Policy Gradient Methods

Tabular methods and Function Approximators are value-based which means the policy looks at the value of the state-actions Q and choose the action with optimal estimated return. In policy gradient methods, we use the function approxmiators to not predict the Q values but to estimate teh probabilities of taking each action:

$$\pi(a|s, \theta) \in [0, 1] \tag{1}$$

and thus the function approximator **is** the policy that gives out the distribution of probabilities over the actions rather than the values for you to read:

$$\pi(s, \theta) = [p(a_1), p(a_2), \cdots, p(a_n)] \tag{2}$$

and your action to take, a, is sampled from this distribution:

$$a \sim \pi(a|s, \theta) \tag{3}$$

and thus your policy and actions change more smoothly during learning,whereas in value-based methods, you manually choose the action using greedy or ϵ-greedy methods:

$$a = \arg\max_a \hat{q}(s, a). \tag{4}$$

Representing Policies using Neural Networks

Neural networks can learn the weights and propagate the extracted features of the state vector to approximate the values of the state:

$$\hat{y} = \phi_2(\phi_1(x \cdot w_1) \cdot w_2). \tag{5}$$

To train the neural network to output the vector of probability, we compress the output values to values between 0 and 1 such that:

$$\hat{y} = [0, 1]^n \tag{6}$$

where n is the number of possible actions to take.

Softmax Function

In order to achieve the compression, we apply softmax to normalize the values so that not only they compress to $[0, 1]$, they also all sum up to 1:

$$\sigma(x_i) = \frac{e^{x_i}}{\sum e^{x_i}} \tag{7}$$

where the values of the individual outputs are raised to powers of e to get a vector of probability distribution over the possible actions:

$$\pi(a|s, \theta) = [\sigma(a_1), \sigma(a_2), \cdots, \sigma(a_n)]. \tag{8}$$

Policy Evaluation

We define performance measure of the policy/neural network as $J(\theta)$, so that we can compare them such that:

$$J_{\pi_1}(\theta) > J_{\pi_2}(\theta) \Rightarrow \text{We consider } J_{\pi_1}(\theta) \text{ is better than } J_{\pi_2}(\theta). \tag{9}$$

As such the optimal policy has the optimal paramaters θ that give us the maximal performance:

$$\pi_*(a|s,\theta) = \arg\max_{\theta} J(\theta). \tag{10}$$

We obtain the performance estimate $\hat{J}(\theta)$ from experience samples during the simulations, and we perform the Stochastic Gradient Ascent to improve the performance with regards to its parameters:

$$\theta_{t+1} = \theta_t + \alpha \nabla J(\theta) \tag{11}$$

where

$$\nabla J(\theta) = [\frac{\partial \hat{J}(\theta)}{\partial \theta_1}, \frac{\partial \hat{J}(\theta)}{\partial \theta_2}, \cdots, \frac{\partial \hat{J}(\theta)}{\partial \theta_n}]. \tag{12}$$

Policy Gradient Theorem

We define the performance of the policy as:

$$J(\theta) = v_\pi(S_0). \tag{13}$$

The gradient of the policy's performance can be derived to be the gradient of the actions' probabilities. It means that the policy performance J changes wrt to the action probabilities π which in turn changes wrt to the parameter weights θ:

$$\nabla J(\theta) \propto \sum_s \mu(s) \sum_a q_\pi(s,a) \nabla \pi(a|s,\theta) \tag{14}$$

where $\mu(s)$ is the state distribution following policy π.

REINFORCE

Combination of Policy Gradient and Monte Carlo methods where we wait until the end of the simulation to generate the actual rewards from the state and action pairs of the actual experience gathered rather than bootstrapped and temporal-differenced. Then, we adjust the weights of the neural networks using the actual rewards, and state and action vectors:

$$\theta_{t+1} = \theta_t + \alpha \nabla \hat{J}(\theta). \tag{15}$$

where $\hat{J}(\theta)$ is just the approximated gradient because the experience are only from 1 episode.

Using the Policy Graidient Theorem, the gradient of the performance is estimated by the gradient of the action probabilities:

$$\nabla \hat{J} = \gamma^t G_t \cdot \frac{\nabla \pi(A_t|S_t,\theta_t)}{\pi(A_t|S_t,\theta_t)} \tag{16}$$

where bigger return G_t means that the action probability carries more importance to improving the policy performance, and the gradient of the action probability $\nabla\pi(A_t|S_t,\theta_t)$ is normalized by $\pi(A_t|S_t,\theta_t)$ to scale down the gradients for highly popular actions.

Following the policy gradient derived, the parameter weights θ are then updated to improve the estimate policy performance:

$$\theta_{t+1} = \theta_t + \alpha\gamma^t G_t \cdot \frac{\nabla\pi(A_t|S_t,\theta_t)}{\pi(A_t|S_t,\theta_t)} \tag{17}$$

and since $\nabla \ln \pi(A_t|S_t,\theta) = \frac{\nabla\pi(A_t|S_t,\theta_t)}{\pi(A_t|S_t,\theta_t)}$, the final update will be:

$$\theta_{t+1} = \theta_t + \alpha\gamma^t G_t \cdot \nabla \ln \pi(A_t|S_t,\theta). \tag{18}$$

Entropy Regularization

Since the action probabilities are directly outputted by the neural network, the mechanism for choosing the actions are abstracted away from us. In order to ecourage exploration, we incentivse the agent to keep the entropy of its policy as high as possible:

$$H(X) = -\sum_{x\in X} p(x)\cdot \ln p(x) \tag{19}$$

which measures the level of uncertainty of a random variable.

Thus, the uncertainty of the action to be selected by the policy for a state can be expressed as:

$$H_\pi(A_t) = -\sum_{x\in X} \pi(a|S_t)\cdot \ln \pi(a|S_t) \tag{20}$$

and the gradient of which is added to to SGD update as:

$$\theta_{t+1} = \theta_t + \alpha[\gamma^t G_t \cdot \nabla \ln \pi(A_t|S_t,\theta) + \beta\nabla H(\pi)]. \tag{21}$$