ENTROPY BONUS FOR POLICY GRADIENT WITH MULTI-DIMENSIONAL ACTION SPACE

Anonymous authors
Paper under double-blind review

ABSTRACT

In recent years, deep reinforcement learning has been shown to be adept at solving sequential decision processes with high-dimensional state spaces such as in the Atari games. Many reinforcement learning problems, however, involve high-dimensional discrete action spaces as well as high-dimensional state spaces. This paper considers entropy bonus, which is used to encourage exploration in policy gradient. In the case of high-dimensional action spaces, calculating the entropy and its gradient requires enumerating all the actions in the action space and running forward and backpropagation for each action, which may be computationally infeasible. We develop several novel unbiased estimators for the entropy bonus and its gradient. We apply these estimators to several models for the parameterized policies, including Independent Sampling, CommNet, Autoregressive with Modified MDP, and Autoregressive with LSTM. Finally, we test our algorithms on two environments: a multi-hunter multi-rabbit grid game and a multi-agent multi-arm bandit problem. The results show that our entropy estimators substantially improve performance with marginal additional computational cost.

1 INTRODUCTION

In recent years, deep reinforcement learning has been shown to be adept at solving sequential decision processes with high-dimensional state spaces such as in the Go game (Silver et al. (2016)) and Atari games (Mnih et al. (2013), Mnih et al. (2015), Mujika (2016), O'Donoghue et al. (2016), Parisotto et al. (2015), Wang et al. (2016), Czarnecki et al. (2017)). In all of these success stories, the size of the action space was relatively small. Many reinforcement learning problems, however, involve high-dimensional action spaces as well as high-dimensional state spaces. Examples include StarCraft (Vinyals et al. (2017), Lin et al. (2017)), where there are many agents, each of which can take a finite number of actions; and coordinating self-driving cars at an intersection, where each car can take a finite set of actions (Sukhbaatar et al. (2016)).

In policy gradient, in order to encourage sufficient exploration, an entropy bonus term is typically added to the objective function. However, in the case of high-dimensional action spaces, calculating the entropy and its gradient requires enumerating all the actions in the action space and running forward and backpropagation for each action, which may be computationally infeasible.

In this paper, we develop several novel unbiased estimators for the entropy bonus and its gradient. We apply these estimators to several models for the parameterized policies, including Independent Sampling, CommNet, Autoregressive with Modified MDP, and Autoregressive with LSTM. For all of these parameterizations, actions can be efficiently sampled from the policy distribution, and backpropagation can be employed for training. These parameterizations can be combined with the entropy bonus estimators and stochastic gradient descent, giving a new class of policy gradient algorithms with desirable exploration. Finally, we test our algorithms on two environments: a multi-hunter multi-rabbit grid game and a multi-agent multi-arm bandit problem. The results show that our entropy estimators can substantially improve performance with marginal additional computational cost.

2 POLICY GRADIENT FOR MULTI-DIMENSIONAL ACTION SPACE

Consider an MDP with a d-dimensional action space $\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2 \times \cdots \times \mathcal{A}_d$. Denote $\mathbf{a} = (a_1, \ldots, a_d)$ for an action in $\mathcal{A}$. A policy $\pi(\cdot|s)$ specifies for each state s a distribution over the action space $\mathcal{A}$. In the standard RL setting, an agent interacts with an environment over a number of discrete time steps (Sutton & Barto (1998), Silver (2015)). At time step t, the agent is in state s_t and samples an action $\mathbf{a}_t$ from the policy distribution $\pi(\cdot|s_t)$. The agent then receives a scalar reward r_t and the environment enters the next state s_{t+1}. The agent then samples $\mathbf{a}_{t+1}$ from $\pi(\cdot|s_{t+1})$ and so on. The process continues until the end of the episode, denoted by T. The return $R_t = \sum_{k=0}^{T-t} \gamma^k r_{t+k}$ is the discounted accumulated return from time step t until the end of the episode.

In policy gradient, we consider a set of parameterized policies $\pi_\theta(\cdot|s)$, $\theta \in \Theta$, and attempt to find a good θ within a parameter set Θ. Typically, the policy $\pi_\theta(\cdot|s)$ is generated by a neural network with θ denoting the network's weights and biases. The parameters θ are updated by performing stochastic gradient ascent on the expected reward. One example of such an algorithm is REINFORCE (Williams & Peng (1991)), where in a given episode at time step t, θ are updated as followed:

$$\Delta\theta = \alpha \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_t|s_t)(R_t - b_t(s_t))$$

where $b_t(s_t)$ is a baseline. It is well known that the policy gradient algorithm often converges to a local optimum. To discourage convergence to a highly suboptimal policy, the policy entropy is typically added to the update rule:

$$\Delta\theta = \alpha \sum_{t=0}^{T} [\nabla_\theta \log \pi_\theta(\mathbf{a}_t|s_t)(R_t - b_t(s_t)) + \beta \nabla_\theta H_\theta(s_t)] \tag{1}$$

where

$$H_\theta(s_t) := -\sum_{\mathbf{a} \in \mathcal{A}} \pi_\theta(\mathbf{a}|s_t) \log \pi_\theta(\mathbf{a}|s_t) \tag{2}$$

This approach is often referred to as adding entropy bonus or entropy regularization (Williams & Peng (1991)) and is widely used in different applications of neural networks, such as optimal control in Atari games (Mnih et al. (2016)), multi-agent games (Lowe et al. (2017)) and optimizer search for supervised machine learning with RL (Bello et al. (2017)). β is referred to as the entropy weight.

3 ENTROPY BONUS APPROXIMATION FOR LARGE ACTION SPACE

As shown in (1), an entropy bonus is typically included to enhance exploration. However, for large multi-dimensional action space, calculating the entropy and its gradient requires enumerating all the actions in the action space and running forward and backpropagation for each action. In this section, we develop computationally efficient unbiased estimates for the entropy and its gradient.

To abbreviate notations, we write $p_\theta(\mathbf{a})$ for $\pi_\theta(\mathbf{a}|s_t)$ and $\mathbf{a}_i$ for $(a_1, a_2, \ldots, a_i)$. We consider autoregressive models whereby the sample components a_i, $i = 1, \ldots, d$ are sequentially generated. In particular, after obtaining $a_1, a_2, \ldots, a_{i-1}$, we will generate $a_i \in \mathcal{A}_i$ from some parameterized distribution $p_\theta(\cdot|\mathbf{a}_{i-1})$ defined over the one-dimensional set $\mathcal{A}_i$. After generating the distribution $p_\theta(\cdot|\mathbf{a}_{i-1})$, $i = 1, \ldots, d$ and the action components $a_1, \ldots, a_d$ sequentially, we then define $p_\theta(\mathbf{a}) = \prod_{i=1}^{d} p_\theta(a_i|\mathbf{a}_{i-1})$.

Let $\mathbf{A} = (A_1, \ldots, A_d)$ denote a random variable with distribution $p_\theta(\cdot)$. Let H_θ denote the exact entropy of the distribution $p_\theta(\mathbf{a})$:

$$H_\theta = -\sum_{\mathbf{a}} p_\theta(\mathbf{a}) \log p_\theta(\mathbf{a}) = -\mathrm{E}_{\mathbf{A} \sim p_\theta}[\log p_\theta(\mathbf{A})] = -\sum_{i=1}^{d} \mathrm{E}_{\mathbf{A} \sim p_\theta}[\log p_\theta(A_i|\mathbf{A}_{i-1})]$$

3.1 CRUDE UNBIASED ESTIMATOR

During training within an episode, for each state s_t, the policy generates an action $\mathbf{a}$. We refer to this generated action as the episodic sample. A crude approximation of the entropy bonus is:

$$H_\theta^{\text{crude}}(\mathbf{a}) = -\log p_\theta(\mathbf{a}) = -\sum_{i=1}^{d} \log p_\theta(a_i|\mathbf{a}_{i-1})$$

This approximation is an unbiased estimate of H_θ but its variance is likely to be large. To reduce the variance, we can generate multiple action samples when in s_t and average the log action probabilities over the samples. However, generating a large number of samples is costly, especially when each sample is generated from a neural network, since each sample requires one additional forward pass.

3.2 SMOOTHED ESTIMATOR

This section proposes an alternative unbiased estimator for H_θ which only requires the one episodic sample and accounts for the entropy along each dimension of the action space:

$$\widetilde{H}_\theta(\mathbf{a}) := -\sum_{i=1}^{d} \sum_{a \in \mathcal{A}_i} p_\theta(a|\mathbf{a}_{i-1}) \log p_\theta(a|\mathbf{a}_{i-1}) = \sum_{i=1}^{d} H_\theta^{(i)}(\mathbf{a}_{i-1})$$

where

$$H_\theta^{(i)}(\mathbf{a}_{i-1}) := -\sum_{a \in \mathcal{A}_i} p_\theta(a|\mathbf{a}_{i-1}) \log p_\theta(a|\mathbf{a}_{i-1})$$

which is the entropy of A_i conditioned on $\mathbf{a}_{i-1}$. This estimate of the entropy bonus is computationally efficient since for each dimension i, we need to obtain $p_\theta(\cdot|\mathbf{a}_{i-1})$, its log and gradient anyway during training. We refer to this approximation as the smoothed entropy.

The smoothed estimate of the entropy $\widetilde{H}_\theta(\mathbf{A})$ has several appealing properties. The proofs of Theorem 1 and Theorem 3 are straightforward and omitted.

Theorem 1. *$\widetilde{H}_\theta(\mathbf{A})$ is an unbiased estimator of the exact entropy H_θ.*

Theorem 2. *If $p_\theta(\mathbf{a})$ has a multivariable normal distribution with mean and variance depending on θ, then:*

$$\widetilde{H}_\theta(\mathbf{a}) = H_\theta \quad \forall \mathbf{a} \in \mathcal{A}$$

Thus, the smoothed estimate of the entropy equals the exact entropy for a multivariate normal parameterization of the policy (Proof in Appendix B).

Theorem 3. *(i) If there exists a sequence of weights $\theta_1, \theta_2, \ldots$ in Θ such that $p_{\theta_n}(\cdot)$ converges to the uniform distribution over $\mathcal{A}$, then*

$$\sup_{\theta \in \Theta} \widetilde{H}_\theta(\mathbf{a}) = \sup_{\theta \in \Theta} H_\theta \quad \forall \mathbf{a} \in \mathcal{A}$$

(ii) If there exists a sequence of weights $\theta_1, \theta_2, \ldots$ in Θ such that $p_{\theta_n}(\mathbf{a}^*) \to 1$ for some $\mathbf{a}^*$, then

$$\inf_{\theta \in \Theta} \widetilde{H}_\theta(\mathbf{a}) = \inf_{\theta \in \Theta} H_\theta = 0 \quad \forall \mathbf{a} \in \mathcal{A}$$

Thus, the smoothed estimate of the entropy $\widetilde{H}_\theta(\mathbf{a})$ mimics the exact entropy in that it has the same supremum and infinum values as the exact entropy.

The above theorems indicate that $\widetilde{H}_\theta(\mathbf{a})$ may serve as a good proxy for H_θ: it is an unbiased estimator for H_θ, it has the same minimum and maximum values when varying θ; and in the special case when $p_\theta(\mathbf{a})$ has a multivariate normal distribution, it is actually equal to H_θ for all $\mathbf{a} \in \mathcal{A}$. Our numerical experiments have shown that the smoothed estimator $\widetilde{H}_\theta(\mathbf{a})$ typically has lower variance than the crude estimator $H_\theta^{\text{crude}}(\mathbf{a})$. However, it is not generally true that the smoothed estimator always has lower variance as counterexamples can be found.

3.3 SMOOTHED MODE ESTIMATOR

For the smoothed estimate of the entropy $\widetilde{H}_\theta(\mathbf{a})$, we use the episodic action $\mathbf{a}$ to form the estimate. We now consider alternative choices of actions which may improve performance at modest additional computational cost. First consider $E^* = -\sum_{i=1}^d \sum_{a\in\mathcal{A}_i} p_\theta(a|a_1^*,\ldots,a_{i-1}^*)\log p_\theta(a|a_1^*,\ldots,a_{i-1}^*)$ where $\mathbf{a}^* = (a_1^*,\ldots,a_d^*) = \operatorname{argmax}_{\mathbf{a}\in\mathcal{A}} p_\theta(\mathbf{a})$. Thus in this case, instead of calculating the smoothed estimate of the entropy with the episodic action $\mathbf{a}$, we calculate it with the most likely action $\mathbf{a}^*$. The problem here is that it is not easy to find $\mathbf{a}^*$ when the given conditional probabilities $p_\theta(a_i|\mathbf{a}_{i-1})$ are not in closed form but only available algorithmically as outputs of neural networks. A more computationally efficient approach would be to choose the action greedily: $\hat{a}_1 = \operatorname{argmax}_{a\in\mathcal{A}_1} p_\theta(a)$ and $\hat{a}_i = \operatorname{argmax}_{a\in\mathcal{A}_i} p_\theta(a|\hat{a}_1,\ldots,\hat{a}_{i-1})$ for $i = 2,\ldots,d$. This leads to the definition $\widehat{H}_\theta = -\sum_{i=1}^d \sum_{a\in\mathcal{A}_i} p_\theta(a|\hat{a}_1,\ldots,\hat{a}_{i-1})\log p_\theta(a|\hat{a}_1,\ldots,\hat{a}_{i-1})$. The action $\hat{\mathbf{a}}$ is an approximation for the mode of the distribution $p_\theta(\cdot)$. As often done in NLP, we can use beam search to determine an action $\mathbf{a}'$ that has higher probability, that is, $p_\theta(\mathbf{a}') \geq p_\theta(\hat{\mathbf{a}})$. Indeed, the above $\widehat{H}_\theta$ definition is beam search with beam size equal to 1. We refer to $\widehat{H}_\theta$ as the smoothed mode estimate.

$\widehat{H}_\theta$ with an appropriate beam size may be a better approximation for the entropy H_θ than $\widetilde{H}_\theta(\mathbf{a})$. However, calculating $\widehat{H}_\theta$ and its gradient comes with some computational cost. For example, with a beam size equal to one, we would have to make two passes through the policy neural network at each time step: one to obtain the episodic sample $\mathbf{a}$ and the other to obtain the greedy action $\hat{\mathbf{a}}$. For beam size n, we would need to make $n+1$ passes. We note that $\widehat{H}_\theta$ is a biased estimator for H_θ but with no variance. Thus there is a bias-variance tradeoff between $\widetilde{H}_\theta(\mathbf{a})$ and $\widehat{H}_\theta$. Note that $\widehat{H}_\theta$ also satisfies Theorems 2 and 3 in subsection 3.2.

3.4 ESTIMATING THE GRADIENT OF THE ENTROPY

So far we have been looking at estimates of entropy. But the update rule (1) uses the gradient of the entropy rather than the entropy. As it turns out, the gradients of the estimators $H_\theta^{\text{crude}}(\mathbf{a})$ and $\widetilde{H}_\theta(\mathbf{a})$ are not unbiased estimates of the gradient of the entropy. In this subsection, we provide unbiased estimators for the gradient of the entropy. For simplicity, in this section, we assume a one-step decision setting, such as in a multi-armed bandit problem. A straightforward calculation shows:

$$\nabla_\theta H_\theta = \mathrm{E}_{\mathbf{A}\sim p_\theta}[-\log p_\theta(\mathbf{A})\nabla_\theta \log p_\theta(\mathbf{A})] \tag{3}$$

Suppose $\mathbf{a}$ is one sample from $p_\theta(\cdot)$. A crude unbiased estimator for the gradient of the entropy therefore is: $-\log p_\theta(\mathbf{a})\nabla_\theta \log p_\theta(\mathbf{a}) = \log p_\theta(\mathbf{a})\nabla_\theta H_\theta^{\text{crude}}(\mathbf{a})$. Note that this estimator is equal to the gradient of the crude estimator multiplied by a correction factor.

Analogous to the smoothed estimator for entropy, we can also derive a smoothed estimator for the gradient of the entropy.

Theorem 4. *If* $\mathbf{a}$ *is a sample from* $p_\theta(\cdot)$*, then*

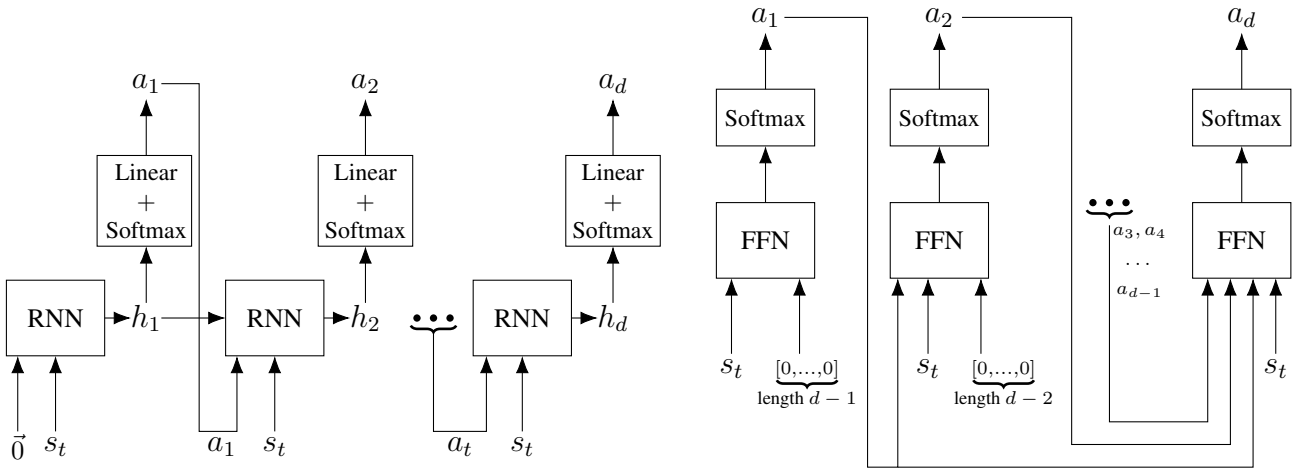
$$\nabla_\theta \widetilde{H}_\theta(\mathbf{a}) + \sum_{i=1}^d H_\theta^{(i)}(\mathbf{a}_{i-1})\nabla_\theta \sum_{j=1}^{i-1} \log p_\theta(a_j|\mathbf{a}_{j-1})$$

is an unbiased estimator for the gradient of the entropy (Proof in Appendix C).

Note that this estimate for the gradient of the entropy is equal to the gradient of the smoothed estimate $\widetilde{H}_\theta(\mathbf{a})$ plus a correction term. We refer to this estimate of the entropy gradient as the unbiased gradient estimate.

4 POLICY PARAMETERIZATION FOR EFFICIENT SAMPLING

We will apply the estimators in the previous section to several models for the paramaterized marginal policies $p_\theta(a|\mathbf{a}_{i-1})$, $a \in \mathcal{A}_i$. In this discussion, we assume that the size of the one-dimensional action sets are equal, that is, $|\mathcal{A}_1| = |\mathcal{A}_2| = ... = |\mathcal{A}_d| = K$. To handle action sets of different sizes, we include inconsequential actions.



(a) The RNN architecture. To generate a_i, we input s_t and a_{i-1} into the RNN and then pass the resulting hidden state h_i through a linear layer and a softmax to generate a distribution, from which we sample a_i.

(b) The MMDP architecture. To generate a_i, we input s_t and $a_1, a_2, \ldots, a_{i-1}$ into a FFN. The output is passed through a softmax layer, providing a distribution from which we sample a_i. Since the input size of the FFN is fixed, when generating a_i, constants 0 serve as placeholders for $a_{i+1}, \ldots, a_{d-1}$ in the input to the FFN.

Figure 1: The RNN and MMDP architectures for generating parameterized policies.

4.1 Independent Sampling (IS)

This baseline policy is a FFN that takes as input the state, creates a representation of the state, and from the representation outputs d softmax heads, from which the value of each action dimension can be sampled independently (Sukhbaatar et al. (2016)). In this case, the smoothed estimate is equal to the exact entropy.

4.2 Using MMDP to Generate Parameterized Policy

The underlying MDP can be modified to create an equivalent MDP for which the action space is one-dimensional. We refer to this MDP as the Modified MDP (MMDP). In the original MDP, we have state space $\mathcal{S}$ and action space $\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2 \times \cdots \times \mathcal{A}_d$. In MMDP, the state encapsulates the original state and all the action dimensions selected for state s so far (Figure 1(b)). We note that Metz et al. (2017) recently and independently proposed the reformulation of the MDP into MMDP. More details can be found in Appendix D.

4.3 Using an LSTM to Generate the Parameterized Policy

LSTMs have recently been used with great success for autoregressive models in language translation tasks (Sutskever et al. (2014)). Here we use an LSTM to generate a parameterized multi-dimensional distribution $p_\theta(\cdot)$ and to sample $\mathbf{a}$ from that distribution (Figure 1(a)). To generate a_i, we run a forward pass through the LSTM with the input being a_{i-1} and the current state s_t (and implicitly on $a_1, \ldots, a_{i-1}$ which influences h_{i-1}). This produces a hidden state h_i, which is then passed through a linear layer, producing a K dimensional vector. The softmax of this vector is taken to produce the one-dimensional conditional distribution $p_\theta(a|\mathbf{a}_{i-1})$, $a \in \mathcal{A}_i$. a_i is sampled from this one-dimensional distribution, and is then fed into the next stage of the LSTM to produce a_{i+1}. We note that this approach is an adaptation of sequence modeling in supervised machine learning (van den Oord et al. (2016)) to reinforcement learning and has also been proposed by Metz et al. (2017) and Bahdanau et al. (2016).

5 EXPERIMENTAL RESULTS

We designed experiments to compare the different entropy estimators for three models discussed in the previous section as well as for the CommNet model, a related approach introduced by Sukhbaatar et al. (2016). (Appendix E discusses the state representation for CommNet.) For each entropy approximation, the entropy weight β was tuned to give the highest reward. For IS and MMDP, the number of hidden layers was tuned from 1 to 7. For CommNet, the number of communication steps was tuned from 2 to 5, the learning rate was tuned between 3e-3 and 3e-4 and the size of the policy hidden layer was tuned between 128 and 256. Appendix A lists the rest of the hyperparameters.

5.1 HUNTERS AND RABBITS

In this environment, there is a $n \times n$ grid. At the beginning of each episode, d hunters and d rabbits are randomly placed in the grid. The rabbits remain fixed in the episode, and each hunter can move to a neighboring square (including diagonal neighbors) or stay at the current square. So each hunter has nine possible actions, and altogether there are $|\mathcal{A}| = 9^d$ actions at each time step. When a hunter enters a square with a rabbit, the hunter captures the rabbit and remains there until the end of the episode. In each episode, the goal is for the hunters to capture the rabbits as quickly as possible. Each episode is allowed to run for at most ten thousands time steps.

To provide a dense reward signal, we modify the goal as following: capturing a rabbit gives a reward of 1, which is discounted by the number of time steps taken since the beginning of the episode. The discount factor is 0.8. The goal is to maximize the episode's total discounted reward. After a hunter captures a rabbit, they both become inactive.

Comparison of different entropy estimates for IS, LSTM, MMDP and CommNet

Table 1 shows the performance of the IS, LSTM, MMDP and CommNet models with the different entropy estimates. Training and evaluation were performed in a square grid of 5 by 5 with 5 hunters and 5 rabbits. Results are averaged over 5 seeds. For each seed, training and evaluation were run for 1 million and 1 thousand episodes respectively.

As compared with no entropy, crude entropy can actually reduce performance. However, smoothed entropy and smoothed mode entropy always increase performance, often significantly. For the LSTM model, the best performing approximation is smoothed entropy, reducing the mean episode length by 45% and increasing the mean episode reward by 10% compared to without entropy. We also note that there is not a significant difference in performance between the smoothed entropy, smoothed mode entropy, and the unbiased gradient approaches. When comparing the four models, we see that the LSTM model with smoothed entropy does significantly better the other three models. The CommNet model could potentially be improved by allowing the hunters to see more of the state; this could be investigated in future research.

The smoothed estimator is also more robust with respect to the initial seed than without entropy as shown in Figure 2. For example, for the LSTM model, in the case of without entropy, seed 0 leads to significantly worse results than the seeds 1-4. This does not happen with the smoothed estimator.

Figure 2: IS, LSTM and MMDP results across 5 seeds (y-axis denotes mean episode length).

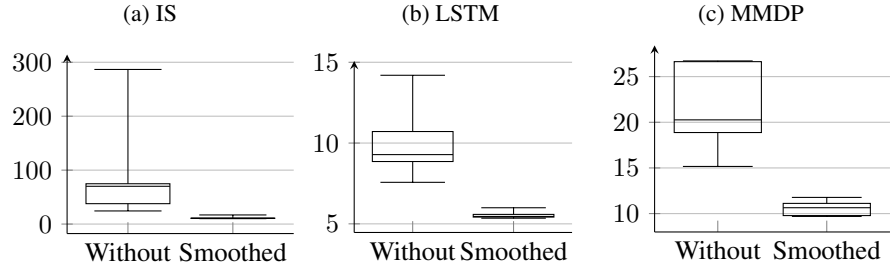


Table 1: Performance of IS, LSTM, MMDP and CommNet across different entropy approximations.

	Without Entropy	Crude Entropy	Smoothed Entropy	Smoothed Mode Entropy	Unbiased Gradient Estimate
IS Mean Episode Length	98.7 ± 78.9	32 ± 12.3	11.8 ± 1.9	11.8 ± 1.9	11.8 ± 1.9
LSTM Mean Episode Length	10.1 ± 1.9	19 ± 8.7	**5.6 ± 0.2**	6.0 ± 0.2	6.0 ± 0.1
MMDP Mean Episode Length	21.5 ± 3.7	37.3 ± 29.6	10.6 ± 0.7	10.6 ± 0.7	9.8 ± 0.6
CommNet Mean Episode Length	22.7 ± 0.6	22.3 ± 0.4	21.9 ± 0.4	21.9 ± 0.4	21.9 ± 0.4
IS Mean Episode Reward	2.2 ± 0.03	2.4 ± 0.05	2.7 ± 0.01	2.7 ± 0.01	2.7 ± 0.01
LSTM Mean Episode Reward	3.0 ± 0.06	3.0 ± 0.03	**3.3 ± 0.04**	3.2 ± 0.04	3.2 ± 0.02
MMDP Mean Episode Reward	2.8 ± 0.03	2.7 ± 0.03	2.9 ± 0.03	2.8 ± 0.04	2.9 ± 0.02
CommNet Mean Episode Reward	2.5 ± 0.01	2.6 ± 0.01	2.6 ± 0.01	2.6 ± 0.01	2.6 ± 0.01

Entropy approximations versus exact entropy

We now consider how policies trained with entropy approximations compare with polices trained with exact entropy. In order to calculate exact entropy in an acceptable amount of time, we reduced the number of hunters and rabbits to 4 hunters and 4 rabbits. Training was run for 50,000 episodes. Table 2 shows the performance differences between policies trained with entropy approximations and exact entropy. We see that the best entropy approximations perform only slightly worse than exact entropy for both LSTM and MMDP. Once again we see that the LSTM model performs better than the MMDP model.

Table 2: LSTM and MMDP results for entropy approximation versus exact entropy.

	LSTM Smoothed Entropy	LSTM Exact Entropy	MMDP Unbiased Gradient Estimate	MMDP Exact Entropy
Mean Episode Length	9.0 ± 0.3	**8.9 ± 0.2**	11.5 ± 0.3	10.7 ± 0.4
Mean Episode Reward	2.14 ± 0.02	**2.19 ± 0.02**	2.01 ± 0.01	2.1 ± 0.01

5.2 MULTI-AGENT MULTI-ARM BANDITS

We examine a multi-agent version of the standard multi-armed bandit problem, where there are d agents each pulling one of K arms, with $d \leq K$. The kth arm generates a reward r_k. The total reward in a round is generated as follows. In each round, each agent chooses an arm. All of the chosen arms are then pulled, with each pulled arm generating a reward. Note that the total number of arms chosen, c, may be less than d since some agents may choose the same arm. The total reward is the sum of rewards from the c chosen arms. The optimal policy is for the d agents to collectively pull the d arms with the highest rewards. Additionally, among all the optimal assignments of d agents to the d arms that yield the highest reward, we add a bonus reward with probability p^* if one particular agents-to-arms configuration is chosen.

We performed experiments with 4 agents and 10 arms, with the kth arm providing a reward of k. The exceptional assignment gets a bonus of 166 (making a total reward of 200) with probability 0.01, and no bonus with probability 0.99. Thus the maximum expected reward is 35.66. Training

was run for 100,000 rounds for each of 10 seeds. Table 3 shows average results for the last 500 of the 100,000 rounds.

Table 3: Performance of IS and LSTM policy parameterizations.

	Without Entropy	Crude Entropy	Smoothed Entropy	Unbiased Gradient Estimate
IS Mean Reward	34.2 ± 1.3	34.4 ± 1.3	34.2 ± 1.3	34.2 ± 1.3
LSTM Mean Reward	34.9 ± 0.8	35.5 ± 1.1	35.9 ± 0.8	$\mathbf{35.9 \pm 0.3}$
IS Percentage Optimal Config Pulled	19.8 ± 39.7	29.7 ± 49.6	19.7 ± 39.7	19.7 ± 39.7
LSTM Percentage Optimal Config Pulled	39.8 ± 35.9	59.4 ± 35.7	95.0 ± 1.9	$\mathbf{95.7 \pm 2.7}$

The results for the multi-agent bandit problem are consistent with those for the hunter-rabbit problem. Policies obtained with the entropy approximations all perform better than policies obtained without entropy or with crude entropy, particularly for the percentage of rounds the arms are pulled with the optimal configuration. Note that LSTM with the unbiased gradient estimator gives the best results.

6 Related Work

Metz et al. (2017) recently and independently proposed the reformulation of MDP into MMDP and the LSTM policy parameterization. They inject noise into the action space to encourage exploration. Usunier et al. (2016) uses MMDP and noise injection in the parameter space to achieve high performance in multi-agent Starcraft micro-management tasks. Instead of noise injection, we propose novel estimators for the entropy bonus that is often used to encourage exploration in policy gradient.

While entropy regularization has been mostly used in policy gradient algorithms, Schulman et al. (2017) applies entropy regularization to Q-learning methods. They make an important observation about the equivalence between policy gradient and entropy regularized Q-learning, which they term soft Q-learning.

To the best of our knowledge, no prior work has dealt with approximating the policy entropy for MDP with large multi-dimensional discrete action space. On the other hand, there have been many attempts to devise methods to encourage beneficial exploration for policy gradient. Nachum et al. (2016) modifies the entropy term by adding weights to the log action probabilities, leading to a new optimization objective termed under-appreciated reward exploration.

Dulac-Arnold et al. (2015) embeds discrete actions in a continuous space, picks actions in the continuous space and map these actions back into the discrete space. However, their algorithm introduces a new hyper-parameter that requires tuning for every new task. Our approach involves no new hyper-parameter other than those normally used in deep learning.

The LSTM policy parameterization can be seen as the adaptation of sequence modeling techniques in supervised machine learning, such as in speech generation (van den Oord et al. (2016)) or machine translation (Sutskever et al. (2014)) to reinforcement learning, as was previously done in Bahdanau et al. (2016).

7 Conclusion

In this paper, we developed several novel unbiased estimators for entropy bonus and its gradient. We did experimental work for two environments with large multi-dimensional action spaces. We found that the smoothed estimate of the entropy and the unbiased estimate of the entropy gradient can significantly increase performance with marginal additional computational cost.

REFERENCES

Dzmitry Bahdanau, Philemon Brakel, Kelvin Xu, Anirudh Goyal, Ryan Lowe, Joelle Pineau, Aaron C. Courville, and Yoshua Bengio. An actor-critic algorithm for sequence prediction. *CoRR*, abs/1607.07086, 2016. URL http://arxiv.org/abs/1607.07086.

Irwan Bello, Barret Zoph, Vijay Vasudevan, and Quoc Le. Neural optimizer search with reinforcement learning. 2017.

Wojciech Marian Czarnecki, Simon Osindero, Max Jaderberg, Grzegorz Swirszcz, and Razvan Pascanu. Sobolev training for neural networks. *CoRR*, abs/1706.04859, 2017. URL http://arxiv.org/abs/1706.04859.

Gabriel Dulac-Arnold, Richard Evans, Hado van Hasselt, Peter Sunehag, Timothy Lillicrap, Jonathan Hunt, Timothy Mann, Theophane Weber, Thomas Degris, and Ben Coppin. Deep reinforcement learning in large discrete action spaces, 2015.

R. A. Johnson and D. W. Wichern (eds.). *Applied Multivariate Statistical Analysis*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988. ISBN 0-130-41146-9.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL http://arxiv.org/abs/1412.6980.

Zeming Lin, Jonas Gehring, Vasil Khalidov, and Gabriel Synnaeve. STARDATA: A starcraft AI research dataset. *CoRR*, abs/1708.02139, 2017. URL http://arxiv.org/abs/1708.02139.

Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments, 2017.

Luke Metz, Julian Ibarz, Navdeep Jaitly, and James Davidson. Discrete sequential prediction of continuous actions for deep RL. *CoRR*, abs/1705.05035, 2017. URL http://arxiv.org/abs/1705.05035.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013. URL http://arxiv.org/abs/1312.5602.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, and et al. Human-level control through deep reinforcement learning, Feb 2015. URL http://www.nature.com/nature/journal/v518/n7540/abs/nature14236.html.

Volodymyr Mnih, Adri Puigdomnech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. 2016.

Asier Mujika. Multi-task learning with deep model based reinforcement learning. *CoRR*, abs/1611.01457, 2016. URL http://arxiv.org/abs/1611.01457.

Ofir Nachum, Mohammad Norouzi, and Dale Schuurmans. Improving policy gradient by exploring under-appreciated rewards, 2016.

Brendan O'Donoghue, Remi Munos, Koray Kavukcuoglu, and Volodymyr Mnih. Combining policy gradient and q-learning, 2016.

Emilio Parisotto, Lei Jimmy Ba, and Ruslan Salakhutdinov. Actor-mimic: Deep multitask and transfer reinforcement learning. *CoRR*, abs/1511.06342, 2015. URL http://arxiv.org/abs/1511.06342.

John Schulman, Nicolas Heess, Theophane Weber, and Pieter Abbeel. Gradient estimation using stochastic computation graphs. In *Advances in Neural Information Processing Systems*, pp. 3528–3536, 2015.

John Schulman, Pieter Abbeel, and Xi Chen. Equivalence between policy gradients and soft q-learning, 2017.

David Silver. Ucl course on rl, 2015. URL http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, jan 2016. ISSN 0028-0836. doi: 10.1038/nature16961.

Sainbayar Sukhbaatar, Arthur Szlam, and Rob Fergus. Learning multiagent communication with backpropagation. *CoRR*, abs/1605.07736, 2016. URL http://arxiv.org/abs/1605.07736.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks, 2014.

Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998. ISBN 0262193981.

Tieleman and Hinton. Rmsprop: Divide the gradient by a running average of its recent magnitude - university of toronto, 2012. URL https://www.coursera.org/learn/neural-networks/lecture/YQHki/rmsprop-divide-the-gradient-by-a-running-average-of-its-recent-magnitude.

Nicolas Usunier, Gabriel Synnaeve, Zeming Lin, and Soumith Chintala. Episodic exploration for deep deterministic policies: An application to starcraft micromanagement tasks. *CoRR*, abs/1609.02993, 2016. URL https://openreview.net/forum?id=r1LXit5ee&noteId=r1LXit5ee.

Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W. Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *CoRR*, abs/1609.03499, 2016. URL http://arxiv.org/abs/1609.03499.

Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, John Quan, Stephen Gaffney, Stig Petersen, Karen Simonyan, Tom Schaul, Hado van Hasselt, David Silver, Timothy P. Lillicrap, Kevin Calderone, Paul Keet, Anthony Brunasso, David Lawrence, Anders Ekermo, Jacob Repp, and Rodney Tsing. Starcraft II: A new challenge for reinforcement learning. *CoRR*, abs/1708.04782, 2017. URL http://arxiv.org/abs/1708.04782.

Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Rémi Munos, Koray Kavukcuoglu, and Nando de Freitas. Sample efficient actor-critic with experience replay. *CoRR*, abs/1611.01224, 2016. URL http://arxiv.org/abs/1611.01224.

Ronald J. Williams and Jing Peng. Function optimization using connectionist reinforcement learning algorithms. 1991. doi: 10.1080/09540099108946587.

APPENDIX

A HYPERPARAMETERS

Hyperparameters for hunter-rabbit game

For IS, the numbers of hidden layers for smoothed entropy, unbiased gradient estimate, crude entropy and without entropy are 1, 1, 5 and 7 respectively. The entropy weights for smoothed entropy, unbiased gradient estimate and crude entropy are 0.03, 0.02 and 0.01 respectively. The hyper-parameters for smoothed mode entropy is not listed since the smoothed mode entropy equals the smoothed entropy for IS.

For CommNet, the numbers of communication step for without entropy, crude entropy, smoothed entropy and unbiased entropy gradient are 2, 2, 2 and 2 respectively. The sizes of the policy hidden layer for without entropy, crude entropy, smoothed entropy and unbiased entropy gradient are 256, 256, 256 and 128 respectively. The entropy weights for crude entropy, smoothed entropy and unbiased entropy gradient are 0.04, 0.04 and 0.01 respectively. The policies were optimized using Adam (Kingma & Ba (2014)) with learning rate 3e-4. We found 3e-4 gives better performance than the learning rate 3e-3 originally used in Sukhbaatar et al. (2016).

The LSTM policy has 128 hidden nodes. For the MMDP policy, the number of hidden layers for smoothed entropy, smoothed mode entropy, unbiased gradient estimate, crude entropy and without entropy are 5, 3, 3, 4 3 and 3 respectively. Each MMDP layer has 128 nodes. We parameterize the baseline in (1) with a FFN with one hidden layer of size 64. This network was trained using first visit Monte Carlo return to minimize the L1 loss between actual and predicted values of states visited during the episode.

Both the policies and baseline are optimized after each episode with RMSprop (Tieleman & Hinton (2012)). The RHS of (1) is clipped to $[-1, 1]$ before updating the policy parameters. The learning rates for the baseline, IS, LSTM and MMDP are 10^{-3}, 10^{-3}, 10^{-3}, 10^{-4} respectively.

To obtain the results in Table 1, the entropy weights for LSTM smoothed entropy, LSTM smoothed mode entropy, LSTM unbiased gradient estimate, LSTM crude entropy, MMDP smoothed entropy, MMDP smoothed mode entropy, MMDP unbiased gradient estimate and MMDP crude entropy are 0.02, 0.021, 0.031, 0.04, 0.02, 0.03, 0.03 and 0.01 respectively.

To obtain the results in Table 2, the entropy weights for LSTM smoothed entropy, LSTM exact entropy, MMDP unbiased gradient estimate and MMDP exact entropy are 0.03, 0.01, 0.03 and 0.01 respectively. The MMDP networks have three layers with 128 nodes in each layer. Experimental results are averaged over five seeds (0-4).

Hyperparamters for Multi-Agent Multi-Arm Bandits

The experiments were run with 4 agents and 10 arms. For the 10 arms, their rewards are i for $i = 1, \ldots, 10$. The LSTM policy has 32 hidden nodes. The baseline

in (1) is a truncated average of the reward of the last 100 rounds. The entropy weight for crude entropy, smoothed entropy and unbiased gradient estimate are 0.005, 0.001 and 0.003 respectively. The learning rates for without entropy, crude entropy, smoothed entropy and unbiased gradient estimate are 0.006, 0.008, 0.002 and 0.005 respectively. Experimental results are averaged over ten seeds.

B PROOF OF THEOREM 2

We first note that for $\begin{bmatrix}\mathbf{X}_1\\ \mathbf{X}_2\end{bmatrix} \sim N\left(\begin{bmatrix}\boldsymbol{\mu}_1\\ \boldsymbol{\mu}_2\end{bmatrix}, \begin{bmatrix}\boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12}\\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22}\end{bmatrix}\right)$ where $\mathbf{X}_1$ and $\mathbf{X}_2$ are random vectors, we have $\mathbf{X}_2 \mid \mathbf{X}_1 = \mathbf{x}_1 \sim N(\bar{\boldsymbol{\mu}}, \bar{\boldsymbol{\Sigma}})$ where

$$\bar{\boldsymbol{\mu}} = \boldsymbol{\mu}_2 + \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}(\mathbf{x}_1 - \boldsymbol{\mu}_1)$$
$$\bar{\boldsymbol{\Sigma}} = \boldsymbol{\Sigma}_{22} - \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}\boldsymbol{\Sigma}_{12}$$

Observe that the covariance matrix of the conditional distribution does not depend on the value of x_1 (Johnson & Wichern (1988)).

Also note that for $\mathbf{X} \sim N(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, the entropy of $\mathbf{X}$ takes the form

$$H(\mathbf{X}) = \frac{k}{2}(\log 2\pi + 1) + \frac{1}{2}|\boldsymbol{\Sigma}|$$

where k is the dimension of $\mathbf{X}$ and $|\cdot|$ denotes the determinant. Therefore, the entropy of a multivariate normal random variable depends only on the variance and not on the mean.

Because $\mathbf{A}$ is multivariate normal, the distribution of A_i given $A_1 = a_1, \ldots, A_{i-1} = a_{i-1}$ has a normal distribution with a variance σ_i^2 that does not depend on $a_1, \ldots, a_{i-1}$. Therefore

$$H_\theta(A_i|a_1, \ldots, a_{i-1}) = \frac{1}{2}(\log 2\pi + 1 + \sigma_i^2)$$

does not depend on $a_1, \ldots, a_{i-1}$ and hence $\widetilde{H}_\theta(\mathbf{a})$ does not depend on $\mathbf{a}$. Combining this with the fact that $\widetilde{H}_\theta(\mathbf{a})$ is an unbiased estimator for H_θ gives $\widetilde{H}_\theta(\mathbf{a}) = H_\theta$ for all $\mathbf{a} \in \mathcal{A}$.

C PROOF OF THEOREM 4

Proof. From (3), we have:

$$\nabla_\theta H_\theta = -\sum_{i=1}^{d}\sum_{j=1}^{d} \mathrm{E}_{\mathbf{A}\sim p_\theta}[\log p_\theta(A_i|\mathbf{A}_{i-1})\nabla_\theta \log p_\theta(A_j|\mathbf{A}_{j-1})] \tag{4}$$

We will now use conditional expectation to calculate the terms in the double sum.

For $i < j$:

$$\begin{aligned}&\mathrm{E}_{\mathbf{A}\sim p_\theta}[\log p_\theta(A_i|\mathbf{A}_{i-1})\nabla_\theta \log p_\theta(A_j|\mathbf{A}_{j-1})|\mathbf{A}_{j-1}]\\ &= \log p_\theta(A_i|\mathbf{A}_{i-1})\mathrm{E}_{\mathbf{A}\sim p_\theta}[\nabla_\theta \log p_\theta(A_j|\mathbf{A}_{j-1})|\mathbf{A}_{j-1}] = 0\end{aligned}$$

For $i > j$:

$$\begin{aligned}
&\mathrm{E}_{\mathbf{A}\sim p_\theta}[\log p_\theta(A_i|\mathbf{A}_{i-1})\nabla_\theta \log p_\theta(A_j|\mathbf{A}_{j-1})|\mathbf{A}_{i-1}] \\
=&\nabla_\theta \log p_\theta(A_j|\mathbf{A}_{j-1})\mathrm{E}_{\mathbf{A}\sim p_\theta}[\log p_\theta(A_i|\mathbf{A}_{i-1})|\mathbf{A}_{i-1}] \\
=&-\nabla_\theta \log p_\theta(A_j|\mathbf{A}_{j-1})H_\theta^{(i)}(\mathbf{A}_{i-1})
\end{aligned}$$

For $i = j$:

$$\mathrm{E}_{\mathbf{A}\sim p_\theta}[\log p_\theta(A_i|\mathbf{A}_{i-1})\nabla_\theta \log p_\theta(A_i|\mathbf{A}_{i-1})|\mathbf{A}_{i-1}] = -\nabla_\theta H_\theta^{(i)}(\mathbf{A}_{i-1})$$

Combining these three conditional expectations with (4), we obtain:

$$\nabla_\theta H_\theta = \mathrm{E}_{\mathbf{A}\sim p_\theta}[\nabla_\theta \widetilde{H}_\theta(\mathbf{A}) + \sum_{i=1}^{d} H_\theta^{(i)}(\mathbf{A}_{i-1})\nabla_\theta \sum_{j=1}^{i-1} \log p_\theta(A_j|\mathbf{A}_{j-1})]$$

Alternatively, we could also prove Theorem 4 by fitting our model into the model of Schulman et al. (2015) and applying Theorem 1 of that paper.

D MMDP Details

In MMDP, the new action space is $\widetilde{\mathcal{A}} = \{0, 1, \ldots, K\}$ and the new state space is $\mathcal{S} \times \{0, 1, \ldots, K\}^{d-1}$ where $K = |\mathcal{A}_i|$. The newly created states has no reward discounting. Since MMDP has a one-dimensional action space, we can use a FFN to generate each action component (Figure 1(b)). The state transition probabilities for the MMDP are given by:

$$\begin{aligned}
&\widetilde{P}((s, a_1, 0, \ldots, 0)|(s, 0, \ldots, 0), a_1) = 1 \\
&\widetilde{P}((s, a_1, a_2, 0, \ldots, 0)|(s, a_1, 0, \ldots, 0), a_2) = 1 \\
&\qquad\vdots \\
&\widetilde{P}((s, a_1, \ldots, a_{d-1})|(s, a_1, \ldots, a_{d-2}, 0), a_{d-1}) = 1 \\
&\widetilde{P}((s', 0, \ldots, 0)|(s, a_1, \ldots, a_{d-1}), a_d) = P(s'|s, a_1, \ldots, a_d)
\end{aligned}$$

where $P(s'|s, a_1, \ldots, a_d)$ is the transition probabiliy of the original MDP. The reward is only generated after all d component actions are taken. It is easily seen that the MMDP is equivalent to the original MDP.

Note that the FFN input layer size is always $|\mathcal{S}| + K - 1$ and the output layer size is K. We also tried representing the actions fed into the FFN as one-hot vector, but did not find significant performance improvement.

E State Representation For CommNet

Sukhbaatar et al. (2016) proposes CommNet to handle multi-agent environments where each agent observes only part of the state and the number of agents changes throughout an episode. We thus modify the state representation of the hunters and rabbits environment to better reflect the strengths of CommNet. Each hunter only sees its own id, its position and the positions of all rabbits. More precisely, the state each hunter receives is [*hunter id*, *hunter position*, *all rabbit positions*].