**). Note that the reward is shown in log scale for the Fantasy world. **Right:** Graphs showing effects of transfer learning and prioritized sampling on the Home world.

a maximum of 30 steps on the Home world and for 100 steps on the Fantasy world. For the prioritized sampling, we used $\rho = 0.25$ for both worlds. We employed a mini-batch size of 64 and word embedding size $d = 20$ in all experiments.

6 Results

Home World Figure 3 illustrates the performance of LSTM-DQN compared to the baselines. We can observe that the Random baseline performs quite poorly, completing only around 10% of quests on average[9] obtaining a low reward of around -1.58. The BOW-DQN model performs significantly better and is able to complete around 46% of the quests, with an average reward of 0.20. The improvement in reward is due to both greater quest success rate and a lower rate of issuing invalid commands (e.g. *eat apple* would be invalid in the bedroom since there is no apple). We notice that both the reward and quest completion graphs of this model are volatile. This is because the model fails to pick out differences between quests like *Not you are hungry now but you are sleepy now* and *Not you are sleepy now but you are hungry now*. The BI-DQN model suffers from the same issue although it performs slightly better than BOW-DQN by completing 48% of quests. In contrast, the LSTM-DQN model does not suffer from this issue and is able to complete 100% of the quests after around 50 epochs of training, achieving close to the optimal reward possible.[10] This demonstrates that having an expressive representation for text is crucial to understanding the game states and choosing intelligent actions.

In addition, we also investigated the impact of using a deep neural network for modeling the action scorer ϕ_A. Figure 4 illustrates the performance of the BOW-DQN and BI-DQN models along with their simpler versions BOW-LIN and BI-LIN, which use a single linear layer for ϕ_A. It can be seen that the DQN models clearly achieve better performance than their linear counterparts, which points to them modeling the control policy better.

Fantasy World We evaluate all the models on the Fantasy world in the same manner as before and report reward, quest completion rates and Q-

[9] Averaged over the last 10 epochs.

[10] Note that since each step incurs a penalty of -0.01, the best reward (on average) a player can get is around 0.98.

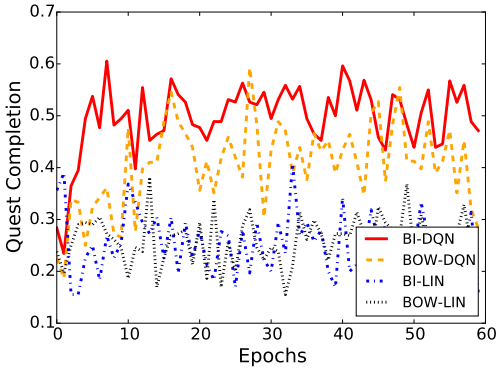


Figure 4: Quest completion rates of DQN vs. Linear models on Home world.

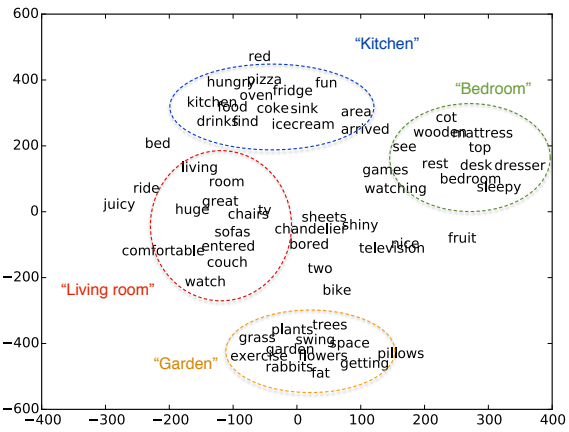


Figure 5: t-SNE visualization of the word embeddings (except stopwords) after training on Home world. The embedding values are initialized randomly.

values. The quest we evaluate on involves crossing the broken bridge (which takes a minimum of five steps), with the possibility of falling off at random (a 5% chance) when the player is on the bridge. The game has an additional quest of reaching a secret tomb. However, this is a complex quest that requires the player to memorize game events and perform high-level planning which are beyond the scope of this current work. Therefore, we focus only on the first quest.

From Figure 3 (bottom), we can see that the Random baseline does poorly in terms of both average per-episode reward[11] and quest completion rates. BOW-DQN converges to a much higher average reward of -12.68 and achieves around 82% quest completion. Again, the BOW-DQN is often confused by varying (10 different) descriptions of the portions of the bridge, which reflects in its erratic performance on the quest. The BI-DQN performs very well on quest completion by finishing 97% of quests. However, this model tends to find sub-optimal solutions and gets an average reward of -26.68, even worse than BOW-DQN. One reason for this is the negative rewards the agent obtains after falling off the bridge. The LSTM-DQN model again performs best, achieving an average reward of -11.33 and completing 96% of quests on average. Though this world does not contain descriptions adversarial to BOW-DQN or BI-DQN, the LSTM-DQN obtains higher average reward by completing the quest in fewer steps and showing more resilience to variations in the state descriptions.

Transfer Learning We would like the representations learnt by ϕ_R to be generic enough and *transferable* to new game worlds. To test this, we created a second Home world with the same rooms, but a completely different map, changing the locations of the rooms and the pathways between them. The main differentiating factor of this world from the original home world lies in the high-level planning required to complete quests.

We initialized the LSTM part of an LSTM-DQN agent with parameters θ_R learnt from the original home world and trained it on the new world.[12] Figure 3 (top right) demonstrates that the agent with transferred parameters is able to learn quicker than an agent starting from scratch initialized with random parameters (*No Transfer*), reaching the optimal policy almost 20 epochs earlier. This indicates that these simulated worlds can be used to learn good representations for language that transfer across worlds.

Prioritized sampling We also investigate the effects of different minibatch sampling procedures on the parameter learning. From Figure 3 (bottom right), we observe that using prioritized sampling significantly speeds up learning, with the agent achieving the optimal policy around 50 epochs faster than using uniform sampling. This shows promise for further research into different schemes of assigning priority to transitions.

Representation Analysis We analyzed the representations learnt by the LSTM-DQN model on the Home world. Figure 5 shows a visualization

[11] Note that the rewards graph is in log scale.

[12] The parameters for the Action Scorer (θ_A) are initialized randomly.

Description	Nearest neighbor
You are halfways out on the unstable bridge. From the castle you hear a distant howling sound, like that of a large dog or other beast.	The bridge slopes precariously where it extends westwards towards the lowest point - the center point of the hang bridge. You clasp the ropes firmly as the bridge sways and creaks under you.
The ruins opens up to the sky in a small open area, lined by columns. ... To the west is the gatehouse and entrance to the castle, whereas southwards the columns make way for a wide open courtyard.	The old gatehouse is near collapse. .... East the gatehouse leads out to a small open area surrounded by the remains of the castle. There is also a standing archway offering passage to a path along the old southern inner wall.

Table 2: Sample descriptions from the Fantasy world and their nearest neighbors (NN) according to their vector representations from the LSTM representation generator. The NNs are often descriptions of the same or similar (nearby) states in the game.

of learnt word embeddings, reduced to two dimensions using t-SNE (Van der Maaten and Hinton, 2008). All the vectors were initialized randomly before training. We can see that semantically similar words appear close together to form coherent subspaces. In fact, we observe four different subspaces, each for one type of room along with its corresponding object(s) and quest words. For instance, food items like *pizza* and rooms like *kitchen* are very close to the word *hungry* which appears in a quest description. This shows that the agent learns to form meaningful associations between the semantics of the quest and the environment. Table 2 shows some examples of descriptions from Fantasy world and their nearest neighbors using cosine similarity between their corresponding vector representations produced by LSTM-DQN. The model is able to correlate descriptions of the same (or similar) underlying states and project them onto nearby points in the representation subspace.

7 Conclusions

We address the task of end-to-end learning of control policies for text-based games. In these games, all interactions in the virtual world are through text and the underlying state is not observed. The resulting language variability makes such environments challenging for automatic game players. We employ a deep reinforcement learning framework to jointly learn state representations and action policies using game rewards as feedback. This framework enables us to map text descriptions into vector representations that capture the semantics of the game states. Our experiments demonstrate the importance of learning good representations of text in order to play these games well. Future directions include tackling high-level planning and strategy learning to improve the performance of intelligent agents.

Acknowledgements

We are grateful to the developers of Evennia, the game framework upon which this work is based. We also thank Nate Kushman, Clement Gehring, Gustavo Goretkin, members of MIT's NLP group and the anonymous EMNLP reviewers for insightful comments and feedback. T. Kulkarni was graciously supported by the Leventhal Fellowship. We would also like to acknowledge MIT's Center for Brains, Minds and Machines (CBMM) for support.

References

Christopher Amato and Guy Shani. 2010. High-level reinforcement learning in strategy games. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 1*, pages 75–82. International Foundation for Autonomous Agents and Multiagent Systems.

Eyal Amir and Patrick Doyle. 2002. Adventure games: A challenge for cognitive robotics. In *Proc. Int. Cognitive Robotics Workshop*, pages 148–155.

Jacob Andreas and Dan Klein. 2015. Alignment-based compositional semantics for instruction following. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

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

SRK Branavan, Luke S Zettlemoyer, and Regina Barzilay. 2010. Reading between the lines: Learning to map high-level instructions to commands. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1268–1277. Association for Computational Linguistics.

SRK Branavan, David Silver, and Regina Barzilay. 2011a. 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.

SRK Branavan, David Silver, and Regina Barzilay. 2011b. Non-linear monte-carlo search in Civilization II. AAAI Press/International Joint Conferences on Artificial Intelligence.

Pavel Curtis. 1992. Mudding: Social phenomena in text-based virtual realities. *High noon on the electronic frontier: Conceptual issues in cyberspace*, pages 347–374.

Mark A DePristo and Robert Zubek. 2001. being-in-the-world. In *Proceedings of the 2001 AAAI Spring Symposium on Artificial Intelligence and Interactive Entertainment*, pages 31–34.

Jacob Eisenstein, James Clarke, Dan Goldwasser, and Dan Roth. 2009. Reading to learn: Constructing features from semantic abstracts. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 958–967, Singapore, August. Association for Computational Linguistics.

Peter Gorniak and Deb Roy. 2005. Speaking with your sidekick: Understanding situated speech in computer role playing games. In R. Michael Young and John E. Laird, editors, *Proceedings of the First Artificial Intelligence and Interactive Digital Entertainment Conference, June 1-5, 2005, Marina del Rey, California, USA*, pages 57–62. AAAI Press.

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

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

Jan Koutník, Giuseppe Cuccu, Jürgen Schmidhuber, and Faustino Gomez. 2013. Evolving large-scale neural networks for vision-based reinforcement learning. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pages 1061–1068. ACM.

Nate Kushman, Yoav Artzi, Luke Zettlemoyer, and Regina Barzilay. 2014. Learning to automatically solve algebra word problems. *ACL (1)*, pages 271–281.

Cynthia Matuszek, Evan Herbst, Luke Zettlemoyer, and Dieter Fox. 2013. Learning to parse natural language commands to a robot control system. In *Experimental Robotics*, pages 403–415. Springer.

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

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. 2015. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 02.

Andrew W Moore and Christopher G Atkeson. 1993. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, 13(1):103–130.

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

David Silver, Richard S Sutton, and Martin Müller. 2007. Reinforcement learning of local shape in the game of go. In *IJCAI*, volume 7, pages 1053–1058.

Ilya Sutskever, Oriol Vinyals, and Quoc VV Le. 2014. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pages 3104–3112.

Richard S Sutton and Andrew G Barto. 1998. *Introduction to reinforcement learning*. MIT Press.

István Szita. 2012. Reinforcement learning in games. In *Reinforcement Learning*, pages 539–577. Springer.

Kai Sheng Tai, Richard Socher, and Christopher D. Manning. 2015. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1556–1566, Beijing, China, July. Association for Computational Linguistics.

Tijmen Tieleman and Geoffrey Hinton. 2012. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 4.

Laurens Van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(2579-2605):85.

Adam Vogel and Dan Jurafsky. 2010. Learning to follow navigational directions. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 806–814. Association for Computational Linguistics.

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