Centralised Multi-Agent Reinforcement Learning By Leveraging Tractable Inference

Jian Vora
Computer Science Department, Stanford University
jianv@cs.stanford.edu

Abstract

Most previous works on multi-agent reinforcement learning deal with the case of centralised learning for decentralised execution of policies. One of the reasons why learning centralised policies is not preferred is because the joint action space grows exponentially with the number of agents (if each agent can take k actions, the joint action space with n agents is k^n). In this work, we aim to learn the joint policy for multi-agent systems with mild assumptions on the joint action-value function which shall allow us tractability. The joint policy is taken to be a sum-product network which allows for efficient inference.

1 Multi-agent Reinforcement Learning

1.1 Centralised Training for Decentralised Execution

Most MARL methods operate in this domain where during training time, we have a centralised critic and have observations of actions followed by each agent. This information is used to guide the learning of Q-functions of individual agents. The Q functions thus learnt have signal from actions taken by other agents as opposed to learning all Q functions independently assuming no interaction between the agents whatsoever. At test time, the joint action is taken by taking the argmax of the joint Q-function (Q_{tot}) over the joint action space. To ensure decentralised execution at test time, we need to ensure that this joint argmax splits into argmax of Q functions of each agents over their own action space. In other terms we want the following to hold true,

$$\underset{\mathbf{u}}{\operatorname{argmax}}\, Q_{tot}(\boldsymbol{\tau}, \mathbf{u}) = \begin{pmatrix} \operatorname{argmax}_{u^1} Q_1(\tau^1, u^1) \\ \vdots \\ \operatorname{argmax}_{u^n} Q_n(\tau^n, u^n) \end{pmatrix}$$

Figure 1: Factorisation needed for Centralised Training Decentralised Execution

The advantage of this particular approach is that at test time, we do not need any communication between the agents. All computation is done locally on the agent and they act actions greedily which maximised their own Q-function. This works as the joint Q-function is trained in a centralised manner and hence Q functions of individual agent are such that they have obtained some signal on what kinds of actions the other agents shall perform. Some common valued based methods used for performing centralised training for decentralised execution involve:

1. **Independent Q-Learning**: This baseline just considers all the agents to be independent of each other. At train time, each agents just optimises its own Q-function based on the states it has seen and actions it performs. At test time, the action the agent takes is simply greedily maximising its own learnt Q-function. Thus, all agents are left to themselves in order to achieve the given task which can be pretty sub-optimal for many environments.

2. **Value Decomposition Network** [10]: This involves factorising the joint Q-function as a sum of Q functions of individual agents. The joint Q-function is then trained to minimize the squared TD (temporal difference) error just like that in DQN. This particular factorisation ensures that the taking the argmax of the joint Q-function over the joint action space actually decomposes into argmax of indivudal Q functions as shown in 1.

$$\text{VDN [10]: } Q_{\text{tot}}(s, \mathbf{u}) = \sum_{i}^{n} Q_i(s_i, u_i)$$

3. **QMIX** [7]:This work is a generalisation of the VDN approach discussed earlier. We note that for the argmax property to hold true, taking the joint Q-function to be the sum of Q functions of every agent is rather restrictive. In fact, the authors of QMIX note that the joint Q-function can be any monotonic transformation of the action-value functions of each agent. Even in this particular form of factorisation, the argmax property shall hold true. If g_θ is any neural network with positive weights, then the QMIX factorization can be written in the following manner:

$$\text{QMIX [7]: } Q_{\text{tot}}(\tau, \mathbf{u}) = g_\theta(Q_1(\tau_1, u_1), ..., Q_n(\tau_n, u_n))$$

1.2 When do you not want to do decentralised control?

Decentralised execution of policies is attractive as it is neat and does not involve any interaction or communication between the agents. The hard problem of taking an argmax over a high-dimensional joint action is also solved by factorising the joint Q-function in a manner which makes finding the policy easy. However, despite its advantages, decentralised execution has some disadvantages as it does not really learn a "joint" policy. If at test time, if we knew what actions other agents shall be performing, then under the above framework, we cannot use that information and the action that an agent shall take is only by taking the argmax of its own Q-function greedily and all the agents are oblivious to this extra information which is available and hence may not action optimally. Secondly, as one might note, the above only takes argmax of the joint Q-function which means that the policy learnt is a deterministic policy.

In the recent max-ent RL framework as introduced in [5], it has been shown that learning energy-based stochastic policies useful for a couple of reasons - it provides for a natural way to explore the environment [4] and it also has been shown to learn robust policies in the sense that the agent can adapt faster to perturbations in the state space during test time [2]. This motivates us to actually learn a joint policy π over the joint action space which provides us the above benefits and at the same time enables us to do inference when we have some additional data available to us. In the subsequent section, we shall look into a way to learn a joint policy in the max-ent RL framework which shall provide us the benefits of the same discussed above.

2 Sum-Product Networks

In this section, we shall take a detour from the MARL problem and look at a class of generative models which allow us for tractable efficient inference. Sum-Product Networks are such generative models which were introduced in [6]. These models are rooted DAGs and have the following three building blocks:

1. **Leaf Nodes**: These nodes are essential the leaves of the computational graph and encode a distribution of the variables which are input at that particular leaf. For example, if a variable X_1 is an input to a leaf node, then the output shall be $p(X_1)$ where $p(.)$ is a parameterized distribution (gaussian/categorical etc.).
2. **Sum Nodes**: Sum node takes as input two or more weighted edges and their output is a weighted linear combination of it's children with the weights given by the edges. These weights of the model are learnable. For example, if $p(X_1)$ and $p(X_2)$ are input to the sum node, then the output shall be $w_1 p(X_1) + w_2 p(X_2)$ where w_1, w_2 are the weights of the respective edges.

3. **Product Nodes**: Product nodes simply output the product of all it's inputs. For example, if $p(X_1)$ and $p(X_2)$ are input to the product node, then the output shall be $p(X_1)p(X_2)$. As an example, the following figure shows how a typical sum-product network looks like:

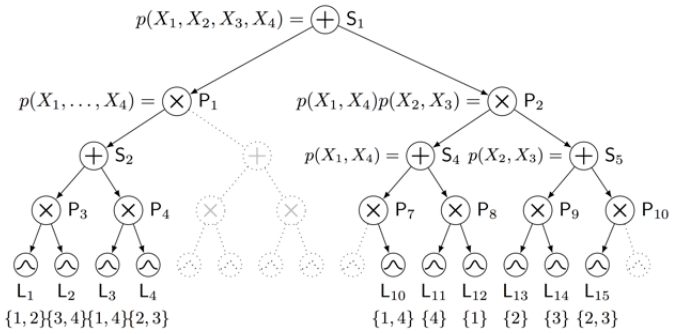

Figure 2: An example architecture of a sum-product network

2.1 Where do SPNs sit in the spectrum of generative models?

We want to perform inference such as marginalisation, conditioning etc. on the learnt joint distribution $p(x_1, x_2, \ldots, x_n)$ efficiently. The simplest models such a a gaussian mixture model, naive bayes allow answer the above inference queries efficiently. However, these toy models are not very expressive and cannot be used for hard tasks. On the other hand, neural density estimators such as GANs, VAEs, diffusion models can model very complicated interactions between variables but cannot answer the above inference queries efficiently. SPNs sit somewhere in the middle of this spectrum and allow for a good tradeoff between expressiveness and tractability. One can see that gaussian mixture models can be derived from SPNs if we do not have any product nodes and only a single sum node at the top of the computational graph. SPNs on the other hand are definitely not as expressive as neural network based methods but allow for tractable inference.
TL;DR - Sum-product networks are a class of generative models which allow for tractable inference for queries such as marginalisation, conditionals and likelihood evaluation but for achieving this, we lose a bit of expressive power.

3 Centralised Multi-agent Reinforcement Learning

3.1 Notation and Goal

Consider the case of cooperative multi-agent reinforcement learning where there are n agents and each of them can take one of k discrete actions. There exists a single reward function r for all the agents. If an agent i takes an action u_i, then we denote $\mathbf{u} = (u_1, u_2, \ldots, u_n)$ as the joint action. Since we are dealing with the case of centralised learning of policies, assume that we can somehow sense the global state s of the system which computes the joint policy and communicates each agent the action it has to perform. Thus, we aim to learn $Q_{\text{joint}}(s, \mathbf{u})$ and the joint policy $\pi(\mathbf{u}|s)$.

3.2 Applications

One can think of centralised execution as having a "super agent" which senses the global state of the system and then samples from the joint policy to instructs all the agents to perform their respective actions. Some applications where this might be useful include:

1. **Swarm Robotics**: There a many agents and we want to form a particular configuration for a task (and hence its a cooperative task). Here, a drone can easily sense the global position of each and every agent.
2. **Robotic Warehouses**: Use of robots in warehouses for inventory management. This is also a cooperative task and the global state can be determined by cameras installed in the warehouse.

Essentially, the kind of applications which we are looking at involve a lot of agents (to make the joint action space very high dimensional) and a setting where sensing the global state of the system is easy to do. One can even think of our method being applied to a robotic arm where each joint can be thought of as a different agent.

3.3 Inference on the joint policy

As motivated earlier, one reason why we want to learn the joint policy $\pi(\mathbf{u}|s)$ is because we can perform inference on top of it. Some of the inferences motivated from their applications are mentioned in the following points:

1. **Marginalisation**: We can marginalise out the actions of some agents in the joint policy if we do not have any information/care about those agents. One application could be if the agent loses communication with the super agent and we still need to continue to provide control to the other agents. Another application could be in a football game where a player shall not care about all players behind him and assume that all of them perform kind of reasonably on average.
2. **Conditioning**: If we have some information about what action some subset of agents are going to take, we can find the conditional distribution and sample from this particular distribution for the rest of the agents. One concrete application could be as follows - imagine an action "take no action"; this could be used to model an agent/robot which is non-operational and conditioned on this, we want the other agents to do the task optimally.
3. **Likelihood Evaluation**: We can evaluate the likelihood of a joint action taken under the learnt policy and this can help us solve many tasks such as anomaly detection (an agent not behaving the way we we have instructed it to).

In the subsequent section, we shall look into a method combining SPNs and the max-ent RL framework to find a joint policy. We also propose a particular type of factorisation for the joint Q-function which helps us to learn a joint policy despite the curse of dimensionality.

4 Method

We use an actor-critic update to find the Q-function Q_{joint} from the experience and we parameterize our joint policy $\pi(.|s)$ as an SPN. Inspired from the max-ent RL framework as shown in [5],

$$\pi(\mathbf{u}|s) \propto \exp(Q_{\text{joint}}(s,\mathbf{u})) \text{ from the relation } \pi(\mathbf{u}|s) = \exp(Q_{\text{joint}}(s,\mathbf{u}) - V_{\text{joint}}(s)) \tag{1}$$

Thus, given Q_{joint}, we need to find the normalizing constant to find the policy which is hard as it involves summation over $\mathbf{u}$ which can be of the order of $4^5 - 4^{10}$. However, as π is taken to be an SPN, we can perform variational inference exactly, i.e.,

$$\min D_{KL}(\pi(\mathbf{u}|s), \exp(Q_{\text{joint}}(s,\mathbf{u})) - V_{\text{joint}}(s)) = \min \mathbb{E}_{\mathbf{u}\sim\pi(\mathbf{u}|s)} \left(\log(\pi(\mathbf{u}|s)) - Q_{\text{joint}}(s,\mathbf{u})\right) \tag{2}$$

However, the above can be computed exactly only if Q_{joint} is a multi-linear function of actions. Thus, we now factor Q_{joint} using ideas from factor graphs and consider it to be a sum of binomials/trinomials based on the number of nodes in the factor graph. We factorise Q_{joint} in the following manner,

$$Q_{\text{joint}}(s,\mathbf{u}) = \sum_{i=1}^{i=P} f_\theta^{(i)}(s) u_l u_m \text{ or } Q_{\text{joint}}(s,\mathbf{u}) = \sum_{i=1}^{i=P} f_\theta^{(i)}(s) u_l u_m u_n \tag{3}$$

In the above equation, $f_\theta(.)$ can be any non-linear function such as a neural network. The indices l, m, n can be chosen at random for now and given that we choose P large enough, we ensure the coverage of all actions in representing the Q function. As an example, with 4 agents and $P = 4$,

$$Q_{\text{joint}}(s,\mathbf{u}) = f_\theta^{(1)}(s)u_1u_3 + f_\theta^{(2)}(s)u_2u_3 + f_\theta^{(3)}(s)u_1u_4 + f_\theta^{(4)}(s)u_2u_4$$

We minimise the following to get Q_{joint} : $\sum (Q_{\text{joint}}(s_t,\mathbf{u}_t) - r(s_t,\mathbf{u}_t) - \gamma \mathbb{E}_{s_{t+1}} V(s_{t+1}))^2$

Contrast this with other multi-agent RL methods which do things like (Q_i = neural network):

VDN [10]: $Q_{\text{tot}}(s,\mathbf{u}) = \sum_i^n Q_i(s_i,u_i)$ or QMIX [7]: $Q_{\text{tot}}(\tau,\mathbf{u}) = g_\theta(Q_1(\tau_1,u_1),...,Q_n(\tau_n,u_n))$

4.1 Related Work

As one might note, we are not really solving an MARL problem. Rather, we want to perform max-ent RL on high dimensional discrete action spaces where taking a softmax to find the policy is infeasible. Some related work which tries to find a joint policy and considers different types of factorisations include:

1. **Using Free Energies to Represent Q-values in a Multiagent Reinforcement Learning Task** [8]: This is the most related work to ours. They factorise the joint Q-function as a product of experts (bipartite restricted boltzmann machine) and use a energy-based policy. However, they compute the expectation by performing Gibbs sampling which is clearly not scalable to extremely high-dimensional problems.
2. **Coordinated Reinforcement Learning** [3]: This paper decomposes the joint Q-function into sum factors in which each factor only depends on actions of nearby agents which are atmost a few. In this way, taking argmax becomes easier because the action of an agent only appears in a few terms of the summation. argmax can be found by using ideas from variable elimination in the graphical models literature and hence the cost of finding argmax is reduced from exponential to polynomial time. For example, with 4 agents, the factorisation can be:
$$Q_{joint}(s, \mathbf{u}) = Q_1(s, u_1, u_2) + Q_2(s, u_2, u_4) + Q_3(s, u_3, u_1) + Q_4(s, u_1, u_4)$$
3. **Deep reinforcement learning in large discrete action spaces** [1]: This work essentially assumes that the action is continuous while training. Once a continous action prototype is generated, they search for k-nearest neighbours of it in the discrete grid of actual actions and find the approximate argmax of the Q-function over this plausible k actions only.
4. **Q-Learning in Enormous Action Spaces Via Amortized Approximate Maximization** [11]: In this work, they train an additional proposal network which learns a distribution over actions. Ones with larger mass are more likely to maximise the Q-function. Once we have a proposal network, we can simply sample a constant number of actions from it and check which of these samples maximises the Q-function which we are learning.

4.2 Some Comments on Multilinear Polynomials

As said earlier, we model Q_{joint} as a multilinear polynomial over actions with the weights coming from the state. We need to wonder what kind of assumptions we have taken in this particular form of modelling and how much expressive power do we give up in order to ensure tractability. For a recall, we factorise Q_{joint} in the following manner:

$$Q_{\text{joint}}(s, \mathbf{u}) = \sum_{i=1}^{i=P} f_\theta^{(i)}(s) u_l u_m \text{ or } Q_{\text{joint}}(s, \mathbf{u}) = \sum_{i=1}^{i=P} f_\theta^{(i)}(s) u_l u_m u_n$$

The crucial part in our control is the size of each monomial and the number of monomials P. Clearly, if P is exponential in n, then the above factorisation can model **any** function. As a simple example, consider a case of only 2 agents each taking one of two actions 0 or 1. The the following factorisation is completely general:

$$Q_{\text{joint}}(s, u_1, u_2) = f_\theta^{(1)}(s) u_1 u_2 + f_\theta^{(2)}(s) \bar{u_1} u_2 + f_\theta^{(3)}(s) u_1 \bar{u_2} + f_\theta^{(4)}(s) \bar{u_1} \bar{u_2}$$

However, clearly the above is computationally not feasible as computing the joint Q-function would be an exponential time algorithm. Instead, we approximate the general term with only a fixed number of terms. This is not a very bad approximation as it has been shown that Q-functions are typically **low-rank** [12]. Having only a fixed number of terms in the factorisation ensures this sparsity/low-rank constraint is taken care of and we do not lose much in expressive power.

5 Pros and Cons of the Method

1. **High-Dimensional Discrete Action Spaces**: The goal of using SPNs for performing variational inference especially for discrete action spaces where the famous "reparameterization trick" won't be applicable and alternative methods such as REINFORCE suffer from high variance problems. This is inspired from [9] and helps learn a policy in multi-agent settings.

2. **Inference on the Learnt Policy:** Once we learn $\pi(\mathbf{u}|s)$, we can perform inferences such as $\pi(u_i, u_k, .., u_l|s, u_a, u_b, ..., u_c)$ (how should other agents behave given we know actions of a subset of agents) and this is easy to do as π is an SPN. This shall also be extended to if an agent is non-operational and we can marginalise out it's action from the joint policy.

3. **Expressiveness**: We lose some expressiveness for tractability as we implicitly force the policy π to be log-multilinear in the action space.

4. **Centralised Execution**: This requires us access to a "super agent" or a cloud which senses the global state at each time step and we also have to deal with communication bottlenecks.

6 Experiments

We shall start off with simple multi-agent gridworld settings. One setting is to have n goals and n agents and all the agents get a reward of 1 if all the n agents reach any of the n goals and cover all the n goals as well (and hence it is a cooperative MARL problem). In the gridworld, each agent can take 6 actions hence the action space is 6^n.

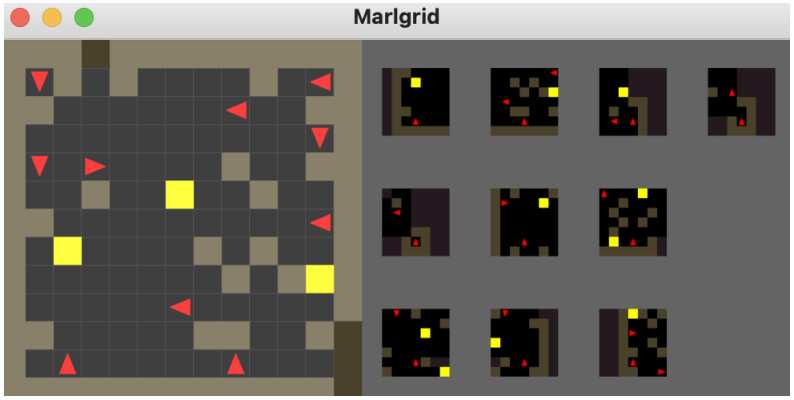

Figure 3: Example of MARL GridWorld

Other methods of comparison include either different ways of solving the optimization problem or different ways of modelling the joint Q-function. Some other methods of comparison include:

1. **Mean Field Inference**: Instead of taking the policy to be an SPN, we take the policy (variational family) to be a factored policy across all the agents. This also allows for tractable inference but is clearly less expressive compared to an SPN.

2. **Reinforce**: For solving discrete variational optimization problems, reinforce or the log-derivative trick is used commonly however this has been known to suffer from high variance in its estimates.

3. **Independent Q-Learning**: We learn the actions by taking argmax over individual Q-functions which are learnt independently of each other.

4. **Value Decomposition Network**: The joint Q-function is modelled as the sum of individual Q functions with the argmax factorisation.

The plot below 4 shows the cumulative reward plotted against the episodes during training time. For the SPN agent, the number of factors P were 20 and the size of each factor was 2. The length of each episode was kept to be a maximum of 200 and the max reward which all the agents can obtain was 1 if they solve the coverage of all the goals correctly. There were a total of $n = 6$ agents in a grid of size 20×20.

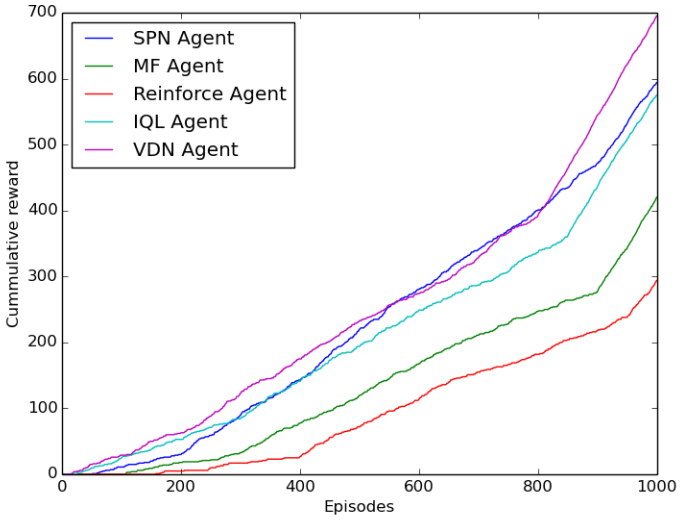

Figure 4: Cumulative Reward vs Episodes

6.1 Inferences on Results

As we can see in the plot, the SPN agent is very competitive with the VDN agent even for just solving the RL task and we also have the added benefit of actually have learnt a joint policy on which we can perform inferences. Overall this shows the effectiveness of the above approach and even says that modelling the joint Q-function as a factor graph is not that limiting. The main takeaway is the we need to put some more effort in the modelling of the Q-function by maybe adding more terms with different factor sizes etc. to make the model more expressive.

6.2 Conclusion and Contributions

In this work, we presented the following:

1. An efficient way to learn a joint MARL policy in high-dimensional discrete action spaces
2. A particular type of low-rank factorisation of the joint Q-function which helps in tractable inference for performing max-ent RL in high-dimensional discrete action spaces

7 Future Work

1. Try out a more expressive form of the joint Q-function or even try it learn the structure of the factorisation based on the task and environment provided
2. Try out more complicated MARL environments such as StarCraft II
3. Try out the inference experiments such as conditioning on the actions of some agents and seeing how the other agents behave

References

[1] Gabriel Dulac-Arnold et al. "Reinforcement Learning in Large Discrete Action Spaces". In: *CoRR* abs/1512.07679 (2015). arXiv: 1512.07679. URL: http://arxiv.org/abs/1512.07679.

[2] Benjamin Eysenbach and Sergey Levine. "Maximum Entropy RL (Provably) Solves Some Robust RL Problems". In: *CoRR* abs/2103.06257 (2021). arXiv: 2103.06257. URL: https://arxiv.org/abs/2103.06257.

[3] Carlos Guestrin, Michail Lagoudakis, and Ronald Parr. "Coordinated Reinforcement Learning". In: *In Proceedings of the ICML-2002 The Nineteenth International Conference on Machine Learning*. 2002, pp. 227–234.

[4] Tuomas Haarnoja et al. "Reinforcement Learning with Deep Energy-Based Policies". In: *CoRR* abs/1702.08165 (2017). arXiv: 1702.08165. URL: http://arxiv.org/abs/1702.08165.

[5] Sergey Levine. "Reinforcement Learning and Control as Probabilistic Inference: Tutorial and Review". In: *CoRR* abs/1805.00909 (2018). arXiv: 1805.00909. URL: http://arxiv.org/abs/1805.00909.

[6] Hoifung Poon and Pedro Domingos. "Sum-product networks: A new deep architecture". In: *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*. 2011, pp. 689–690. DOI: 10.1109/ICCVW.2011.6130310.

[7] Tabish Rashid et al. "QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning". In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, Oct. 2018, pp. 4295–4304. URL: https://proceedings.mlr.press/v80/rashid18a.html.

[8] Brian Sallans and Geoffrey E Hinton. "Using Free Energies to Represent Q-values in a Multiagent Reinforcement Learning Task". In: *Advances in Neural Information Processing Systems*. Ed. by T. Leen, T. Dietterich, and V. Tresp. Vol. 13. MIT Press, 2000. URL: https://proceedings.neurips.cc/paper/2000/file/2d1b2a5ff364606ff041650887723470-Paper.pdf.

[9] Andy Shih and Stefano Ermon. "Probabilistic Circuits for Variational Inference in Discrete Graphical Models". In: *Advances in Neural Information Processing Systems 33 (NeurIPS)*. Dec. 2020.

[10] Peter Sunehag et al. *Value-Decomposition Networks For Cooperative Multi-Agent Learning*. 2017. arXiv: 1706.05296 [cs.AI].

[11] Tom Van de Wiele et al. "Q-Learning in enormous action spaces via amortized approximate maximization". In: *CoRR* abs/2001.08116 (2020). arXiv: 2001.08116. URL: https://arxiv.org/abs/2001.08116.

[12] Yuzhe Yang et al. "Harnessing Structures for Value-Based Planning and Reinforcement Learning". In: *International Conference on Learning Representations*. 2020. URL: https://openreview.net/forum?id=rklHqRVKvH.