SUPPLEMENT TO 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 RELATED WORK

1.1 GRAPH BASED REINFORCEMENT LEARNING

Previous work has applied graph-based architectures to control problems (Wang et al., 2018) and text-based games (Ammanabrolu & Riedl, 2018). In each case, the design of the architecture is guided by the problem structure. In NerveNet (Wang et al., 2018), the state and action spaces can be decomposed according to the anatomy of the agent; the Graph Neural Network in that study therefore also represents the anatomy. The KG-DQN agent (Ammanabrolu & Riedl, 2018) is targeted at games in which building a description of the relations of the world from text based interaction is a challenge. In contrast, we used the knowledge graph to inject semantic priors into the Graph-DQN, and trained agents to operate on visual input and achieve much larger relative gains in performance.

Whereas KG-DQN combines the knowledge graph and the state with feature concatenation in the final layer, here we use broadcast and joint convolution operations to exchange information between the entities in the graph and instances of that entity in the game state. We repeat this module in a deep network to form the encoder. By pooling state features into the graph and performing convolution, our model implements a global operation similar to the self-attention layer used in the Relational RL architecture (Zambaldi et al., 2019). However, that model tackles the problem of learning relational knowledge during training, without any *a priori* knowledge. Graph-DQN is designed to exploit external knowledge to generalize to new objects at test time.

1.2 NATURAL LANGUAGE GUIDED RL

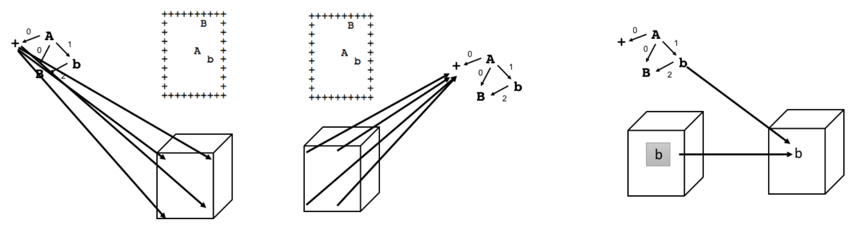
Previous work has used natural language instruction to train agents (Ammanabrolu & Riedl, 2018; Co-Reyes et al., 2018; Fu et al., 2019; Kaplan et al., 2017). Our approach is complementary to these as the information extraction algorithms in the literature (for e.g. (Angeli et al., 2015)) could be used to structure natural language corpora, rules or instructions as knowledge graphs, which can be provided to Graph-DQN. We believe that graphs could more generally serve as an efficient and interpretable representational mechanism for prior knowledge or instruction.

Several studies have attempted to extract objects from visual input using unsupervised or semi-supervised methods (Chen et al., 2016; Cheung et al., 2014; Higgins et al., 2016; 2017; Ionescu et al., 2018; Kingma & Welling, 2013; Locatello et al., 2018). As the focus of our study is the combination of scene graphs and knowledge graphs, and not the extraction of symbols themselves, we assume that our network has object level ground truth information available from the scene. For this reason we use environments that can be programatically generated.

1.3 METALEARNING

Several prior studies study generalization to novel tasks. Techniques such as MAML (Finn et al., 2017) train on a set of environments or tasks and attempt to learn a weight manifold on which a new task can be learned with minimal additional examples. Here we structure the relevant information

[*]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.



`Broadcast` features from the knowledge graph to the state based on symbol presence.

`Pooling` features from state grid into the knowledge graph. Note: we only show some connections from the wall symbol.

`KG-Conv`. In addition to the connections created by applying a filter to the neighborhood, we also add a connection from the entity present at the point in the output feature map.

Figure 1: Layer types in our Graph-DQN architecture, including methods to `Broadcast` from the knowledge graph $\mathcal{K}$ to the state presentation $\mathcal{S}$, `Pooling` from $\mathcal{S} \to \mathcal{K}$, and updating the state by jointly convolving over $(\mathcal{S}, \mathcal{K})$.

for solving the task in the form of a knowledge graph to avoid needing additional training. In environments where such prior knowledge may be unavailable, the Graph-DQN could be augmented with metalearning or curriculum learning. Other techniques such as DARLA (Higgins et al., 2017) and progressive nets (Rusu et al., 2016) have studied generalization in the context of domain shifts (e.g. from simulation to reality) but not in the context of generalization to new objects studied here.

2 Graph-DQN

2.1 Details of Model components

Figure 1 shows graphical depictions of the three contributed methods (`Broadcast`, `Pooling`, and `KG-Conv` for transferring information between the knowledge graph and the scene representation.

2.1.1 Broadcast

We define the function `Broadcast` : $\mathcal{K} \to \mathcal{S}$. For each entity i in the knowledge graph, we copy its graph representation v_i to each occurrence of i in the game map. This is used to initialize the state representation $\mathcal{S}$ such that we are using a common embedding to refer to entities in both $\mathcal{K}$ and $\mathcal{S}$. Each location (i, j) in the state is computed as

$$\mathcal{S}_{i,j} = \sum_{v \in \mathcal{V}} \delta_v(i,j) v \tag{1}$$

where $\delta_v(i,j) = 1$ if the entity corresponding to v is present at location (i,j) and zero otherwise. Thus, symbols in the game map not present in the knowledge graph are initialized with a zero vector.

2.1.2 Pooling

The reverse of `Broadcast`, this operation is used to update the entity representations in the knowledge graph. In `Pooling` : $\mathcal{S} \to \mathcal{K}$, we update the graph's representation v by averaging the features in $\mathcal{S}$ over all instances of entity corresponding to v in the state:

$$v_i = \frac{1}{N_v} \sum_{(i,j) \in \mathcal{S}} W \delta_v(i,j) \mathcal{S}_{ij} \tag{2}$$

where $N_v = \sum_{\mathcal{S}} \delta_v(i,j)$ is the number of instances of v in the state. Since $\mathcal{S}$ and $\mathcal{V}$ may have different number of features, we used the weight matrix W to project from the state vectors to the dimensionality of the vertex features in the graph.

2.1.3 KG-Conv

To update the state representation $\mathcal{S}$, we augment a regular convolution layer with the knowledge graph. In addition to applying convolutional filters to the neighborhood of a location, we also add the node representation v_i of the entity i at that location, passed through linear layer to v_i to match the number of filters in the convolution. Formally, we can describe this operation as:

$$\text{Conv}_{3\times 3\times d}(\mathcal{S}) + \text{Conv}_{1\times 1\times d}(\texttt{Broadcast}(\mathcal{K})) \tag{3}$$

This provides a skip connection allowing deeper layers in the network to more easily make use of the global representations.

2.2 Model Architecture

The model architecture is shown in Figure 2. First, we apply two layers of edge-conditioned graph convolution (ECC) (Simonovsky & Komodakis, 2017) to $\mathcal{K}$ to enrich the node features with information from the neighborhood of each node. Those features are then encoded in the state representation $\mathcal{S}$ through a `Broadcast` layer. The network then consists of N repeated blocks. In each block, we first update the knowledge graph with `Pooling` from the state, followed by graph convolutions. Then, we update the state representation with a `KG-Conv` layer, which incorporates the updated knowledge graph. Finally, we use a few linear layers to compute the Q-values for each action. While this model is designed to scale to deeper blocks, in our experiments we only use $N = 1$ blocks.

3 Supplemental information on Experiments

3.1 Baseline DQN

We used DQN (Mnih et al., 2015) as the baseline RL algorithm. To keep the comparison fair, the DQN also received symbolic input. In the Warehouse experiments, we used a convolutional network consisting of $Conv(3 \times 3, 64) \rightarrow Conv(3 \times 3, 64) \rightarrow Dense(64) \rightarrow Dense(4)$. This model is equivalent to the Graph-DQN architecture with the connections from the knowledge graph removed. We performed an architecture search and did not find a model that outperformed it.

The best agent in Pacman had a deeper and wider convolutional network with four $Conv(3 \times 3)$ layers with $(64, 128, 128, 64)$ filters, followed by a multilayer perception of $Dense(100) \rightarrow Dense(50) \rightarrow Dense(4)$.

In both models, after the convolutional layers, we computed a per-channel mean over the 2D map and passed the resulting vector into the multilayer perceptron (MLP).

We validated our implementation of the algorithm by comparing our performance on the Cartpole and Pong environments with those in Coach (Caspi et al., 2017) and Ptan (Lap). Software was implemented in Pytorch (Paszke et al., 2017). OpenAI Gym (Bro) and *pycolab* (Stepleton, 2017) were used to implement the environments.

3.2 Hyperparameters

We ran our experiments using the Adam optimizer with learning rate of 0.0001 in the Warehouse environments and 0.00025 in Pacman (Kingma & Ba, 2015). We used a replay buffer size of 100,000 throughout; at every step, we sampled 32 transitions from the buffer and trained the agent by minimizing the L_2 loss. In the Warehouse environments, we allowed the agent to run for 10,000 steps before commencing training.

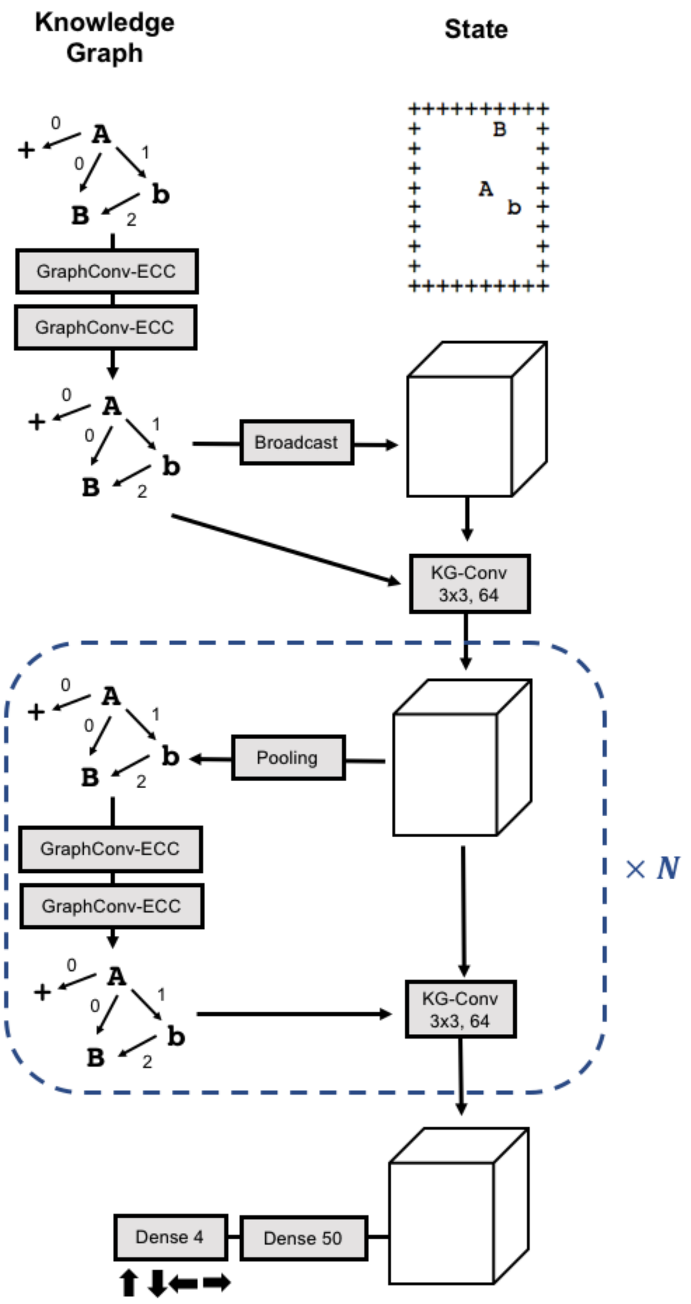


Figure 2: Graph-DQN architecture. The knowledge graph is first operated on by graph convolution layers (GraphConv-ECC) to enrich the node features to $d = 64$ dimensions (Simonovsky & Komodakis, 2017). We then use `Broadcast` to create a compatible scene representation $S \in \mathbb{R}^{10\times10\times d}$, indicated here by the cube. We then apply N blocks (blue dotted rectangle). In each block, we first update the knowledge graph $\mathcal{K}$ via `Pooling` and Graph Convolution. Then, we update $\mathcal{S}$ using a joint convolution over $\mathcal{S}$ and $\mathcal{K}$. Finally, a small MLP computes the Q-values for the actions.

4 Supplemental Results

4.1 Knowledge graph types

In order to determine whether the results in Figure 1 (in extended abstract) are sensitive to the choice of the knowledge graph architecture, we trained the Graph-DQN model with variants of the base knowledge graph, as shown in Figure 3:

1. Base: Knowledge graph was cropped to entities that were present in the scene and the edges between them
2. Same edges: Knowledge graph had same edge labels for all edges. No distinct relationships were encoded, but connectivity was same as the base knowledge graph
3. No edges: All edges were removed.
4. Fully connected: Nodes in the knowledge graph were fully connected, with the same edge label for all edges.
5. Fully connected (distinct): Nodes in the knowledge graph were fully connected, with distinct edge labels for each edge.
6. Complete: Complete knowledge graph is provided for both training and testing. No cropping to entities only present in the scene was performed.

When we removed the edge distinctiveness ('Same Edges'), the model still trained, but failed to generalize to novel objects. If we removed edges entirely ('No Edges'), the performance is the same as the baseline DQN. These results show that encoding the game structure into the knowledge graph is important for generalizing to the test environment but not necessary for the training environments.

Surprisingly, when the knowledge graph is fully connected ('FC' and 'FC-distinct'), the model does not train, suggesting that the prior structure cannot be learned by Graph-DQN. If the complete graph is available during training, including nodes for objects that only appear in the test environments, the model generalizes to near-optimal performance (see orange lines in 'Complete'). In this condition, even though the object 'c' is not in the training environment, gradients still flow through the $A \rightarrow c$ edge. To avoid any contamination during training into the knowledge graph of information about ball-bucket object pairs seen during test, for the base condition we crop the knowledge graph only to entities (and corresponding edges) seen in the training environments.

4.2 What does the Warehouse agent learn?

In addition to Pacman, we also ran experiments with Warehouse where we took an agent trained on the knowledge graph, and observed its behavior when the input knowledge graph was altered (Figure 4). We were able to manipulate the agent behavior, and confirm that the learned edge semantics match the game structure and can be applied to novel objects. Just by changing the knowledge graph at test time, the agent can be manipulated to push buckets into balls, or push balls into other balls.

5 Discussion

Here we show that using knowledge graphs to provide DQNs information about entities and their inter-relationships provides a way to facilitate faster learning. In addition, this provides a framework for faster generalization to new entities with similar relationships. We tested this generalization with sampling from various numbers of object pairs in train and test environments. We compared the Graph-DQN method to baselines with Conv-DQN or Graph-DQN without edges in the knowledge graph. Graph-DQN significantly outperformed the baseline, and was able to generalize to novel objects.

Our approach is complementary to other approaches in RL that strive to improve sample efficiency and generalization such as hierarchical RL (Kulkarni et al., 2016), metalearning (Finn et al., 2017), or better exploration policies (Ecoffet et al., 2018) and can be combined as such with these approaches to build better overall systems.

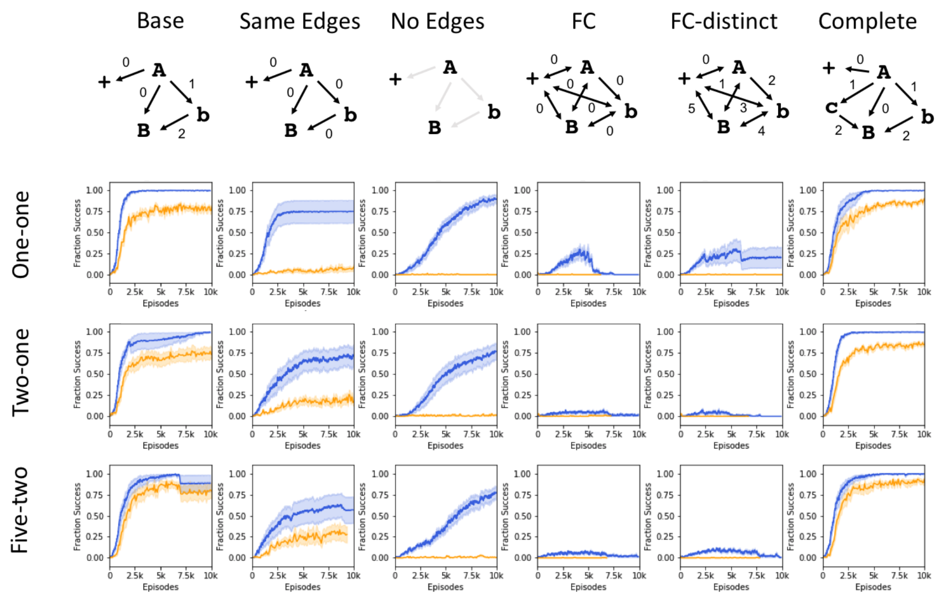


Figure 3: Model performance when trained on various knowledge graph types, in the one-one, two-one, and five-two environments in our most challenging Warehouse environment (buckets-repeat). Tested types are described in the paper. Train performance in blue, and test performance in orange. Shaded errors indicate the standard error over $n = 10$ repetitions. Results in other environments are similar, but omitted for space reasons. See Supplement for additional environments.

For future work, ablation studies on Graph-DQN can help determine when and how the scene and knowledge graphs should exchange information. Attempts to learn the knowledge graph during training were not successful (see ’fully connected’ in Figure 3), and we speculate that graph attention models (Velickovic et al., 2018) could help prune the graph to only the useful relations. We used simple one-hot edge features throughout, whereas one could use word embeddings (Mikolov et al., 2013; Sa et al., 2018) to seed the knowledge graph with semantic information. We could also test on previously published environments such as BoxWorld (Zambaldi et al., 2019), if code becomes available, as well as environments where multiple properties of entities need to be combined.

6 Future Directions

6.1 Scenes

The use of scene graphs could provide a framework to handle partial observability by building out portions of the environment as they are explored and storing them in the scene graph. As models that can extract objects from frames improve (Chen et al., 2016; Cheung et al., 2014; Higgins et al., 2016; Ionescu et al., 2018; Kingma & Welling, 2013; Locatello et al., 2018), connecting the outputs of these models as inputs to the models developed here could provide a mechanism to go directly from pixels to actions.

6.2 Interpretability

The knowledge graph provides an interpretable way to instruct the Deep RL system the rules of the game. While not explored here these rules could include the model of the environment facilitating use of Graph-DQN in model-based RL. Future work could explore whether the structure of the knowledge graph combined with the interpretability of the nodes and edges could serve as a mechanism to overcome catastrophic forgetting. For example, new entities and relationships could be incrementally added to the knowledge graph encoded in a way that is compatible with existing relationships and with potentially minimal disruption to existing entities and their relationships. A

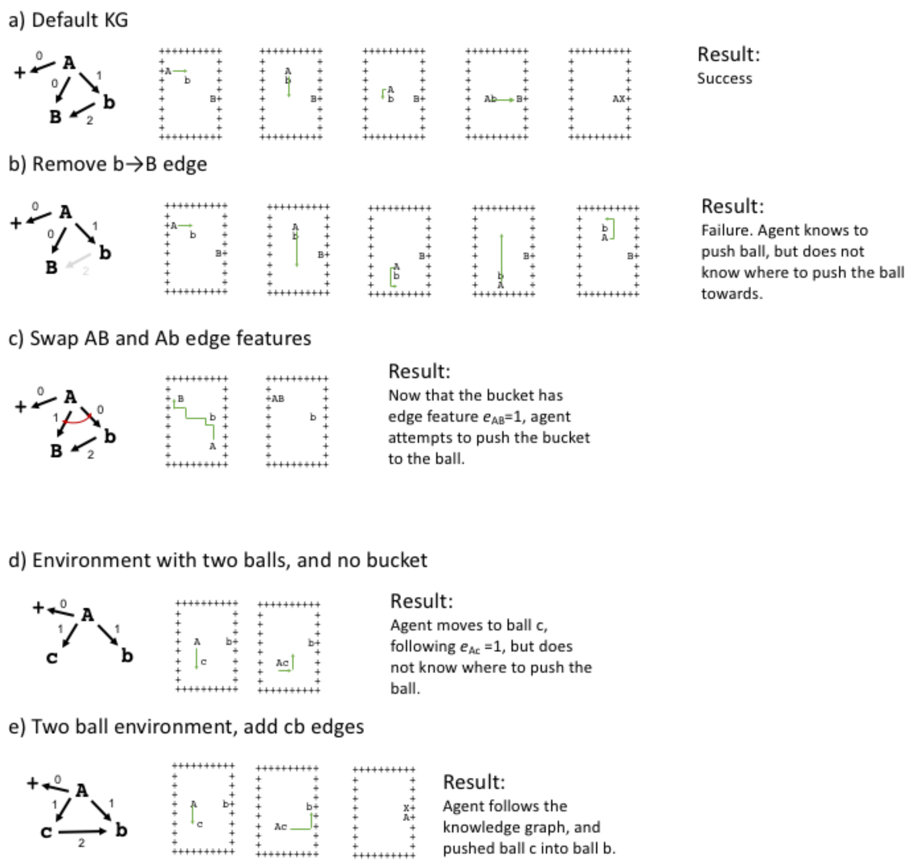

a) Default KG
Result:
Success
b) Remove b→B edge
Result:
Failure. Agent knows to push ball, but does not know where to push the ball towards.
c) Swap AB and Ab edge features
Result:
Now that the bucket has edge feature e_{AB}=1, agent attempts to push the bucket to the ball.
d) Environment with two balls, and no bucket
Result:
Agent moves to ball c, following e_{Ac} =1, but does not know where to push the ball.
e) Two ball environment, add cb edges
Result:
Agent follows the knowledge graph, and pushed ball c into ball b.

Figure 4: Manipulating agent behavior. We use an already trained agent, and manipulated its behavior at test time by modifying the input knowledge graph. For each manipulation, we show the resulting knowledge graph, the game state, and the resulting agent behavior. These studies show that the agent learned the semantic meaning of edges ('push', 'target') that we intended, and are able to apply those learned relations to different objects. For example, the trained agent can be manipulated to push buckets into balls, or balls into other balls without any additional training.

limitation is that even though the knowledge graph itself is interpretable, once the messages from the knowledge graph are combined with messages in the scene graph we sacrifice interpretability in favor of the learning power of gradient based Deep Learning.

6.3 KNOWLEDGE GRAPH

While we are hand coding the knowledge graph in this study, future work could learn the knowledge graph directly from a set of environments, or via information extraction approaches on text corpora, or learn graph attention models over existing large knowledge graphs (Angeli et al., 2015; Beetz et al., 2015; Bollacker et al., 2008; Lenat et al., 1986; Liu & Singh, 2004; Saxena et al., 2014; Suchanek et al., 2007). Knowledge graphs could also be generalized beyond the $\langle subject, relation, object\rangle$ triplet structure to incorporate prior or instructional information in the form of computational graphs (Abadi et al., 2016; Al-Rfou et al., 2016; Cyphers et al., 2018; Wolfram).

6.4 Environments

While we limited our analysis here to relatively small environments to test the fundamental aspects of our approach, scaling to larger environments is another obvious direction. Environments such as OpenAI Retro (Nichol et al., 2018) or CoinRun (Cobbe et al., 2018) have helped spark an interest in the problem of generalization in Deep RL. However, the lack of readily available ground truth and inability to programatically generate levels hinders a rigorous development of algorithmic approaches to solve this problem using Retro. We believe that further development of benchmarks for generalization in Deep RL (Packer et al., 2018) that enable programmatic game creation and make ground truth accessible will help the field.

References

Martn Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mane, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viegas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. arXiv:1603.04467, 2016.

Rami Al-Rfou, Guillaume Alain, Amjad Almahairi, Christof Angermüller, Dzmitry Bahdanau, Nicolas Ballas, Frédéric Bastien, Justin Bayer, Anatoly Belikov, Alexander Belopolsky, Yoshua Bengio, Arnaud Bergeron, James Bergstra, Valentin Bisson, Josh Bleecher Snyder, Nicolas Bouchard, Nicolas Boulanger-Lewandowski, Xavier Bouthillier, Alexandre de Brébisson, Olivier Breuleux, Pierre Luc Carrier, Kyunghyun Cho, Jan Chorowski, Paul F. Christiano, Tim Cooijmans, Marc-Alexandre Côté, Myriam Côté, Aaron C. Courville, Yann N. Dauphin, Olivier Delalleau, Julien Demouth, Guillaume Desjardins, Sander Dieleman, Laurent Dinh, Melanie Ducoffe, Vincent Dumoulin, Samira Ebrahimi Kahou, Dumitru Erhan, Ziye Fan, Orhan Firat, Mathieu Germain, Xavier Glorot, Ian J. Goodfellow, Matthew Graham, Çaglar Gülçehre, Philippe Hamel, Iban Harlouchet, Jean-Philippe Heng, Balázs Hidasi, Sina Honari, Arjun Jain, Sébastien Jean, Kai Jia, Mikhail Korobov, Vivek Kulkarni, Alex Lamb, Pascal Lamblin, Eric Larsen, César Laurent, Sean Lee, Simon Lefrançois, Simon Lemieux, Nicholas Léonard, Zhouhan Lin, Jesse A. Livezey, Cory Lorenz, Jeremiah Lowin, Qianli Ma, Pierre-Antoine Manzagol, Olivier Mastropietro, Robert McGibbon, Roland Memisevic, Bart van Merriënboer, Vincent Michalski, Mehdi Mirza, Alberto Orlandi, Christopher Joseph Pal, Razvan Pascanu, Mohammad Pezeshki, Colin Raffel, Daniel Renshaw, Matthew Rocklin, Adriana Romero, Markus Roth, Peter Sadowski, John Salvatier, François Savard, Jan Schlüter, John Schulman, Gabriel Schwartz, Iulian Vlad Serban, Dmitriy Serdyuk, Samira Shabanian, Étienne Simon, Sigurd Spieckermann, S. Ramana Subramanyam, Jakub Sygnowski, Jérémie Tanguay, Gijs van Tulder, Joseph P. Turian, Sebastian Urban, Pascal Vincent, Francesco Visin, Harm de Vries, David Warde-Farley, Dustin J. Webb, Matthew Willson, Kelvin Xu, Lijun Xue, Li Yao, Saizheng Zhang, and Ying Zhang. Theano: A python framework for fast computation of mathematical expressions. *CoRR*, abs/1605.02688, 2016. URL http://arxiv.org/abs/1605.02688.

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

Gabor Angeli, Melvin Jose Johnson Premkumar, and Christopher D. Manning. Leveraging linguistic structure for open domain information extraction. 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)*, pp. 344–354. Association for Computational Linguistics, 2015. doi: 10.3115/v1/P15-1034. URL http://aclweb.org/anthology/P15-1034.

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.

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *In SIGMOD Conference*, pp. 1247–1250, 2008.

Itai Caspi, Gal Leibovich, Gal Novik, and Shadi Endrawis. Reinforcement Learning Coach, December 2017. URL `https://doi.org/10.5281/zenodo.1134899`.

Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. InfoGAN: Interpretable representation learning by information maximizing generative adversarial nets. arXiv:1606.03657, 2016.

Brian Cheung, Jesse A. Livezey, Arjun K. Bansal, and Bruno A. Olshausen. Discovering hidden factors of variation in deep networks. arXiv:1412.6583, 2014.

John D. Co-Reyes, Abhishek Gupta, Suvansh Sanjeev, Nick Altieri, John DeNero, Pieter Abbeel, and Sergey Levine. Guiding policies with language via meta-learning. arXiv:1811.07882, 2018.

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

Scott Cyphers, Arjun K. Bansal, Anahita Bhiwandiwalla, Jayaram Bobba, Matthew Brookhart, Avijit Chakraborty, Will Constable, Christian Convey, Leona Cook, Omar Kanawi, Robert Kimball, Jason Knight, Nikolay Korovaiko, Varun Kumar, Yixing Lao, Christopher R. Lishka, Jaikrishnan Menon, Jennifer Myers, Sandeep Aswath Narayana, Adam Procter, and Tristan J. Webb. Intel nGraph: An intermediate representation, compiler, and executor for deep learning. arXiv:1801.08058, 2018.

Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. Montezuma's Revenge solved by Go-Explore, a new algorithm for hard-exploration problems (sets records on Pitfall, too), 2018. URL `https://eng.uber.com/go-explore/`.

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`.

Justin Fu, Anoop Korattikara, Sergey Levine, and Sergio Guadarrama. From language to goals: Inverse reinforcement learning for vision-based instruction following. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=r1lq1hRqYQ`.

Irina Higgins, Loïc Matthey, Xavier Glorot, Arka Pal, Benigno Uria, Charles Blundell, Shakir Mohamed, and Alexander Lerchner. Early visual concept learning with unsupervised deep learning. *CoRR*, abs/1606.05579, 2016. URL `http://arxiv.org/abs/1606.05579`.

Irina Higgins, Arka Pal, Andrei A. Rusu, Loïc Matthey, Christopher Burgess, Alexander Pritzel, Matthew Botvinick, Charles Blundell, and Alexander Lerchner. DARLA: Improving zero-shot transfer in reinforcement learning. In *ICML*, 2017.

Catalin Ionescu, Tejas Kulkarni, Aron van den Oord, Andriy Mnih, and Vlad Mnih. Learning to control visual abstractions for structured exploration in deep reinforcement learning. *Deep Reinforcement Learning workshop, NeurIPS 2018*, 2018.

Russell Kaplan, Christopher Sauer, and Alexander Sosa. Beating Atari with natural language guided reinforcement learning. arXiv:1704.05539, 2017.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference for Learning Representations*, 2015.

Diederik P. Kingma and Max Welling. Auto-encoding variational Bayes. *CoRR*, abs/1312.6114, 2013.

Tejas D. Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Joshua B. Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *CoRR*, abs/1604.06057, 2016. URL `http://arxiv.org/abs/1604.06057`.

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`.

H. Liu and P. Singh. ConceptNet - a practical commonsense reasoning tool-kit. *BT Technology Journal*, 22(4):211–226, October 2004. ISSN 1358-3948. doi: 10.1023/B:BTTJ.0000047600.45421.6d. URL `http://dx.doi.org/10.1023/B:BTTJ.0000047600.45421.6d`.

Francesco Locatello, Stefan Bauer, Mario Lucic, Sylvain Gelly, Bernhard Schlkopf, and Olivier Bachem. Challenging common assumptions in the unsupervised learning of disentangled representations. arXiv:1811.12359, 2018.

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

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 Krhenbhl, Vladlen Koltun, and Dawn Song. Assessing generalization in deep reinforcement learning. arXiv:1810.12282, 2018.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. 2017.

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. URL `http://arxiv.org/abs/1606.04671`.

Christopher De Sa, Albert Gu, Christopher R, and Frederic Sala. Representation tradeoffs for hyperbolic embeddings, 2018.

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.

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`.

Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: A core of semantic knowledge. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pp. 697–706, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-654-7. doi: 10.1145/1242572.1242667. URL `http://doi.acm.org/10.1145/1242572.1242667`.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=rJXMpikCZ`.

Tingwu Wang, Renjie Liao, Jimmy Ba, and Sanja Fidler. NerveNet: Learning structured policy with graph neural networks. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=S1sqHMZCb`.

Wolfram. Mathematica. Champaign, IL, 2018.

Vinicius Zambaldi, David Raposo, Adam Santoro, Victor Bapst, Yujia Li, Igor Babuschkin, Karl Tuyls, David Reichert, Timothy Lillicrap, Edward Lockhart, Murray Shanahan, Victoria Langston, Razvan Pascanu, Matthew Botvinick, Oriol Vinyals, and Peter Battaglia. Deep reinforcement learning with relational inductive biases. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=HkxaFoC9KQ`.