Deep Transfer in Reinforcement Learning by Language Grounding

Karthik Narasimhan, Regina Barzilay and Tommi Jaakkola
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
{karthikn, regina, tommi}@csail.mit.edu

Abstract

In this paper, we explore the utilization of natural language to drive transfer for reinforcement learning (RL). Despite the wide-spread application of deep RL techniques, learning generalized policy representations that work across domains remains a challenging problem. We demonstrate that textual descriptions of environments provide a compact intermediate channel to facilitate effective policy transfer. We employ a model-based RL approach consisting of a differentiable planning module, a model-free component and a factorized representation to effectively utilize entity descriptions. Our model outperforms prior work on both transfer and multi-task scenarios in a variety of different environments.

1 Introduction

Deep reinforcement learning has emerged as a method of choice for many control applications, ranging from computer games (Mnih et al., 2015; Silver et al., 2016) to robotics (Levine et al., 2016). However, the success of this approach depends on a substantial number of interactions with the environment during training, easily reaching millions of steps (Nair et al., 2015; Mnih et al., 2016). Moreover, given a new task, even a related one, this training process has to be performed from scratch. This inefficiency has motivated recent work in learning universal policies that can generalize across related tasks (Schaul et al., 2015), as well as other transfer approaches (Parisotto et al., 2016; Rajendran et al., 2017). In this paper, we explore transfer methods that utilize text descriptions to facilitate policy generalization across tasks.

As an example, consider the game environments in Figure 1. The two games – Boulderchase and Bomberman – differ in their layouts and entity types. However, the high-level behavior of most entities in both games is similar. For instance, the scorpion in Boulderchase (left) is a moving entity which the agent has to avoid, similar to the spider in Bomberman (right). Though this similarity is clearly reflected in the text descriptions in Figure 1, it may take multiple environment interactions to discover. Therefore, exploiting these textual clues could help an autonomous agent understand this connection more effectively, leading to faster policy learning.

To test this hypothesis, we consider multiple environments augmented with textual descriptions. These descriptions provide a short overview of the objects and their modes of interaction in the en-

Figure 1: Two different game environments with a few associated text descriptions. Entity names are replaced with icons for the purpose of illustration.
vironment. They do not describe control strategies, which were commonly used in prior work on grounding (Vogel and Jurafsky, 2010; Branavan et al., 2011). Instead, they specify the dynamics of the environments, which are more conducive to cross-domain transfer.

In order to effectively utilize this type of information, we employ a model-based reinforcement learning approach. Typically, representations of the environment learned by these approaches are inherently domain specific. We address this issue by using natural language as an implicit intermediate channel for transfer. Specifically, our model learns to map text descriptions to transitions and rewards in an environment, a capability that speeds up learning in unseen domains. We induce a two-part representation for the input state that generalizes over domains, incorporating both domain-specific information and textual knowledge. This representation is utilized by an action-value function, parametrized as a single deep neural network with a differentiable value iteration module (Tamar et al., 2016). The entire model is trained end-to-end using rewards from the environment.

We evaluate our model on several game worlds from the GVGAI framework (Perez-Liebana et al., 2016). In our first evaluation scenario of transfer learning, an agent is trained on a set of source tasks and its learning performance is evaluated on a different set of target tasks. Across multiple evaluation metrics, our method consistently outperforms several baselines and an existing transfer approach called Actor Mimic (Parisotto et al., 2016). For instance, our model achieves up to 35% higher average reward and up to 15% higher jumpstart reward. We also evaluate on a multi-task setting where learning is simultaneously performed on multiple environments. In this case, we obtain gains of up to 30% and 7% on average and asymptotic reward, respectively.

2 Related Work

Grounding language in interactive environments

In recent years, there has been increasing interest in systems that can utilize textual knowledge to learn control policies. Such applications include interpreting help documentation (Branavan et al., 2010), instruction following (Vogel and Jurafsky, 2010; Kollar et al., 2010; Artzi and Zettlemoyer, 2013; Matusek et al., 2013; Andreas and Klein, 2015) and learning to play computer games (Branavan et al., 2011; Narasimhan et al., 2015). In all these applications, the models are trained and tested on the same domain.

Our work represents two departures from prior work on grounding. First, rather than optimizing control performance for a single domain, we are interested in the multi-domain transfer scenario, where language descriptions drive generalization. Second, prior work uses text in the form of strategy advice to directly learn the policy. Since the policies are typically optimized for a specific task, they may be harder to transfer across domains. Instead, we utilize text to bootstrap the induction of the environment dynamics, moving beyond task-specific strategies.

Another related line of work consists of systems that learn spatial and topographical maps of the environment for robot navigation using natural language descriptions (Walter et al., 2013; Hemachandra et al., 2014). These approaches use text mainly containing appearance and positional information, and integrate it with other semantic sources (such as appearance models) to obtain more accurate maps. In contrast, our work uses language describing the dynamics of the environment, such as entity movements and interactions, which is complementary to positional information received through state observations. Further, our goal is to help an agent learn policies that generalize over different stochastic domains, while their work considers a single domain.

Transfer in Reinforcement Learning

Transferring policies across domains is a challenging problem in reinforcement learning (Konidaris, 2006; Taylor and Stone, 2009). The main hurdle lies in learning a good mapping between the state and action spaces of different domains to enable effective transfer. Most previous approaches have either explored skill transfer (Konidaris and Barto, 2007) or value function transfer (Liu and Stone, 2006). There have been a few attempts at model-based transfer for RL (Taylor et al., 2008; Nguyen et al., 2012; Gašic et al., 2013; Wang et al., 2015) but these methods either rely on hand-coded inter-task mappings for state and actions spaces or require significant interactions.
in the target task to learn an effective mapping. Our approach doesn’t use any explicit mappings and can learn to predict the dynamics of a target task using its descriptions.

A closely related line of work concerns transfer methods for deep reinforcement learning. Parisotto et al. (2016) train a deep network to mimic pre-trained experts on source tasks using policy distillation. The learned parameters are then used to initialize a network on a target task to perform transfer. Rusu et al. (2016) perform transfer by freezing parameters learned on source tasks and adding a new set of parameters for every new target task, while using both sets to learn the new policy. Work by (Rajendran et al., 2017) uses attention networks to selectively transfer from a set of expert policies to a new task. Our approach is orthogonal since we use text to bootstrap transfer, and can potentially be combined with these methods to achieve more effective transfer.

3 General Framework

Environment Setup We model a single environment as a Markov Decision Process (MDP), represented by \( E = \langle S, A, T, R, O, Z \rangle \). Here, \( S \) is the state space, and \( A \) is the set of actions available to the agent. In this work, we consider every state \( s \in S \) to be a 2-dimensional grid of size \( m \times n \), with each cell containing an entity symbol \( o \in O \).\(^1\) \( T \) is the transition distribution over all possible next states \( s' \) conditioned on the agent choosing action \( a \) in state \( s \). \( R \) determines the reward provided to the agent at each time step. The agent does not have access to the true \( T \) and \( R \) of the environment. Each domain also has a goal state \( s_g \in S \) which determines when an episode terminates. Finally, \( Z \) is the complete set of text descriptions provided to the agent for this particular environment.

Reinforcement learning (RL) The goal of an autonomous agent is to maximize cumulative reward obtained from the environment. A traditional way to achieve this is by learning an action value function \( Q(s, a) \) through reinforcement. The \( Q\)-function predicts the expected future reward for choosing action \( a \) in state \( s \). A straightforward policy then is to simply choose the action that maximizes the \( Q \)-value in the current state: \( \pi(s) = \arg\max_a Q(s, a) \). If we also make use of the descriptions, we have a text-conditioned policy: \( \pi(s, Z) = \arg\max_a Q(s, a, Z) \).

A successful control policy for an environment will contain both knowledge of the environment dynamics and the capability to identify goal states. While the latter is task-specific, the former characteristic is more useful for learning a general policy that transfers to different domains. Based on this hypothesis, we employ a model-aware RL approach that can learn the dynamics of the world while estimating the optimal \( Q \). Specifically, we make use of Value Iteration (VI) (Sutton and Barto, 1998), an algorithm based on dynamic programming. The update equations are as follows:

\[
Q^{(n+1)}(s, a, Z) = R(s, a, Z) + \gamma \sum_{s' \in S} T(s'|s, a, Z) V^{(n)}(s', Z)
\]

\[
V^{(n+1)}(s, Z) = \max_a Q^{(n+1)}(s, a, Z)
\]  

(1)

where \( \gamma \) is a discount factor and \( n \) is the iteration number. The updates require an estimate of \( T \) and \( R \), which the agent must obtain through exploration of the environment.

Text descriptions Estimating the dynamics of the environment from interactive experience can require a significant number of samples. Our main hypothesis is that if an agent can derive information about the dynamics from text descriptions, it can determine \( T \) and \( R \) faster and more accurately.

For instance, consider the sentence “Red bat that moves horizontally, left to right.”. This talks about the movement of a third-party entity ('bat'), independent of the agent’s goal. Provided the agent can learn to interpret this sentence, it can then infer the direction of movement of a different entity (e.g. "A tan car moving slowly to the left") in a different domain. Further, this inference is useful even if the agent has a completely different goal. On the other hand, instruction-like text, such as “Move towards the wooden door”, is highly context-specific, only relevant to domains that have the mentioned goal.

\(^1\)In our experiments, we relax this assumption to allow for multiple entities per cell, but for ease of description, we shall assume a single entity. The assumption of 2-D worlds can also be easily relaxed to generalize our model to other situations.
• Scorpion2: Red scorpion that moves up and down
• Alien3: This character slowly moves from right to left while having the ability to shoot upwards
• Sword1: This item is picked up and used by the player for attacking enemies

Figure 2: Some example text descriptions of entities in different environments.

With this in mind, we provide the agent with text descriptions that collectively portray characteristics of the world. A single description talks about one particular entity in the world. The text contains (partial) information about the entity’s movement and interaction with the player avatar. Each description is also aligned to its corresponding entity in the environment. Figure 2 provides some samples; details on data collection and statistics are in Section 5.

Transfer for RL A natural scenario to test our grounding hypothesis is to consider learning across multiple environments. The agent can learn to ground language semantics in an environment $E_1$ and then we can test its understanding capability by placing it in a new unseen domain, $E_2$. The agent is allowed unlimited experience in $E_1$, and after convergence of its policy, it is then allowed to interact with and learn a policy for $E_2$. We do not provide the agent with any mapping between entities or goals across domains, either directly or through the text. The agent’s goal is to re-utilize information obtained in $E_1$ to learn more efficiently in $E_2$.

4 Model

Grounding language for policy transfer across domains requires a model that meets two needs. First, it must allow for a flexible representation that fuses information from both state observations and text descriptions. This representation should capture the compositional nature of language while mapping linguistic semantics to characteristics of the world. Second, the model must have the capability to learn an accurate prototype of the environment (i.e. transitions and rewards) using only interactive feedback. Overall, the model must enable an agent to map text descriptions to environment dynamics; this allows it to predict transitions and rewards in a completely new world, without requiring substantial interaction.

To this end, we propose a neural architecture consisting of two main components: (1) a representation generator ($\phi$), and (2) a value iteration network (VIN) (Tamar et al., 2016). The representation generator takes the state observation and the set of text descriptions to produce a tensor, capturing essential information for decision making. The VIN module implicitly encodes the value iteration computation (Eq. 1) into a recurrent network with convolutional modules, producing an action-value function using the tensor representation as input. Together, both modules form an end-to-end differentiable network that can be trained using simple back-propagation.

4.1 Representation generator

The main purpose of this module is to fuse together information from two inputs - the state, and the text specifications. An important consideration, however, is the ability to handle partial or incomplete text descriptions, which may not contain all the particulars on an entity. Thus, we would like to incorporate useful information from the text, yet, not rely on it completely. This motivates us to utilize a factorized representation over the two input modalities.

Formally, given a state matrix $s$ and a set of text descriptions $Z$, the module produces a tensor $\phi(s, Z)$. Consider a cell in $s$ containing an entity $o_i$, with a corresponding description $z_i$ (if available). Each such cell is converted into a vector $\phi_i = [v_{o_i}; v_{z_i}]$, consisting of two parts concatenated together:

1. $v_{o_i}$, which is an entity-specific vector embed-
Reward

\[ R \]

New value

\[ \hat{V}^{(n+1)} \]

Prev value

\[ \hat{V}^{(n)} \]

\[ f_T \]

\[ f_R \]

\[ \delta_s \]

\[ k \text{ recurrence} \]

\[ Q_{vin} \]

\[ \phi(s, Z) \]

\[ f_R \]

\[ f_T \]

\[ \hat{Q}^{(n+1)}(s, a, Z) \]

\[ \hat{V}^{(n+1)}(s, Z) = \max_a \hat{Q}^{(n+1)}(s, a, Z) \]

\[ (2) \]

\[ (3) \]

\[ \hat{Q}^{(n+1)}(s, a, Z) = f_T\left(f_R(\phi(s, Z), a; \theta_R), \hat{V}^{(n)}(s, Z); \theta_T\right) \]

\[ (2) \]

Figure 4: Value iteration network module to compute \( Q_{vin} \) from \( \phi(s) \). Functions \( f_T \) and \( f_R \) are implemented using convolutional neural networks (CNNs).

2. \( vz_i \) (also of dimension \( d \)), produced from \( z_i \) using an LSTM recurrent neural network (Hochreiter and Schmidhuber, 1997).

This gives us a tensor \( \phi \) with dimensions \( m \times n \times 2d \) for the entire state. For cells with no entity (i.e. empty space), \( \phi_i \) is simply a zero vector, and for entities without a description, \( vz_i = \vec{0} \). Figure 3 illustrates this module.

This decomposition allows us to learn policies based on both the ID of an object and its described behavior in text. In a new environment, previously seen entities can reuse the learned representations directly based on their symbols. For completely new entities (with unseen IDs), the model can form useful representations from their text descriptions.

4.2 Value iteration network

For a model-based RL approach to this task, we require some means to estimate \( T \) and \( R \) of an environment. One way to achieve this is by explicitly using predictive models for both functions and learning these through transitions experienced by the agent. These models can be then used to estimate the optimal \( Q \) with equation 1. However, this pipelined approach would result in errors propagating through the different stages of predictions.

A value iteration network (VIN) (Tamar et al., 2016) abstracts away explicit computation of \( T \) and \( R \) by directly predicting the outcome of value iteration (Figure 4), thereby avoiding the aforementioned error propagation. The VI computation is mimicked by a recurrent network with two key operations at each step. First, to compute \( Q \), we have two functions – \( f_R \) and \( f_T \). \( f_R \) is a reward predictor that operates on \( \phi(s, Z) \) while \( f_T \) utilizes the output of \( f_R \) and any previous \( V \) to predict \( Q \). Both functions are parametrized as convolutional neural networks (CNNs),\(^2\) to suit our tensor representation \( \phi \). Second, the network employs max pooling over the action channels in the \( Q \)-value map produced by \( f_T \) to obtain \( V \). The value iteration computation (Eq. 1) can thus be approximated as:

\[ \hat{Q}^{(n+1)}(s, a, Z) = f_T\left(f_R(\phi(s, Z), a; \theta_R), \hat{V}^{(n)}(s, Z); \theta_T\right) \]

\[ (2) \]

\[ \hat{V}^{(n+1)}(s, Z) = \max_a \hat{Q}^{(n+1)}(s, a, Z) \]

Note that while the VIN operates on \( \phi(s, Z) \), we write \( \hat{Q} \) and \( \hat{V} \) in terms of the original state input \( s \) and text \( Z \), since these are independent of our chosen representation.

The outputs of both CNNs are real-valued tensors – that of \( f_R \) has the same dimensions as the input state \((m \times n)\), while \( f_T \) produces \( \hat{Q}^{(n+1)} \) as a tensor.

\(^2\)Other parameterizations are possible for different input types, as noted in Tamar et al. (2016).
of dimension $m \times n \times |A|$. A key point to note here is that the model produces $Q$ and $V$ values for each cell of the input state matrix, assuming the agent’s position to be that particular cell. The convolution filters help capture information from neighboring cells in our state matrix, which act as approximations for $V^{(n)}(s', Z)$. The parameters of the CNNs, $\theta_R$ and $\theta_T$, approximate $R$ and $T$, respectively. See Tamar et al. (2016) for a more detailed discussion.

The recursive computation of traditional value iteration (Eq. 1) is captured by employing the CNNs in a recurrent fashion for $k$ steps.\(^3\) Intuitively, larger values of $k$ imply a larger field of neighbors influencing the Q-value prediction for a particular cell, as the information propagates longer. Note that the output of this recurrent computation, $\hat{Q}^{(k)}$, will be a 3-D tensor. However, since we need a policy only for the agent’s current location, we use an appropriate selection function $\delta_s$, which reduces this Q-value map to a single set of action values for the agent’s location:

$$Q_{\text{vin}}(s, a, Z; \Theta_1) = \delta_s(\hat{Q}^{(k)}(s, a, Z)) \quad (4)$$

**Final prediction** Games follow a complex dynamics which is challenging to capture precisely, especially longer term. VINs approximate the dynamics implicitly via learned convolutional operations. It is thus likely that the estimated $Q_{\text{vin}}$ values are most helpful for short-term planning that corresponds to a limited number of iterations $k$. We need to complement these local Q-values with estimates based on a more global view.

Following the VIN specification in (Tamar et al., 2016), our architecture also contains a model-free component, implemented as a deep Q-network (DQN) (Mnih et al., 2015). This network also provides a prediction of a Q-value, $Q_v(s, a, Z; \Theta_2)$, which is combined with $Q_{\text{vin}}$ using a composition function $g$:\(^4\)

$$Q(s, a, Z; \Theta) = g(Q_{\text{vin}}(s, a, Z; \Theta_1), Q_v(s, a, Z; \Theta_2)) \quad (5)$$

The fusion of our model components enables our agent to establish the connection between input text descriptions, represented as vectors, and the environment’s transitions and rewards, encoded as VIN parameters. In a new domain, the model can produce a reasonable policy using corresponding text, even before receiving any interactive feedback.

### 4.3 Parameter learning

Our entire model is end-to-end differentiable. We perform updates derived from the Bellman equation (Sutton and Barto, 1998):

$$Q_{t+1}(s, a, Z) = E[r + \gamma \max_{a'} Q_t(s', a', Z) | s, a]$$

where the expectation is over all transitions from state $s$ with action $a$ and $i$ is the update number. To learn our parametrized Q-function (the result of Eq. 5), we can use backpropagation through the network to minimize the following loss:

$$\mathcal{L}_i(\Theta_i) = E_{s,a}[|y_i - Q(\hat{s}, \hat{a}, Z; \Theta_i)|^2] \quad (6)$$

where $y_i = r + \gamma \max_{a'} Q(s', a', Z; \Theta_{i-1})$ is the target Q-value with parameters $\Theta_{i-1}$ fixed from the previous iteration. We employ an experience replay memory $\mathcal{D}$ to store transitions (Mnih et al., 2015), and periodically perform updates with random samples from this memory. We use an $\epsilon$-greedy policy (Sutton and Barto, 1998) for exploration.

**Transfer procedure** The traditional transfer learning scenario considers a single task in both source and target environments. To better test generalization and robustness of our methods, we consider transfer from multiple source tasks to multiple target tasks. We first train a model to achieve

---

3. $k$ is a model hyperparameter.

4. Although $g$ can also be learned, we use component-wise addition in our experiments.

---

**Algorithm 1 MULTITASK_TRAIN($\mathcal{E}$)**

1: Initialize parameters $\Theta$ and experience replay $\mathcal{D}$
2: for $k = 1, M$ do
3:     New episode
4:     Initialize $E_k$; get start state $s_1 \in S_k$
5:     for $t = 1, N$ do
6:         New step
7:         Select $a_t \sim \text{EPS-GREEDY}(s_t, Q_{\Theta}, \epsilon)$
8:         Execute action $a_t$, observe reward $r_t$ and new state $s_{t+1}$
9:         Sample mini batch $(s_j, a_j, r_j, s_{j+1}, Z_k) \sim \mathcal{D}$
10:        Do gradient descent on loss $\mathcal{L}$ to update $\Theta$
11:        if $s_{t+1}$ is terminal then break
12:    Return $\Theta$
optimal performance on the set of source tasks. All model parameters ($\Theta$) are shared between the tasks. The agent receives one episode at a time from each environment in a round-robin fashion, along with the corresponding text descriptions. Algorithm 1 details this multi-task training procedure.

After training converges, we use the learned parameters to initialize a model for the target tasks. All parameters of the VIN are replicated, while most weights of the representation generator are reused. Specifically, previously seen objects and words obtain their learned entity-specific embeddings ($v_o$), whereas vectors for new objects and unseen words in the target tasks are initialized randomly. All parameters are then fine-tuned on the target tasks, again with episodes sampled in a round-robin fashion.

5 Experimental Setup

Environments We perform experiments on a series of 2-D environments within the GVGAI framework (Perez-Liebana et al., 2016), which is used for an annual video game AI competition. In addition to pre-specified games, the framework supports the creation of new games using the Py-VGDL description language (Schaul, 2013). We use four different games to evaluate transfer and multitask learning: Freeway, Bomberman, Boulderchase and Friends & Enemies. There are certain similarities between these games. For one, each game consists of a 16x16 grid with the player controlling a movable avatar with two degrees of freedom. Also, each domain contains other entities, both stationary and moving (e.g. diamonds, spiders), that can interact with the avatar.

However, each game also has its own distinct characteristics. In Freeway, the goal is to cross a multi-lane freeway while avoiding cars in the lanes. The cars move at various paces in either horizontal direction. Bomberman and Boulderchase involve the player seeking an exit door while avoiding enemies that either chase the player, run away or move at random. The agent also has to collect resources like diamonds and dig or place bombs to clear paths. These three games have five level variants each with different map layouts and starting entity placements.

Friends & Enemies (F&E) is a new environment we designed, with a larger variation of entity types. This game has a total of twenty different non-player entities, each with different types of movement and interaction with the player’s avatar. For instance, some entities move at random while some chase the avatar or shoot bullets that the avatar must avoid. The objective of the player is to meet all friendly entities while avoiding enemies. For each game instance, four non-player entities are sampled from this pool and randomly placed in the grid. This makes F&E instances significantly more varied than the previous three games. We created two versions of this game: F&E-1 and F&E-2, with the sprites in F&E-2 moving faster, making it a harder environment. Table 1 shows all the different transfer scenarios we consider in our experiments.

| Condition            | Source | Target |
|----------------------|--------|--------|
| F&E-1 → F&E-2        | 7      | 3      |
| F&E-1 → Freeway      | 7      | 5      |
| Bomberman → Boulderchase | 5   | 5      |

Table 1: Number of source and target game instances for various transfer experiments.

Text descriptions We collect textual descriptions using Amazon Mechanical Turk (Buhrmester et al., 2011). We provide annotators with sample gameplay videos of each game and ask them to describe specific entities in terms of their movement and interactions with the avatar. Since we ask the users to provide an independent account of each entity, we obtain “descriptive” sentences as opposed to “instructive” ones which inform the optimal course of action from the avatar’s viewpoint. From manual verification, we find less than 3% of the obtained annotations to be “instructive”.

We aggregated together four sets of descriptions, each from a different annotator, for every environment. Each description in an environment is aligned to one constituent entity. We also make sure that the entity names are not repeated across games (even for the same entity type). Table 2 provides corpus-level statistics on the collected data and Figure 2 has sample descriptions.

Evaluation metrics We evaluate transfer performance using three metrics employed in previous ap-
proaches (Taylor and Stone, 2009):

- **Average Reward**, which is area under the reward curve divided by the number of test episodes.

- **Jumpstart performance**, which is the average reward over first 100k steps.

- **Asymptotic performance**, which is the average reward over 100k steps after convergence.

For the multitask scenario, we consider the average and asymptotic reward only. We repeat all experiments with three different random seeds and report average numbers.

### Baselines

We explore several baselines:

- **NO TRANSFER**: A deep Q-network (DQN) (Mnih et al., 2015) is initialized randomly and trained from scratch on target tasks. This is the only case that does not use parameters transferred from source tasks.

- **DQN**: A DQN is trained on source tasks and its parameters are transferred to target tasks. This model does not make use of text descriptions.

- **TEXT-DQN**: DQN with hybrid representation $\phi(s, Z)$, using text descriptions. This is essentially a reactive-only version of our model, i.e. without the VIN planning module.

- **AMN**: Actor-Mimic network, a recently proposed (Parisotto et al., 2016) transfer method for deep RL using policy distillation.\(^6\)

### Implementation details

For all models, we set $\gamma = 0.8$, $D = 250k$. We used the *Adam* (Kingma and Ba, 2014) optimization scheme with a learning rate of $10^{-4}$, annealed linearly to $5 \times 10^{-5}$. The minibatch size was set to 32. $\epsilon$ was annealed from 1 to 0.1 in the source tasks and set to 0.1 in the target tasks. For the value iteration module (VIN), we experimented with different levels of recurrence, $k \in \{1, 2, 3, 5\}$ and found $k = 1$ or $k = 3$ to work best.\(^7\) For DQN, we used two convolutional layers followed by a single fully connected layer, with ReLU non-linearities. The CNNs in the VIN had filters and strides of length 3. The CNNs in the model-free component used filters of sizes $\{4, 2\}$ and corresponding strides of size $\{3, 2\}$. All embeddings are initialized randomly.\(^8\)

### 6 Results

#### Transfer performance

Table 3 demonstrates that transferring policies positively assists learning in new domains. Our model, TEXT-VIN, performs at par or better than the baselines across all the different metrics. On the first metric of *average reward*, TEXT-VIN (1) achieves a 8% gain (absolute) over AMN on F&E-1 $\rightarrow$ F&E-2, while TEXT-VIN (3) achieves a 35% gain (absolute) over TEXT-DQN on F&E-1 $\rightarrow$ Freeway. This is also evident from a sample reward curve, shown in Figure 5 (left).

In *jumpstart* evaluation, all the transfer approaches outperform the NO TRANSFER baseline, except for AMN on Bomberman $\rightarrow$ Boulderchase. In all domains, the TEXT-DQN model obtains higher scores than DQN, already demonstrating the advantage of using text for transfer. TEXT-VIN (3) achieves the highest numbers in all transfer settings, demonstrating effective utilization of text descriptions to bootstrap in a new environment.

On the final metric of *asymptotic performance*, TEXT-VIN achieves the highest convergence scores, except on F&E-1 $\rightarrow$ F&E-2, where AMN obtains a score of 1.64. This is partly due to its smoother convergence; improving the stability of our model training could boost its asymptotic performance.

#### Negative transfer

We also observe the challenging nature of policy transfer in some scenarios. For example, in Bomberman $\rightarrow$ Boulderchase, TEXT-DQN and AMN achieve a lower average reward and lower asymptotic reward than the NO TRANSFER model, exhibiting negative transfer (Taylor and

\(^6^\)We only evaluate AMN on transfer since it does not perform online multitask learning and is not directly comparable.

\(^7^\)We still observed transfer gains with all $k$ values.

\(^8^\)We also experimented with using pre-trained word embeddings for text but obtained equal or worse performance.

\(^9^\)This fact is also noted in (Parisotto et al., 2016)
### Table 3: Transfer learning results under the different metrics for different domains (Avg. is average reward over time, Asymp. is asymptotic reward). Numbers in parentheses for TEXT-VIN indicate k value. TEXT- models make use of textual descriptions. The max reward attainable (ignoring step penalties) in the target environments is 2.0, 1.0 and at least 25.0 in F&E, Freeway and Boulderchase, respectively. Higher scores are better; bold indicates best numbers.

| Model                   | F&E-1 → F&E-2 | F&E-1 → Freeway | Bomberman → Boulderchase |
|-------------------------|---------------|----------------|--------------------------|
|                         | Avg. Jumpstart Asymp. | Avg. Jumpstart Asymp. | Average Jumpstart Asymp. |
| NO TRANSFER             | 0.86 -0.19 1.40 | 0.15 -1.06 0.81 | 9.50 2.88 10.99           |
| DQN                     | 1.02 0.73 1.30 | 0.06 -0.96 0.82 | 9.63 3.84 11.28           |
| TEXT-DQN                | 1.03 0.40 1.33 | 0.38 -0.50 **0.85** | 8.52 3.42 9.45           |
| AMN (Actor Mimic)       | 1.22 0.13 **1.64** | 0.08 -0.84 0.75 | 6.22 0.78 8.20           |
| TEXT-VIN (1)            | **1.38** 0.93 1.53 | 0.63 -0.58 **0.85** | **11.41** 4.42 **12.06** |
| TEXT-VIN (3)            | 1.27 **1.04** 1.44 | **0.73** -0.01 **0.85** | 10.93 **4.49** **12.09** |

![Figure 5: Reward curves for: (left) transfer condition F&E-1 → Freeway, and (right) multitask learning in F&E-2. Numbers in parentheses for TEXT-VIN indicate k value. All graphs averaged over 3 runs with different seeds; shaded areas represent bootstrapped confidence intervals.](image)

Table 4: Multitask learning over 20 games in F&E-2.

| Model       | Avg | Asymp. |
|-------------|-----|--------|
| DQN         | 0.65| 1.38   |
| TEXT-DQN    | 0.71| 1.49   |
| TEXT-VIN (1)| **1.32**| **1.63** |
| TEXT-VIN (3)| 1.24| 1.57   |

Stone, 2009). Further, TEXT-DQN performs worse than a vanilla DQN in such cases, which further underlines the need for a model-aware approach to truly take advantage of the descriptive text.

**Multi-task performance** The learning benefits observed in the transfer scenario are also present in the multi-task setup. Table 4 details the average reward and asymptotic reward for learning across twenty variants of the F&E-2 domain, simultaneously. Our model utilizes the text to learn faster and achieve higher optimum scores, with TEXT-VIN (1) showing gains over TEXT-DQN of 30% and 7% on average and asymptotic rewards, respectively. Figure 5 (right) shows the corresponding reward curves.

**Effect of factorized representation** We investigate the usefulness of our factorized representation by training a variant of our model using only a text-based vector representation (Text only) for each entity. We consider two different transfer scenarios – (a) when both source and target instances are from the same domain (F&E-1 → F&E-1) and (b) when the source/target instances are in different domains (F&E-1 → F&E-2). In both cases, we see that
Figure 6: Value maps (\(\hat{V}^{(k)}(s, Z)\)) produced by the VIN module for (a) seen entity (friend), (b) unseen entity with no description, (c) unseen entity with 'friendly' description, and (d) unseen entity with 'enemy' description. Agent is at (4,4) and the non-player entity is at (2,6).

| Condition          | Model            | Avg | Jump | Asymp. |
|--------------------|------------------|-----|------|--------|
| F&E-1 \(\rightarrow\) F&E-1 | Text only        | 1.64| 0.48 | 1.78   |
|                    | Text+entity ID   | 1.70| 1.09 | 1.78   |
| F&E-1 \(\rightarrow\) F&E-2 | Text only        | 0.86| 0.49 | 1.11   |
|                    | Text+entity ID   | 1.27| 1.04 | 1.44   |

Table 5: Transfer results using different input representations with TEXT-VIN (3). Text only means only a text-based vector is used, i.e. \(\phi(s) = v_z(s, Z)\). Text+entity ID is our full representation, \(\phi(s) = [v_z(s, Z); v_o(s)]\)

Table 6: Average rewards in Bomberman \(\rightarrow\) Boulderchase with different text representations: Sum of word vectors, or an LSTM over the entire sentence.

leads to high values in the surrounding areas, as expected. In the second map, the entity is unseen and without any descriptions; hence, the values are uninformed. The third and fourth maps, however, contain unseen entities with descriptions. In these cases, the module predicts higher or lower values around the entity depending on whether the text portrays it as a friend or enemy. Thus, even before a single interaction in a new domain, our model can utilize text to generate good value maps. This bootstraps the learning process, making it more efficient.

7 Conclusion

We have proposed a novel method of utilizing natural language to drive transfer for reinforcement learning (RL). We show that textual descriptions of environments provide a compact intermediate channel to facilitate effective policy transfer. Using a model-aware RL approach, our design consists of a differentiable planning module (VIN), a model-free component and a two-part representation to effectively utilize entity descriptions. We demonstrate the effectiveness of our approach on both transfer and multi-task scenarios in a variety of environments.
References

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. 2011. 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.

Michael Buhrmester, Tracy Kwang, and Samuel D. Gosling. 2011. Amazon’s mechanical turk. Perspectives on Psychological Science, 6(1):3–5. PMID: 26162106.

Miša Gašić, Catherine Breslin, Matthew Henderson, Dongho Kim, Martin Szummer, Blaise Thomson, Pirros Tsiakoulis, and Steve Young. 2013. Pomdp-based dialogue manager adaptation to extended domains. In Proceedings of SIGDIAL.

Sachithra Hemachandra, Matthew R Walter, Stefanie Tellex, and Seth Teller. 2014. Learning spatial-semantic representations from natural language descriptions and scene classifications. In Robotics and Automation (ICRA), 2014 IEEE International Conference on, pages 2623–2630. IEEE.

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

Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

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.

George Konidaris and Andrew G Barto. 2007. Building portable options: Skill transfer in reinforcement learning. In IJCAI, volume 7, pages 895–900.

George D Konidaris. 2006. A framework for transfer in reinforcement learning. In ICML-06 Workshop on Structural Knowledge Transfer for Machine Learning.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. 2016. End-to-end training of deep visuomotor policies. Journal of Machine Learning Research, 17(39):1–40.

Yaxin Liu and Peter Stone. 2006. Value-function-based transfer for reinforcement learning using structure mapping. In Proceedings of the national conference on artificial intelligence, volume 21, page 415. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999.

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.

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, Dharsan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. 2015. Human-level control through deep reinforcement learning. Nature, 518(7540):529–533, 02.

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

Arun Nair, Praveen Srinivasan, Sam Blackwell, Cagdas Alcicek, Rory Fearon, Alessandro De Maria, Vedavyas Panneershelvam, Mustafa Suleyman, Charles Beattie, Stig Petersen, et al. 2015. Massively parallel methods for deep reinforcement learning.

Karthik Narasimhan, Tejas Kulkarni, and Regina Barzilay. 2015. Language understanding for text-based games using deep reinforcement learning. In Proceedings of the Conference on Empirical Methods in Natural Language Processing.

Trung Nguyen, Tomi Silander, and Tze Y Leong. 2012. Transferring expectations in model-based reinforcement learning. In Advances in Neural Information Processing Systems, pages 2555–2563.

Emilio Parisotto, Jimmy Lei Ba, and Ruslan Salakhutdinov. 2016. Actor-mimic: Deep multitask and transfer reinforcement learning. International Conference on Learning Representations.

Diego Perez-Liebana, Spyridon Samothrakis, Julian Togelius, Tom Schaul, and Simon M Lucas. 2016. General video game ai: Competition, challenges and opportunities. In Thirtieth AAAI Conference on Artificial Intelligence.

Janarthanan Rajendran, Aravind Lakshminarayanan, Mitesh M Khapra, Balaraman Ravindran, et al. 2017.
*a2t*: Attend, adapt and transfer: Attentive deep architecture for adaptive transfer from multiple sources.

Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. 2016. Progressive neural networks. *arXiv preprint arXiv:1606.04671*.

Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. 2015. Universal value function approximators. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 1312–1320.

Tom Schaul. 2013. A video game description language for model-based or interactive learning. In *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*, pages 1–8. IEEE.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489.

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

Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. 2016. Value iteration networks. In *Advances in Neural Information Processing Systems*, pages 2154–2162.

Matthew E Taylor and Peter Stone. 2009. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(Jul):1633–1685.

Matthew E Taylor, Nicholas K Jong, and Peter Stone. 2008. Transferring instances for model-based reinforcement learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 488–505. Springer.

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.

Matthew R Walter, Sachithra Hemachandra, Bianca Homberg, Stefanie Tellex, and Seth Teller. 2013. Learning semantic maps from natural language descriptions. *Robotics: Science and Systems*.

Zhuoran Wang, Tsung-Hsien Wen, Pei-Hao Su, and Yannis Stylianou. 2015. Learning domain-independent dialogue policies via ontology parameterisation. In *SIGDIAL Conference*, pages 412–416.