Graph neural induction of value iteration

Andreea Deac [1 2] **Pierre-Luc Bacon** [1 2] **Jian Tang** [1 3]

Abstract

Many reinforcement learning tasks can benefit from explicit planning based on an internal model of the environment. Previously, such planning components have been incorporated through a neural network that partially aligns with the computational graph of value iteration. Such network have so far been focused on restrictive environments (e.g. grid-worlds), and modelled the planning procedure only indirectly. We relax these constraints, proposing a graph neural network (GNN) that executes the value iteration (VI) algorithm, across arbitrary environment models, with direct supervision on the intermediate steps of VI. The results indicate that GNNs are able to model value iteration accurately, recovering favourable metrics and policies across a variety of out-of-distribution tests. This suggests that GNN executors with strong supervision are a viable component within deep reinforcement learning systems.

1. Introduction

The goal of reinforcement learning (RL) (Mnih et al., 2015; Levine et al., 2016) is to derive a control strategy (a policy) with good long-term behaviour, such as the maximization of returns. The RL framework is highly abstract, encompassing many possible environments, including ones where rewards are very sparse. Sparse rewards imply that many action steps can be performed before any response is given from the environment, making credit assignment (determining the most meaningful actions in a sequence) very challenging for most RL approaches. This motivates the application of *model-based planning* (Schrittwieser et al., 2019; Pascanu et al., 2017; Rivlin et al., 2020), where an explicit model of the environment is used to perform reasoning steps.

Value Iteration Networks (VIN) (Tamar et al., 2016) and

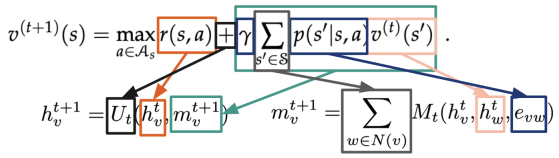

Figure 1. Correspondences between value iteration and graph convolution

Generalized Value Iteration Networks (GVIN) (Niu et al., 2018) propose architectures for deep reinforcement learning that incorporate an explicit planning component. While the planning module partially aligns with the computational graph of the value iteration planning algorithm (Bellman, 1957), there is no explicit guidance for the network to simulate it – the entire system is optimised via only a temporal difference loss (Sutton, 1988; Mnih et al., 2015). Recent work demonstrated that, when such information is available, steering neural networks towards appropriate step-level outputs can yield tangible benefits for out-of-distribution generalisation and multi-task learning (Xu et al., 2019; Veličković et al., 2019; Yan et al., 2020). Inspired by this research direction, our first proposal is directly supervising a neural network on the intermediate steps of value iteration.

Secondly, VIN is aimed at environments where each state may be represented by a pixel within a grid-world, leveraging the parallels between the computation of value iteration and image convolutions that arise in this setting. This is a clearly restricted setup, as general Markov Decision Process (MDP) states can have a variable number of possible successor states. GVIN generalises the setup to arbitrary graphs, but does not explicitly account for the correspondences between value iteration and the graph convolutional operator—making use of graph kernels (Yanardag & Vishwanathan, 2015) instead. We further make note of the fact that value iteration is a *dynamic programming* algorithm, and it was recently shown that graph neural network (GNN) computations algorithmically align with dynamic programming (Xu et al., 2019). Accordingly, our second proposal is leveraging GNNs that are specially aligned with the computations of the value iteration algorithm (Veličković et al., 2017; Gilmer et al., 2017).

[1]Mila - Quebec AI Institute [2]University of Montréal [3]HEC Montréal. Correspondence to: Andreea Deac <andreeadeac22@gmail.com>.

Proceedings of the 37^{th} International Conference on Machine Learning, Vienna, Austria, PMLR 119, 2020.

Table 1. MSE and accuracy for testing different GNNs.

Model	**MSE**			**Accuracy**		
	$\lvert\mathcal{S}\rvert = 20$ $\lvert\mathcal{A}\rvert = 5$	$\lvert\mathcal{S}\rvert = 50$ $\lvert\mathcal{A}\rvert = 10$	$\lvert\mathcal{S}\rvert = 100$ $\lvert\mathcal{A}\rvert = 20$	$\lvert\mathcal{S}\rvert = 20$ $\lvert\mathcal{A}\rvert = 5$	$\lvert\mathcal{S}\rvert = 50$ $\lvert\mathcal{A}\rvert = 10$	$\lvert\mathcal{S}\rvert = 100$ $\lvert\mathcal{A}\rvert = 20$
MPNN-Sum	0.457	2.175	5.154	97.75	99.3	99.32
MPNN-Mean	0.455	2.199	5.199	98.125	99.3	99.32
MPNN-Max	0.454	2.157	5.119	98.	99.25	99.22
MPNN-2-Sum	0.454	2.159	5.123	98.37	99.4	99.37
Attn-Sum	0.757	1.725	3.765	89.75	90.55	89.69

On both fronts, our results are largely positive: we demonstrate that GNNs can be used to simulate value iteration in a manner that generalises to out-of-distribution setups, such as different MDP transition topologies and action/state sizes. GNNs could also enable combining the planning and acting modules – the setup in this project is the first, model-based, step towards that (assuming perfect knowledge of MDP parameters). However, by considering the MDP model (edge information within the input graph) as information to be derived while also teaching the GNN to compute value iteration, the two modules could be combined towards a model-free algorithm, learning purely from trajectories.

2. Architecture

To specify our GNN architecture, we first describe the correspondences between value iteration and graph convolutions—refer to Figure 1 for a summary.

Firstly, we present the update rule of of value iteration, for an MDP with states $s \in \mathcal{S}$, actions $a \in \mathcal{A}$, transition model $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ and reward model $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$:

$$v^{(t+1)}(s) = \max_{a \in \mathcal{A}_s} r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a) v^{(t)}(s') \quad (1)$$

where $v^{(t)} : \mathcal{S} \to \mathbb{R}$ is the estimate of v^*, the optimal discounted cumulative return, at step $t \in \mathbb{N}$ of the algorithm, and $\gamma \in [0, 1)$ is a discount factor.

Message passing neural networks (MPNN) (Gilmer et al., 2017) represent the most generic form of graph convolution. They act upon a graph $G = (V, E)$, computing the representations of each node h_v by first computing a vector message from each of its neighbours. They then aggregate all incoming messages into a message vector m_v, as follows:

$$m_v^{t+1} = \sum_{w \in \mathcal{N}(v)} M_t(h_v^t, h_w^t, e_{vw}) \quad (2)$$

where M is the message function, usually a simple MLP. The next representation of node v is then obtained by further transforming the message together with a skip-connection:

$$h_v^{t+1} = U_t(h_v^t, m_v^{t+1}) \quad (3)$$

where U is a readout function, usually a simple MLP. The key property satisfied by MPNNs is *permutation invariance*: by aggregating neighbouring messages using a permutation-invariant function (such as summation), we guarantee that the results will be identical under all input graph isomorphisms. Note that other permutation-invariant aggregators can be used in Equation 2, and we also experiment with maximisation and averaging.

For each given MDP, we provide separate graphs G_{a_i} for each action a_i, using states s as nodes, and featurise them as follows:

- The input node features are defined as follows: $x_s = (v(s), r(s, a_i))$, containing the previous value function estimate and the reward model for taking a_i in s;
- The input edge features are defined as follows: $e_{s,s'} = (\gamma, p(s'|s, a_i))$, containing the discount factor and the transition model.

From this formulation, we can observe the following alignments between value iteration and MPNNs (cf. Figure 1):

- The MPNN message function M corresponds to taking a product of the edge features with the value-function estimate in the neighbours s';
- The aggregation over neighbours corresponds to taking a sum over s' with $p(s'|s, a) > 0$;
- The MPNN readout function U correspond to summing the reward model (in the source vertex) with the aggregated messages, in order to add the node's reward to the discounted neighbouring rewards;
- The maximisation over actions is aligned with taking the elementwise max over the computed $h_s^{(a_i)}$ within each G_{a_i}. The overall vector $h_s = \max_{a_i} h_s^{(a_i)}$ is then used to recompute the next-step value model: $v'(s) = f(h_s)$, where f is an MLP with a single-scalar output.

Table 2. MSE and accuracy for testing on unseen environments.

Model	**MSE**			**Accuracy**		
	$\lvert\mathcal{S}\rvert = 20$ $\lvert\mathcal{A}\rvert = 5$	$\lvert\mathcal{S}\rvert = 50$ $\lvert\mathcal{A}\rvert = 10$	$\lvert\mathcal{S}\rvert = 100$ $\lvert\mathcal{A}\rvert = 20$	$\lvert\mathcal{S}\rvert = 20$ $\lvert\mathcal{A}\rvert = 5$	$\lvert\mathcal{S}\rvert = 50$ $\lvert\mathcal{A}\rvert = 10$	$\lvert\mathcal{S}\rvert = 100$ $\lvert\mathcal{A}\rvert = 20$
Erdős-Rényi	0.457	2.175	5.154	97.75	99.3	99.32
Barabási-Albert	0.471	2.15	5.186	98.37	99.34	99.4
Star	1.77	2.317	5.324	100.	100.	100.
Caveman	1.452	2.302	5.285	98.12	98.84	99.05
Caterpillar	1.039	2.674	5.474	98.37	93.34	97.05
Lobster	0.928	2.776	5.507	97.5	92.09	95.02
Tree	1.01	2.672	5.494	94.25	95.59	95.19
Grid	0.564	2.511	5.416	92.62	91.65	91.3
Ladder	0.605	2.536	5.487	91.25	92.15	91.
Line	1.0375	2.831	5.643	88.12	88.4	89.34
	$\lvert\mathcal{S}\rvert \approx 20$ $\lvert\mathcal{A}\rvert = 8$			$\lvert\mathcal{S}\rvert \approx 20$ $\lvert\mathcal{A}\rvert = 8$		
Maze (Tamar et al., 2016)	4.95			69.86		

3. Experiments

For all experiments performed, we train on MDPs with transition models following[1] Erdős-Rényi graphs (Erdös et al.) with 20 states and 5 actions ($|\mathcal{S}| = 20$, $|\mathcal{A}| = 5$). For each iteration step t, we optimise the mean-squared error between the prediction $v'(s)$ and the ground-truth value obtained from value iteration at each state. Training is performed with teacher forcing (always predicting the outputs of one iteration from ground-truth inputs), while at test time we perform rollouts of the GNN, feeding back the predicted values $v'(s)$ as inputs in the next step until convergence.

Besides checking the MSE on the final values obtained from test rollouts w.r.t. v^*, we also compute the *policy accuracy*—checking, for each state, the overlap of the policy obtained by taking the argmax of Equation 1 for the predicted values v' and the optimal value v^*. Even if our MPNN model incorrectly estimates v^*, the predicted values could still result in identical policies if the relative differences between the entries of v' match the ones of v^*. In fact, there are many possible solutions v that respect the underlying Bellman equation in a way that recovers the optimal policy.

We aim to check that the model generalises to out-of-distribution samples, learning the VI algorithm and not nuances in the training data: therefore, we conduct out-of-distribution testing on graphs of sizes ($|\mathcal{S}| = 50$, $|\mathcal{A}| = 10$) and ($|\mathcal{S}| = 100$, $|\mathcal{A}| = 20$). Starting with the MPNN-Sum model as described in the previous section (using linear layers for M, U and f), we visualise the generalisation test results for varying state space size $|\mathcal{S}|$ in Figure 2. Here, we fix the number of actions and vary number the of states in $\{20, 50, 100\}$, plotting the MSE and policy accuracy against v^* as a function of MPNN iteration count. The results indicate that the model is robust in all cases in terms of recovered policy, and the iterations gracefully converge to a fixed loss.

The experiments in Table 1 explore the particularities in the GNN, in order to confirm that the architecture proposed in Section 2 is indeed appropriate. Besides MPNN-Sum, we also experiment with averaging (MPNN-Mean) and maximisation (MPNN-Max). The results demonstrate, however, that the model is robust to these changes. Moreover, a one-layer message function (M) might not have the ability to accurately model the product between γ and $p(s'|s, a)$, so we also experiment with a two-layer MLP as M (denoted as MPNN-2-Sum); this yields similar results to MPNN-Sum. Lastly, we note that the message could be used as a multiplicative factor in VI, which aligns it well with attention (Attn-Sum) (Veličković et al., 2017) as a way to scale neighbours, rather than using messages. While this operator indeed reduces the overall MSE, it also significantly reduces the recovered policy accuracy. We hypothesise that this is due to the fact that attending has scale-preservation built-in (and hence less chance of

[1] We define an MDP as *following* a graph distribution when the set of the nonzero entries of its transition model, i.e. $\mathcal{E} = \{(s, s') \mid p(s'|s, a_i) > 0\}$, can be described as generated from this distribution.

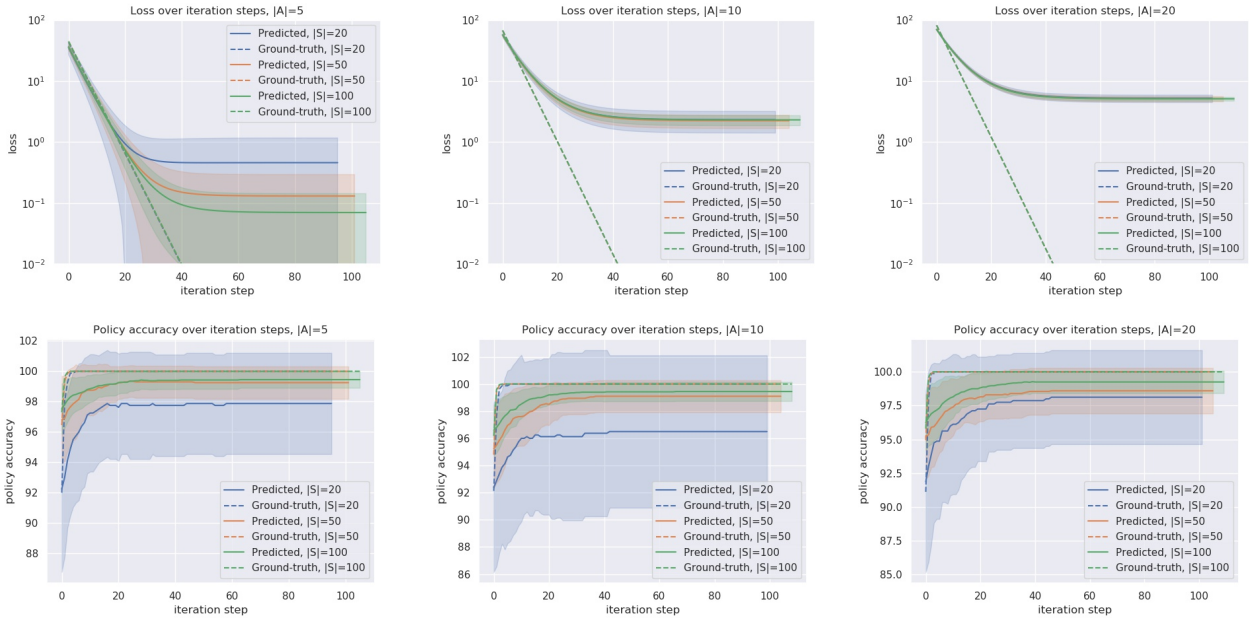

Figure 2. **Fixed** $|\mathcal{A}|$ – from left to right, the number of actions is 5, 10 and 20 respectively. The x-axis shows the number of iteration steps, while the y-axis represents the MSE in the first row and the accuracy in the second row. *Blue* is $|\mathcal{S}| = 20$, *orange* is $|\mathcal{S}| = 50$ and *green* is $|\mathcal{S}| = 100$. The continuous line corresponds to MPNN-Sum, while the dashed line corresponds to value iteration.

over-estimating v^*), but is still not powerful enough to model the relative values of v' in the best way.

We conclude with a generalisation test on the graph distribution that the MDP follows. After training the MPNN-sum model on Erdős-Rényi MDPs, we check for generalisation over different environments by testing on MDPs with different underlying graph distributions.

The results are enumerated in Table 2, and span the same graph distributions as in (Corso et al., 2020): from scale-free graphs (Barabási-Albert (Barabási & Albert, 1999)), through graphs with densely-connected patterns (Star, Caveman (Watts, 1999)) to sparse graphs (Caterpillar, Lobster, Tree, Grid, Ladder, Line). Finally, we evaluate the extent to which our VI executor can zero-shot generalise to a fully deterministic 8×8 maze environment as described in VIN (Tamar et al., 2016).

We can observe that the model exhibits a strong level of zero-shot generalisation to most unseen MDP graphs; better so when the graphs are more dense or more closely related to the training distribution. As the graphs get sparser, so do the MSE and policy accuracy degrade—dropping to 70% in the fully deterministic case. Such results are to be expected, as the MPNN was trained on dense graphs with edges and transitions sampled uniformly at random. If we can anticipate any properties of our test environments at training time (e.g. determinism), we may appropriately modify the training distribution to accommodate this.

4. Conclusions

We proposed a method for performing neural induction of value iteration using graph neural networks. By testing on out-of-distribution graph sizes and unseen environments, we show that the algorithm learnt by GNNs is robust w.r.t the number of states and on other types of random graphs. Out-of-distribution performance aligns with the structural similarities between training and testing MDPs in an expected manner, highlighting the potential of training a targeted executor provided we know certain characteristics of our test environments, for example sparsity, degree distribution and determinism.

Our work indicates that GNN executors can become a robust component in differentiable planning systems; however, further work remains to be done before they can be broadly applicable. GNNs could also enable the combination of planning and acting: by re-estimating the MDP model parameters from observed trajectory data, simultaneously with learning to execute value iteration, we can derive a model-free general purpose algorithm which combines latent graph inference with differentiable planning.

References

Barabási, A.-L. and Albert, R. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999.

Bellman, R. A markovian decision process. *Journal of mathematics and mechanics*, pp. 679–684, 1957.

Corso, G., Cavalleri, L., Beaini, D., Liò, P., and Veličković, P. Principal neighbourhood aggregation for graph nets. *arXiv preprint arXiv:2004.05718*, 2020.

Erdös, P. et al. On random graphs.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1263–1272. JMLR. org, 2017.

Levine, S., Finn, C., Darrell, T., and Abbeel, P. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529–533, 2015.

Niu, S., Chen, S., Guo, H., Targonski, C., Smith, M. C., and Kovačević, J. Generalized value iteration networks: Life beyond lattices. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Pascanu, R., Li, Y., Vinyals, O., Heess, N., Buesing, L., Racanière, S., Reichert, D., Weber, T., Wierstra, D., and Battaglia, P. Learning model-based planning from scratch. *arXiv preprint arXiv:1707.06170*, 2017.

Rivlin, O., Hazan, T., and Karpas, E. Generalized planning with deep reinforcement learning. *arXiv preprint arXiv:2005.02305*, 2020.

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., et al. Mastering atari, go, chess and shogi by planning with a learned model. *arXiv preprint arXiv:1911.08265*, 2019.

Sutton, R. S. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

Tamar, A., Wu, Y., Thomas, G., Levine, S., and Abbeel, P. Value iteration networks. In *Advances in Neural Information Processing Systems*, pp. 2154–2162, 2016.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Veličković, P., Ying, R., Padovano, M., Hadsell, R., and Blundell, C. Neural execution of graph algorithms. *arXiv preprint arXiv:1910.10593*, 2019.

Watts, D. J. Networks, dynamics, and the small-world phenomenon. *American Journal of sociology*, 105(2): 493–527, 1999.

Xu, K., Li, J., Zhang, M., Du, S. S., Kawarabayashi, K.-i., and Jegelka, S. What can neural networks reason about? *arXiv preprint arXiv:1905.13211*, 2019.

Yan, Y., Swersky, K., Koutra, D., Ranganathan, P., and Hashemi, M. Neural Execution Engines, 2020. URL https://openreview.net/forum?id=rJg7BA4YDr.

Yanardag, P. and Vishwanathan, S. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1365–1374, 2015.