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., 2018) 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 $\langle subject, relation, object \rangle$ 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 $\mathcal{K} = (\mathcal{V}, \mathcal{E})$ is a directed graph provided as vertices for each symbol in the environment (for subjects and objects), $\mathcal{V} = \{v_1, v_2, v_3, v_4, \ldots\}$ initially encoded as one-hot vectors of length $|\mathcal{V}|$, and edge features $\mathcal{E} = \{e_{12}, e_{13}, e_{23}, \ldots\}$. 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:

$$v_i = \sum_{v_j \in \mathcal{N}(v_i)} \Theta[e_{ij}] v_j + b \tag{1}$$

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 $\mathcal{V} \in \mathbb{R}^{|\mathcal{V}| \times d}$

2.3 MODEL COMPONENTS

We introduce several operations for transferring information between the state representation $\mathcal{S}$ and the knowledge graph $\mathcal{K}$. 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 $(\mathcal{S}, \mathcal{K})$, 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 10×10 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, $\{b, c\} \rightarrow B$ means $b \rightarrow B$ and $c \rightarrow B$ are rewarded.

Name	Training Pairs	Test Pairs
one-one	$b \rightarrow B$	$c \rightarrow B$
two-one	$\{b, c\} \rightarrow B$	$d \rightarrow B$
five-two	$\{b, c, d, e, f\} \rightarrow B$	$\{g, h\} \rightarrow B$
buckets	$b \rightarrow B$, $c \rightarrow C$, $d \rightarrow D$, $e \rightarrow E$, $f \rightarrow F$	$g \rightarrow G$, $h \rightarrow H$, $i \rightarrow I$, $j \rightarrow J$, $k \rightarrow K$
buckets-repeat	$\{b, c, d\} \rightarrow B$, $\{e, f, g\} \rightarrow C, \ldots,$ $\{n, o, p\} \rightarrow F$	$\{q, r, s\} \rightarrow G$, $\{t, u, v\} \rightarrow H$, $\ldots, \{6, 7, 8\} \rightarrow K$

by changing the number of ball-bucket pairs, the complexity of the grouping, and the number of unseen objects. The buckets-repeat is a challenging environment, with complex relationships in the test environment. The agent is identified as the `A` symbol, and the walls with `+`. For each variation, we generated 100 training mazes, and 20 testing mazes, randomly varying the location of the agent, ball(s), and bucket(s) in each maze. The agent received a reward of 3.0 for a successful pairing, and a penalty of -0.1 for each time step taken.

3.2 Symbolic Pacman

We test the agents on the smallGrid, mediumGrid, mediumClassic, and capsuleClassic environments from the text-based Pacman implementation. The environments differed in the size of the map as well as the numbers of ghosts, coins, and capsules present. We used random ghosts (as opposed to ghosts seeking out the agent). The agent received +10 points for eating a coin, +200 for eating a ghost, +500 for finishing the coins, -500 for being eaten, and -1 for each move.

3.3 Knowledge graph construction

For both environments, we add all entities to the knowledge graph with the exception of blank spaces. We then add edges between objects to reflect relationships present in the game structure. Each entity or edge type is assigned a unique one-hot vector; two pairs of entities may be connected by the same edge type. The Warehouse games have edges similar to those shown in Supplemental Figure 3, with an edge feature of '1' from the agent to all balls to encode a 'pushes' relationship; edge feature of '2' between all rewarded ball-bucket pairs; and an edge feature of '0' between the agent and impassable objects: the bucket(s) and the wall symbol. While we attach semantic meaning to these edge categories, their utility is grounded by the model during training. In Pacman, we add an 'impassable' relation from all the agents (player, ghost, and scared ghost) to the wall. We also add distinct edges from the player to all edible entities and agents (coin, capsule, scared ghost, ghost).

4 Results

We compared Graph-DQN to a baseline DQN in the Warehouse and Pacman environments. We also explore the effect of modifying the knowledge graph during training and at test time.

4.1 Warehouse

In the Warehouse environment, the Graph-DQN model was more sample efficient during training than the baseline Conv-DQN algorithm, as shown in Figure 1. For example, in the one-one environment, our model required approximately 8x fewer samples to reach the solution in the training environment (compare blue and green curves in the top row). In addition, in more complex environments with more possible objects and ball-bucket pairings, the baseline Conv-DQN required increasingly more samples to solve, whereas the Graph-DQN solved in the same number of samples.

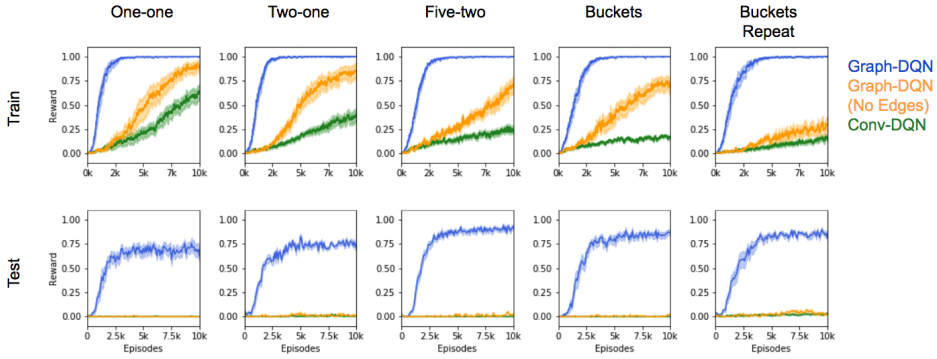

Figure 1: Warehouse results. For the environments described in Table 1 (columns), performance of the baseline DQN (green), our proposed Graph-DQN (blue), and a variant of Graph-DQN with edges removed (orange) over the number of training episodes. The top row shows the success rate (fraction completed 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 3a. 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

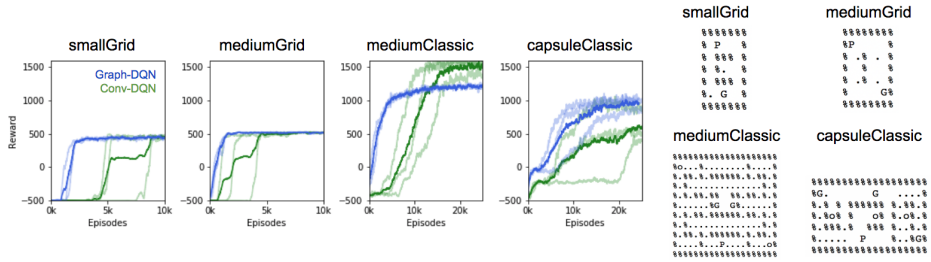

Figure 2: Pacman results. Performance of the baseline Conv-DQN (green) and GraphDQN (blue) agent on several symbolic Pacman environments (smallGrid, mediumGrid, mediumClassic, and capsuleClassic). Bold lines are the mean, with the individual $n = 3$ repetitions indicated by the lighter colors. The symbols are: `%` - wall, `P` - player, `G` - ghost, `H` - scared ghost, `.` - coin, `o` - 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.

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

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.

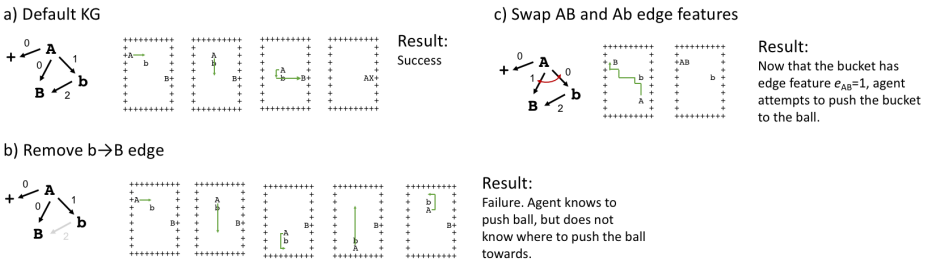

Figure 3: Manipulating Trained Agents in Warehouse. We used agents trained on the base knowledge graph (a), and manipulated their behavior at runtime by changing the input knowledge graph.

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 (Garnelo & Shanahan, 2019).

REFERENCES

Prithviraj Ammanabrolu and Mark O. Riedl. Playing text-adventure games with graph-based deep reinforcement learning. *CoRR*, abs/1812.01628, 2018.

Michael Beetz, Moritz Tenorth, and Jan Winkler. Open-EASE – a knowledge processing service for robots and robotics/AI researchers. In *IEEE International Conference on Robotics and Automation (ICRA)*, Seattle, Washington, USA, 2015. Finalist for the Best Cognitive Robotics Paper Award.

Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. Quantifying generalization in reinforcement learning. *arXiv preprint arXiv:1812.02341*, 2018.

Coline Devin, Pieter Abbeel, Trevor Darrell, and Sergey Levine. Deep object-centric representations for generalizable robot learning. *CoRR*, abs/1708.04225, 2017. URL http://arxiv.org/abs/1708.04225.

Rachit Dubey, Pulkit Agrawal, Deepak Pathak, Thomas L. Griffiths, and Alexei A. Efros. Investigating human priors for playing video games. arXiv:1802.10217, 2018.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *CoRR*, abs/1703.03400, 2017. URL http://arxiv.org/abs/1703.03400.

Marta Garnelo and Murray Shanahan. Reconciling deep learning with symbolic artificial intelligence: representing objects and relations. *Current Opinion in Behavioral Sciences*, 29:17–23, Oct 2019.

Michael Janner, Sergey Levine, William T. Freeman, Joshua B. Tenenbaum, Chelsea Finn, and Jiajun Wu. Reasoning about physical interactions with object-centric models. In *International Conference on Learning Representations*, 2019.

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.

Doug Lenat, Mayank Prakash, and Mary Shepherd. CYC: Using common sense knowledge to overcome brittleness and knowledge acquistion bottlenecks. *AI Mag.*, 6(4):65–85, January 1986. ISSN 0738-4602. URL http://dl.acm.org/citation.cfm?id=13432.13435.

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. Human-level control through deep reinforcement learning. *Nature*, arXiv:1312.5602(7540):529–533, 2015.

Alex Nichol, Vicki Pfau, Christopher Hesse, Oleg Klimov, and John Schulman. Gotta learn fast: A new benchmark for generalization in rl. arXiv:1804.03720, 2018.

Charles Packer, Katelyn Gao, Jernej Kos, Philipp Krahenbuhl, Vladlen Koltun, and Dawn Song. Assessing generalization in deep reinforcement learning. arXiv:1810.12282, 2018.

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

Ashutosh Saxena, Ashesh Jain, Ozan Sener, Aditya Jami, Dipendra K. Misra, and Hema S. Koppula. RoboBrain: Large-scale knowledge engine for robots. arXiv:1412.0691, 2014.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, arXiv:1712.01815:1140–1144, 2018.

Martin Simonovsky and Nikos Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs. *CoRR*, abs/1704.02901, 2017. URL http://arxiv.org/abs/1704.02901.

Thomas Stepleton. The pycolab game engine, 2017. URL https://github.com/deepmind/pycolab.

Dequan Wang, Coline Devin, Qi-Zhi Cai, Fisher Yu, and Trevor Darrell. Deep object centric policies for autonomous driving. *CoRR*, abs/1811.05432, 2018. URL http://arxiv.org/abs/1811.05432.

Vinicius Zambaldi. Deep reinforcement learning with relational inductive biases. In *International Conference on Learning Representations*, 2019.