GRAPH-DQN: FAST GENERALIZATION TO NOVEL OBJECTS USING PRIOR RELATIONAL KNOWLEDGE

Varun V. Kumar, Hanlin Tang & Arjun K. Bansal∗
Intel AI Lab
{varun.v.kumar,hanlin.tang,arjun.bansal}@intel.com

1 INTRODUCTION

Humans have a remarkable ability to both generalize known actions to novel objects, and reason about novel objects once their relationship to known objects is understood. For example, on being told a novel object (e.g. ‘bees’) is to be avoided, we readily apply our prior experience avoiding known objects without needing to experience a sting. Deep Reinforcement Learning (RL) has achieved many remarkable successes in recent years including results with Atari (Mnih et al., 2015) games and Go (Silver et al., 2016) that have matched or exceeded human performance. While a human playing Atari games can, with a few sentences of natural language instruction, quickly reach a decent level of performance, modern end-to-end deep reinforcement learning methods still require millions of frames of experience (for e.g. see Fig. 3 in (Lake et al., 2016)). Past studies have hypothesized a role for prior knowledge in addressing this gap between human performance and Deep RL (Dubey et al., 2018; Lake et al., 2016). However, scalable approaches for combining prior or instructional knowledge with deep reinforcement learning have remained elusive.

While other works have studied the problem of generalizing tasks involving the same objects (and relations) to novel environments, goals, or dynamics (Finn et al., 2017; Nichol et al., 2018; Packer et al., 2018; Rusu et al., 2016; Wang et al., 2018; Zambaldi, 2019), here we specifically study the problem of generalizing known relationships to novel objects. Zero-shot transfer of such relations could provide a powerful mechanism for learning to solve novel tasks. We speculated that objects might be an intermediate representation to combine the visual scene with prior knowledge about objects and their relations (Devin et al., 2017; Janner et al., 2019; Wang et al., 2018). Prior information or instruction can take the form of a knowledge graph (Ammanabrolu & Riedl, 2018; Beetz et al., 2015; Lenat et al., 1986; Saxena et al., 2014) in the form of \( \langle \text{subject}, \text{relation}, \text{object} \rangle \) triplets.

In this paper, we present a new model, Graph-DQN, which combines information from knowledge graphs and visual scenes, allowing the agent to learn, reason, and apply agent-object and object-object relations. Graph-DQN was more sample efficient by 5-10x compared to the baseline DQN algorithm (Mnih et al., 2015) on a Warehouse game and a symbolic version of Pacman. When tested on unseen objects in the warehouse game, Graph-DQN generalizes whereas the baseline fails. We observed agents' behavior while removing edges during runtime (i.e., using a trained Graph-DQN), which confirmed that those edges were grounded to the game semantics. These results demonstrate that Graph-DQN can provide a foundation for faster learning and generalization in deep reinforcement learning by leveraging prior information in the form of structured relations.

2 GRAPH-DQN

In deep learning based RL agents such as Deep Q-Networks (DQN), the input state is typically a 2-D feature map representing the game world as either RGB pixels, or as a symbolic environment. In this paper, we design Graph-DQNs for symbolic grid worlds. Our model combines a provided prior knowledge graph with the symbolic environment, and leverages graph convolutions to reason over entities in both the knowledge graph and the game state.

∗Use footnote for providing further information about author (webpage, alternative address)—not for acknowledging funding agencies. Funding acknowledgements go at the end of the submission.
2.1 KNOWLEDGE GRAPH

The knowledge graph \( K = (V, E) \) is a directed graph provided as vertices for each symbol in the environment (for subjects and objects), \( V = \{v_1, v_2, v_3, v_4, \ldots \} \) initially encoded as one-hot vectors of length \( |V| \), and edge features \( E = \{e_{12}, e_{13}, e_{23}, \ldots \} \). The edge features (for relations) are represented as one-hot vectors. The connectivity of the graph, as well as the edge features are designed to reflect the structure of the environment. During training, the knowledge graph’s structure and features are fixed. Importantly, while we provide the one-hot encoded representation of the edge relationships, the Graph-DQN must learn to ground the meaning of this representation in terms of rewarding actions during training. If successfully grounded, the Graph-DQN may use this representation during the test phase when it encounters novel objects connected with known relationships to entities in the knowledge graph.

2.2 GRAPH CONVOLUTION

In order to compute features for the entities in the knowledge graph, we use an edge-conditioned graph convolution (ECC) (Simonovsky & Komodakis, 2017). In this formulation, a multilayer perceptron network is used to generate the filters given the edge features as input. Each graph layer \( g \) computes the feature of each vertex \( v_i \) as:

\[
v_i = \sum_{v_j \in \mathcal{N}(v_i)} \Theta[e_{ij}] v_j + b
\]

where the weight function \( \Theta[e_{ij}] \) depends only on the edge feature and is parameterized as a neural network. \( \mathcal{N}(v_i) \) is the set of nodes with edges into \( v_i \). Our implementation uses graph convolution layers with \( d = 64 \) features, and the weight network is a single linear layer with 8 hidden units. The output of \( g \) is a graph with vertices \( V \in \mathbb{R}^{|V| \times d} \).

2.3 MODEL COMPONENTS

We introduce several operations for transferring information between the state representation \( S \) and the knowledge graph \( K \). We can Broadcast the knowledge graph features into the state, or use Pooling to gather the state features into the knowledge graph nodes. We can also update the state representation by jointly convolving over \( (S, K) \), which we call KG-Conv. The supplement has further details of these operations.

3 EXPERIMENTS

Previous environments measured generalization to more difficult levels (Cobbe et al., 2018; Nichol et al., 2018), modified environment dynamics (Packer et al., 2018), or different solution paths (Zambaldi, 2019). These environments, however, do not introduce new objects at test time. To quantify the generalization ability of Graph-DQN to unseen objects, we needed a symbolic game with the ability to increment the difficulty in terms of the number of new objects and relationships. Therefore, we introduce a new Warehouse environment, where the agent pushed balls into the corresponding bucket, and new ball and bucket objects and their pairing are provided at test time. We also benchmarked our model and the baseline DQN algorithm on a symbolic version of Pacman\(^1\) in order to compare their sample efficiency in a more difficult environment.

3.1 WAREHOUSE

The warehouse environment is implemented using the pycolab environment (Stepleton, 2017). The environment consists of a 10 \( \times \) 10 grid, where the agent is rewarded for pushing balls into their matching buckets. The set of rewarded ball-bucket pairs varies, and in the test games the agent sees balls or buckets not seen during training. For the variations, see Table [1]. Lower case alphanumeric characters refer to balls, and upper case to buckets. We increase the difficulty of the environment

\[^1\]http://ai.berkeley.edu/project_overview.html
Table 1: Experiment variations for the Warehouse environment. The agent is rewarded for pushing the ball into the correct bucket. For each type, we list the rewarded ball-bucket pairs in the training and test games. Note that the test games only include ball types not seen in the training games. Sets denote rewarded combinations. For example, \{b, c\} → B means b → B and c → B are rewarded.

<table>
<thead>
<tr>
<th>Name</th>
<th>Training Pairs</th>
<th>Test Pairs</th>
</tr>
</thead>
<tbody>
<tr>
<td>one-one</td>
<td>b → B</td>
<td>c → B</td>
</tr>
<tr>
<td>two-one</td>
<td>{b, c} → B</td>
<td>d → B</td>
</tr>
<tr>
<td>five-two</td>
<td>{b, c, d, e, f} → B</td>
<td>{g, h} → B</td>
</tr>
<tr>
<td>buckets</td>
<td>b → B, c → C</td>
<td>g → G, h → H</td>
</tr>
<tr>
<td></td>
<td>d → D, e → E</td>
<td>i → I, j → J</td>
</tr>
<tr>
<td></td>
<td>f → F</td>
<td>k → K</td>
</tr>
<tr>
<td>buckets-repeat</td>
<td>{b, c, d} → B</td>
<td>{q, r, s} → G,</td>
</tr>
<tr>
<td></td>
<td>{e, f, g} → C, ...,</td>
<td>{t, u, v} → H,</td>
</tr>
<tr>
<td></td>
<td>{n, o, p} → F</td>
<td>..., {6, 7, 8} → K</td>
</tr>
</tbody>
</table>
Figure 1: Warehouse results. For the environments described in Table 1 (columns), performance of the baseline DQN (green), our proposed Graph-DQN (blue), and a variant of Graph-DQN with edges removed (orange) over the number of training episodes. The top row shows the success rate (fraction completed within 100 steps) on training environments, and the bottom row shows the success rate on test environments without additional training. Bold lines are the mean success rate over $n = 10$ runs, and shaded area denotes the standard error. A moving average of $t = 100$ episodes was applied. The Graph-DQN model is more sample efficient during training, and also generalizes to test environments.

We tested zero-shot transfer learning by placing the trained agents in environments with objects unseen during training. The Graph-DQN is able to leverage the knowledge graph to generalize, solving in $> 80\%$ of the test environments (see Figure 1 bottom row). The baseline DQN failed completely to generalize to these environments.

Additional control experiments ruled out confounding factors. The baseline DQN was able to generalize to an environment with the same pairs as training but new locations, so spatial generalization was not a trivial failure mode (Success rate=72% in the one-one environment; figure not shown). When we deleted the edges from the Graph-DQN (orange lines), the model trained slower and failed to generalize. The No Edge condition still trained faster than the baseline Conv-DQN, possibly due to additional parameters in the KG-Conv. However, that advantage is minimal in our most complex Warehouse environment, the buckets-repeat. We also tested baselines with significantly more parameters and different learning rates without improvement.

### 4.2 PACMAN

We compare Graph-DQN to the baseline Conv-DQN on four symbolic Pacman environments (Figure 2). The Graph DQN converges significantly faster to a performing control policy than the convolution-based DQN on all four environments. The difference is particularly noticeable on the mediumClassic map, where the Graph DQN achieves a significant positive reward of 1,000 within 5,000 episodes, whereas the convolutional model takes over 10,000 episodes to reach a reward of 500. However, we also note that at the end of the training period, both agents have a similar level of performance. This is expected, since the convolutional model should eventually be able to deduce the relations between the symbols with enough training. In the mediumClassic environment, the Conv-DQN surpassed the Graph-DQN reward after 10,000 episodes, while in capsuleClassic Graph-DQN sustained a higher final reward. Future work will assess the source of these differences.

### 4.3 WHAT DO THE AGENTS LEARN?

To understand how the agents are interpreting the edge relations between objects, we observed the behavior of a trained agent running in an environment while manipulating the knowledge graph (Figure 3). For simplicity consider the one-one environment, with one bucket pair $(b \rightarrow B)$ during training and one pair $(c \rightarrow B)$ during testing. A successful behavior is shown in Figure 3. When we removed $b \rightarrow B$, the agent still pushes the ball, but does not know where to push the ball towards, suggesting that the agent has learned to ground the feature $e_{bB} = 2$ as ‘goal’ or ‘fills’. We swapped
Figure 2: Pacman results. Performance of the baseline Conv-DQN (green) and GraphDQN (blue) agent on several symbolic Pacman environments (smallGrid, mediumGrid, mediumClassic, and capsuleClassic). Bold lines are the mean, with the individual \( n = 3 \) repetitions indicated by the lighter colors. The symbols are: \( \% \) - wall, \( \text{P} \) - player, \( \text{G} \) - ghost, \( \text{H} \) - scared ghost, \( \cdot \) - coin, \( \circ \) - capsule.

Table 2: Manipulating Pacman behavior. Behavior and score of the Graph-DQN agent on the mediumClassic map when various edges are removed or features substituted. Reward is shown as mean± standard error over \( n = 100 \) repetitions.

<table>
<thead>
<tr>
<th>Variation</th>
<th>Reward</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base</td>
<td>1169 ± 39</td>
<td>Default behavior</td>
</tr>
<tr>
<td>Remove Ghost→Player edge</td>
<td>-170 ± 27</td>
<td>No clear interpretation</td>
</tr>
<tr>
<td>Set Ghost→Player to Player→Coin feature</td>
<td>-78 ± 38</td>
<td>Does not avoid ghosts</td>
</tr>
<tr>
<td>Remove Player→Scared Ghost edge</td>
<td>558 ± 25</td>
<td>Does not chase scared ghosts</td>
</tr>
<tr>
<td>Remove Ghost→Scared Ghost edge</td>
<td>1161 ± 29</td>
<td>No effect</td>
</tr>
<tr>
<td>Remove Player→Coin edge</td>
<td>-376 ± 20</td>
<td>Pacman moves randomly</td>
</tr>
<tr>
<td>Remove Player→Capsule edge</td>
<td>323 ± 37</td>
<td>Does not eat the capsule</td>
</tr>
<tr>
<td>Remove Player→Wall edge</td>
<td>-339 ± 21</td>
<td>Runs into the nearest wall</td>
</tr>
<tr>
<td>Remove Ghost→Wall edge</td>
<td>267 ± 33</td>
<td>No clear interpretation</td>
</tr>
<tr>
<td>Remove Scared Ghost→Wall edge</td>
<td>530 ± 28</td>
<td>Does not chase scared ghosts</td>
</tr>
</tbody>
</table>

the edge features of \( A \rightarrow B \) and \( A \rightarrow b \), and the agent attempts to push the bucket into the ball. The knowledge graph could also be manipulated such that the agent pushes a ball into another ball (Supplement). These studies show that the agent learned the ‘push’ and ‘fills’ relation and is able to apply these actions to objects it has never pushed before.

Similarly, in Pacman, if we remove the Player→Scared Ghost edge, the agent no longer chases the scared ghosts (Table 2). Without an edge to the capsule, the agent no longer eats the capsule. The agent can also be manipulated to not avoid ghosts by changing the Ghost→Player feature to the Player→Coin edge relation.

Figure 3: Manipulating Trained Agents in Warehouse. We used agents trained on the base knowledge graph (a), and manipulated their behavior at runtime by changing the input knowledge graph.

Conclusion: The field has long debated the importance of reasoning with symbols (that may incorporate prior knowledge) and its compatibility with gradient based learning. Graph-DQN provides one framework to bridge these seemingly disparate approaches \cite{Garnelo2019}.
REFERENCES


Brenden M. Lake, Tomer D. Ullman, Joshua B. Tenenbaum, and Samuel J. Gershman. Building machines that learn and think like people. *Behavioral and Brain Sciences*, 40, Nov 2016. ISSN 1469-1825. doi: 10.1017/s0140525x16001837. URL [http://dx.doi.org/10.1017/s0140525x16001837](http://dx.doi.org/10.1017/s0140525x16001837).


