Mahindra™ University
Global Thinkers. Engaged Leaders.

Reinforcement Learning and Autonomous Systems (CS4122)

Lecture 34 (13/11/2024)
Lecture 35 (18/11/2024)
Lecture 36 (19/11/2024)
Lecture 37 (20/11/2024)

Instructor: Gourav Saha

Basic Idea of Training Policy Gradient RL

- In lectures 32 and 33, we discussed that in policy gradient RL, the policy π is captured by a neural network parameterized by θ.
  - The **input** to the neural network is the **state $\boldsymbol{x}$**.
  - The **output** of the neural network is **probability of taking different actions**. So here, **actions are the classes** from the perspective of ML/DL. And, the output layer of the neural network should have softmax activation.

So essentially the policy can be written as $\boldsymbol{\pi(a|x,\theta)}$.

Basic Idea of Training Policy Gradient RL

- In policy gradient we fix the structure of the policy $\pi(a|x,\theta)$. This essentially means that we **fix the neural network** architecture then learn the parameters θ of policy $\pi(a|x,\theta)$ using **gradient ascent** as follows,

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta) \tag{1}$$

where,

- α is the learning rate.
- This $+$ sign is because we are doing gradient ascent (NOT descent).
- $J(\theta)$ is the expected return under the current policy $\pi(a|x,\theta)$ which is parameterized by θ. NOTE: Since the structure of the neural network is fixed, the parameters θ completely characterizes the policy.
- $\nabla_\theta J(\theta)$ the gradient of the $J(\theta)$ with respect to θ.

Basic Idea of Training Policy Gradient RL

- Gradient $\nabla_\theta J(\theta)$ can be estimated using the following formula that we derived in lectures 32 and 33,

$$\nabla_\theta J(\theta) \approx \nabla_\theta \left(\frac{1}{N} \sum_{n=1}^{N} \sum_{t=0}^{T_n} \log\left(\pi\left(a_{n,t} | x_{n,t}, \theta\right)\right) \hat{Q}_{n,t} \right) \tag{2}$$

where $\hat{Q}_{n,t}$ given by the following formula is called "**reward-to-go**" for the n^{th} sampled trajectory starting from time slot t of the n^{th} sampled trajectory,

$$\hat{Q}_{n,t} = \sum_{k=t}^{T} \beta^k r_{n,k} \tag{3}$$

$$= \beta^t \sum_{k=t}^{T} \beta^{k-t} r_{n,k} \tag{4}$$

Basic Idea of Training Policy Gradient RL

$$\nabla_\theta J(\theta) \approx \nabla_\theta \left(\frac{1}{N} \sum_{n=1}^{N} \sum_{t=0}^{T_n} \log\left(\pi\left(a_{n,t} | x_{n,t}, \theta\right)\right) \hat{Q}_{n,t} \right)$$

$$\hat{Q}_{n,t} = \beta^t \sum_{k=t}^{T} \beta^{k-t} r_{n,k}$$

- In order to estimate the gradient $\nabla_\theta J(\theta)$ we need to sample N trajectories.
- For the n^{th} trajectory, the sample is the sequence $\{x_{n,t}, a_{n,t}, r_{n,t}\}_{t=1}^{T_n}$.
- How to generate the sequence $\{x_{n,t}, a_{n,t}, r_{n,t}\}_{t=1}^{T_n}$?

Pseudocode to Sample Trajectories

Given: A policy given by neural network, $\pi(\cdot \mid \cdot, \theta)$, and the number of trajectories N.

(S1): Create an empty list τ that will contain N trajectories.

(S2): For $i = 1, 2, \cdots, N$:

(S3): Reset the environment to get the initial state x_0. Initialize time $t = 0$, and an empty list τ_i that will contain the trajectory for the current episode.

(S4): while episode did not end:

(S5): Choose action a_t according to the probability distribution $\pi(\cdot \mid x_t, \theta)$.

(S6): Take action a_t. Environment will return reward r_t and transition to next state x_{t+1}.

(S7): Append the state, action, reward pair (x_t, a_t, r_t) to τ_i. Set $t = t + 1$.

(S8): Append list τ_i to list τ.

(S9): Return trajectory τ.

Pseudocode of Training Policy Gradient RL

Loss function for Keras

$$\nabla_\theta J(\theta) \approx \nabla_\theta \left(\frac{1}{N} \sum_{n=1}^{N} \sum_{t=0}^{T_n} \log\left(\pi\left(a_{n,t}|x_{n,t},\theta\right)\right) \hat{Q}_{n,t} \right)$$

$$\hat{Q}_{n,t} = \beta^t \sum_{k=t}^{T} \beta^{k-t} r_{n,k}$$

$$J(\theta) \approx \frac{1}{N} \sum_{(n,t) \in S_N \times S_T} \log\left(\pi(a_{n,t}|x_{n,t},\theta)\right) T \hat{\theta}_{n,t}$$

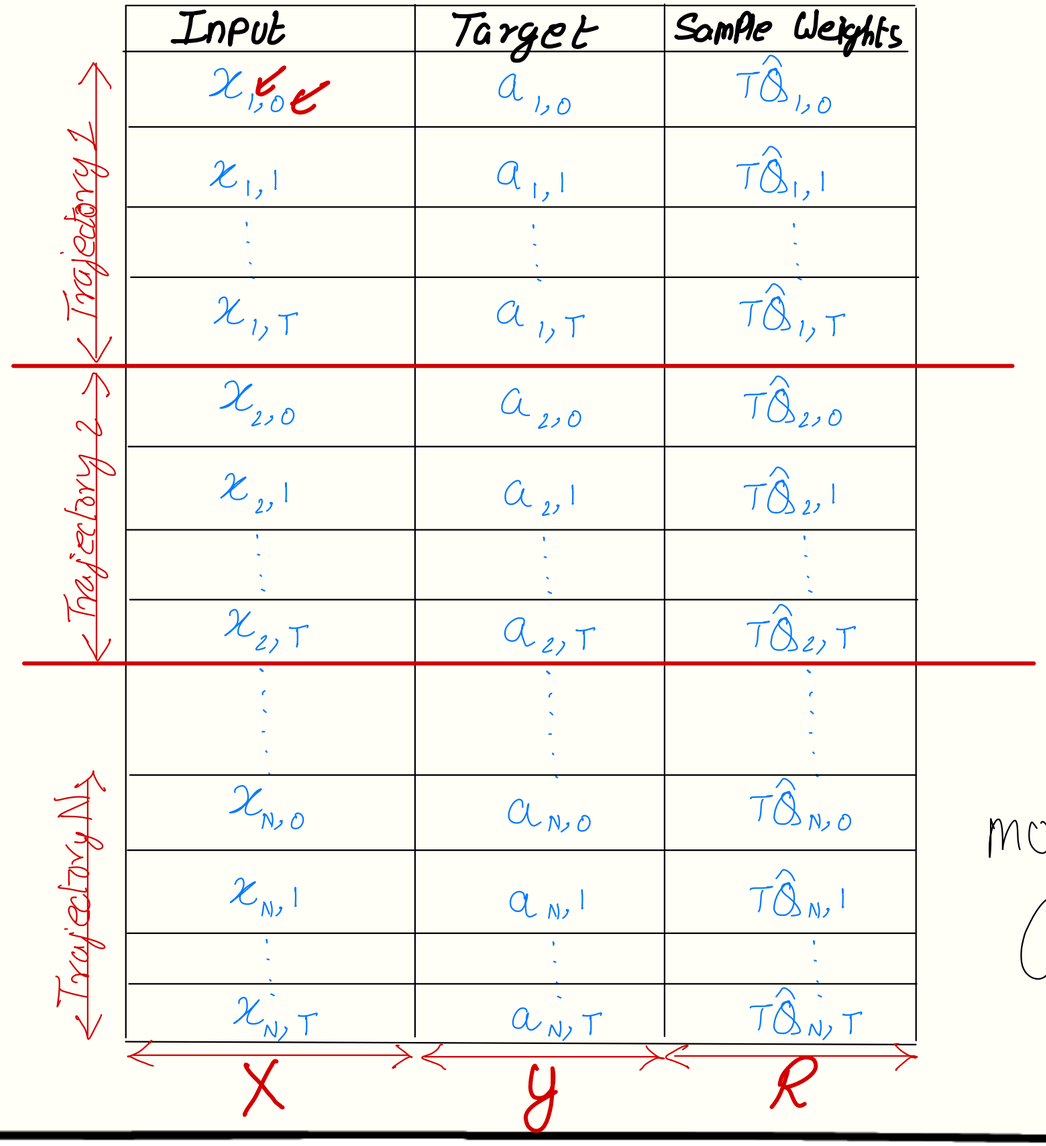
where $\hat{\theta}_{n,t} = \beta^t \sum_{k=t}^{T} \beta^{k-t} r_{n,k}$

$S_N = \{1, 2, \ldots, N\}$

$S_T = \{0, 1, \ldots, T\}$

Pseudocode of Training Policy Gradient RL

What does training data look like?



model.train_on_batch

(X, y, sample_weight=R)

Pseudocode of Training Policy Gradient RL

Algorithm 1: Psuedocode for Deep Policy Gradient RL

1 Initialize a neural network $\pi(\cdot\,;\theta)$ whose output is "probability like" (mostly a softmax). Then input to this neural network is the state. The number of ouptuts of the neural network is equal to the number of actions. $\pi(\cdot\,;\theta)$ represents the policy.

2 Set the maximum trajectory length, T, and the number of sample trajectories, N.

3 **for** *every episode until convergence* **do**

4 Sample N trajectories/episodes using the current policy $\pi(\cdot\,;\theta)$ and save the tuple (x, a, r) for every time slot and every trajectory in a buffer, B. The environment must be reset in the beginning of every trajectory/episode. We must collect (x, a, r) for T time slots for every episode (not more nor less). If the episode ends before T time slots, like in episodic case, the remaining time slots must be filled with dummy states, dummy actions, and **zero** reward.

5 Using the sampled trajectories, generate the training batch: $\left\{x_{n,t}\,,\ a_{n,t}\,,\quad T\hat{Q}_{n,t}\right\}$ for all $n \in \{1,\ldots,N\}$ and $t \in \{0,\ldots,T\}$ as shown in the table in the previous page. Note that $\hat{Q}_{n,t}$ is calculated using equation (4).

6 Use batch training data to update the parameters θ of the neural network by taking one gradient descent step.

> On or Off policy?

> `model.train_on_batch(X, y, sample_weight=R)`

> If $N=1$, the algo is called REINFORCE.

Only one trajectory/episode.

← Multiple gradient ascent per batch.

Reducing Variance of Policy Gradient RL

1) **Using causality**: We derived both these expressions in lecture 32 and 33 notes.

Without using causality

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{n=1}^{N} \sum_{t=0}^{T} \nabla_\theta \log(\pi(a_{n,t}|x_{n,t},\theta)) \left(\sum_{K=0}^{T} \beta^K r_K \right)$$

Using causality

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{n=1}^{N} \sum_{t=0}^{T} \nabla_\theta \log(\pi(a_{n,t}|x_{n,t},\theta)) \left(\sum_{K=t}^{T} \beta^K r_K \right)$$

> r_K is a random variable. By using causality, we are summing over a fewer number of random variables. It is likely (not always the case) that sum over a fewer number of random variables have less variance.

Reducing Variance of Policy Gradient RL

2) Baselines

› We can prove that,

$$\nabla_\theta J(\theta) = E_{P[\pi_T;\theta]}\left[\sum_{t=0}^{T} \nabla_\theta \log\left(\pi(a_t|x_t,\theta)\right) \beta^t \left(\sum_{k=t}^{T} \beta^{k-t} r_k\right)\right]$$

$$= E_{P[\pi_T;\theta]}\left[\sum_{t=0}^{T} \nabla_\theta \log\left(\pi(a_t|x_t,\theta)\right) \beta^t \left(\sum_{k=t}^{T} \beta^{k-t} r_k - h(x_t)\right)\right]$$

where $h(x_t)$ is called the BASELINE. $h(x_t)$ is either a constant, or a function of state. $h(x_t)$ can't be a function of actions.

Reducing Variance of Policy Gradient RL

2) Baselines

$$\nabla_\theta J(\theta) = E_{p[\tau_T;\theta]}\left[\sum_{t=0}^{T} \nabla_\theta \log\left(\pi(a_t|x_t,\theta)\right) \beta^t \left(\sum_{k=t}^{T} \beta^{k-t} r_k - h(x_t)\right)\right]$$

$$\approx \frac{1}{N}\sum_{n=1}^{N}\sum_{t=0}^{T} \nabla_\theta \log\left(\pi(a_{n,t}|x_{n,t},\theta)\right) \hat{A}_{n,t}$$

where,

$$\hat{A}_{n,t} = \beta^t\left(\sum_{k=t}^{T} \beta^{k-t} r_{n,k} - h(x_{n,t})\right)$$

where $\hat{A}_{n,t}$ is called the ADVANTAGE.

Reducing Variance of Policy Gradient RL

2) Baselines

- So, to incorporate baseline in the psuedocode for policy gradient RL, we simply replace $\hat{Q}_{n,t}$ with $\hat{A}_{n,t}$.

  Everything else remains the same.

Reducing Variance of Policy Gradient RL

2) Baselines

But still, two questions remains:

- First question: Why to use baseline?

  The answer is that baseline helps in reducing variance in the estimate of the gradient $\nabla_\theta J(\theta)$.

Reducing Variance of Policy Gradient RL

2) Baselines

But still, two questions remains:

> Second question: How to choose baselines in order to reduce the variance in the estimate of $\nabla_\theta J(\theta)$?

1) Constanst baseline: $$b_{n,t} = \frac{1}{N} \sum_{n=1}^{N} \sum_{K=0}^{t} \beta^t r_{n,K}$$

2) State-dependent baseline: $$b(x_{n,t}) = V(x_{n,t})$$

$\longleftrightarrow$ The value function.

Reducing Variance of Policy Gradient RL

2) Baselines

> But how does the two baselines mentioned in the previous page helps in reducing variance?

Answer: First, the intuitive answer.

$$\hat{A}_{n,t} = \beta^t \left(\sum_{K=t}^{T} \beta^{K-t} r_{n,K} - h(x_{n,t}) \right)$$

Here is a 5 min YouTube link that explains the intuition behind baselines: https://youtu.be/-_N7aehsj3g

Reducing Variance of Policy Gradient RL

2) Baselines

$$\hat{A}_{n,t} = \beta^t \left(\sum_{K=t}^{T} \beta^{K-t} r_{n,K} - h_{n,t} \right)$$

$$h_{n,t} = \frac{1}{N} \sum_{n=1}^{N} \sum_{K=0}^{t} \beta^t r_{n,K}$$

$$\hat{A}_{n,t} = \beta^t \left(\sum_{K=t}^{T} \beta^{K-t} r_{n,K} - h(x_{n,t}) \right)$$

$$h(x_{n,t}) = V(x_{n,t})$$

Reducing Variance of Policy Gradient RL

2) Baselines

- Now we provide a more mathematical answer to how baselines, if chosen properly, can help in reducing variance.

- The paper which analyzed baselines and its effect on variance is titled "Variance Reduction Techniques for Gradient Estimates in Reinforcement Learning", JMLR 2004. This is a very detailed paper. We are only going to scratch the surface of it.

Reducing Variance of Policy Gradient RL

2) Baselines

- The main technique used in this paper is statistical technique that originated around 1965 called "control variates". Let's forget the RL setup for now and focus on the overall idea of this technique.

Reducing Variance of Policy Gradient RL

2) Baselines (Control Variates)

- Say that there is a random variable X whose expected value is μ_x,

$$E[X] = \mu_x$$

We don't know μ_x. We want to find it. One approach is the straightforward: Use sample estimate,

$$\bar{x} = \frac{1}{N} \sum_{K=1}^{N} x_K$$

Reducing Variance of Policy Gradient RL

2) Baselines (Control Variates)

> Now, we discuss another approach. Consider another random variable Y. We have,

$$E[Y] = \mu_y.$$

μ_y is known. We can write,

$$X = X - Y + Y$$

$$\Rightarrow E[X] = E[X-Y] + E[Y]$$

$$\Rightarrow \mu_x = E[X-Y] + \mu_y$$

Unknown — Just estimate this term to find μ_x. — Known

Reducing Variance of Policy Gradient RL

2) Baselines (Control Variates)

> So, we have two approaches to compute μ_x,

Approach 1:

$$\bar{x} = \frac{1}{N} \sum_{K=1}^{N} x_K$$

Approach 2:

$$\tilde{x} = \frac{1}{N} \sum_{K=1}^{N} (x_K - y_K) + \mu_y$$

Now, let's compute the variance of $\bar{x}$ and $\tilde{x}$.

Reducing Variance of Policy Gradient RL

2) Baselines (Control Variates)

> Variance of $\tilde{x}$,

$\mathrm{Var}[\tilde{x}]$

$$= \mathrm{Var}\left[\frac{1}{N}\sum_{k=1}^{N}(x_k - y_k) + \mu_y\right]$$

$$= \mathrm{Var}\left[\frac{1}{N}\sum_{k=1}^{N}(x_k - y_k)\right]$$

$$= \frac{1}{N^2}\mathrm{Var}\left[\sum_{k=1}^{N}(x_k - y_k)\right]$$

$$= \frac{1}{N^2}\sum_{k=1}^{N}\mathrm{Var}[x_k - y_k]$$

x_k's are independent.

y_k's are independent.

$$= \frac{1}{N^2}\cdot N \cdot \mathrm{Var}[x_k - y_k]$$

x_k's are identical.

y_k's are identical.

$$= \frac{1}{N}\cdot \mathrm{Var}[x_k - y_k]$$

Reducing Variance of Policy Gradient RL

2) Baselines (Control Variates)

> Variance of $\tilde{x}$,

$$\text{Var}[\tilde{x}]$$

$$= \frac{1}{N} \cdot \text{Var}[x_k - y_k]$$

Covariance

$$= \frac{1}{N}\left(\text{Var}[x_k] + \text{Var}[y_k] - 2\,\text{Cov}[x_k, y_k]\right)$$

Reducing Variance of Policy Gradient RL

2) Baselines (Control Variates)

- Variance of $\tilde{x}$,

$$\text{Var}[\tilde{x}] = \frac{1}{N} \cdot \text{Var}[x_k - y_k]$$

$$= \frac{1}{N}\left(\text{Var}[x_k] + \text{Var}[y_k] - 2\,\text{cov}[x_k, y_k]\right)$$

Covariance

- By using the same steps we can find that variance of $\bar{x}$ is,

$$\text{Var}[\bar{x}] = \frac{1}{N}\left(\text{Var}[x_k]\right)$$

Reducing Variance of Policy Gradient RL

2) Baselines (Control Variates)

$$\mathrm{Var}[\tilde{x}] = \frac{1}{N}\left(\mathrm{Var}[x_k] + \mathrm{Var}[y_k] - 2\,\mathrm{Cov}[x_k, y_k]\right)$$

$$\mathrm{Var}[\bar{x}] = \frac{1}{N}\left(\mathrm{Var}[x_k]\right)$$

- $\mathrm{Var}[\tilde{x}]$ is less than $\mathrm{Var}[\bar{x}]$ if this term is negative which is possible when $\mathrm{Cov}[x_k, y_k]$ is positive enough, i.e. x_k and y_k are positively correlated.

Reducing Variance of Policy Gradient RL

2) Baselines (Control Variates)

$$\tilde{x} = \frac{1}{N} \sum_{k=1}^{N} (x_k - y_k) + \mu_y$$

- Intuitively, when x_k and y_k are positively correlated, then a noise that increases x_k also increases y_k. Hence, when we use approach 2, the noise in these two terms cancels each other out which in turn helps in reducing variance.

Reducing Variance of Policy Gradient RL

2) Baselines (Control Variates)

> Now, we extend the concept of control variates to variance reduction of the estimate of $\nabla_\theta J(\theta)$,

$$\nabla_\theta J(\theta) = E_{P[\tau_T;\theta]}\left[\sum_{t=0}^{T} \nabla_\theta \log\left(\pi(a_t|x_t,\theta)\right) \beta^t \left(\sum_{k=t}^{T} \beta^{k-t} r_k - h(x_t)\right)\right]$$

$$= E_{P[\tau_T;\theta]}\left[\sum_{t=0}^{T} \nabla_\theta \log\left(\pi(a_t|x_t,\theta)\right) \beta^t \left(\sum_{k=t}^{T} \beta^{k-t} r_k\right)\right]$$

$$- E_{P[\tau_T;\theta]}\left[\sum_{t=0}^{T} \nabla_\theta \log\left(\pi(a_t|x_t,\theta)\right) \beta^t \left(h(x_t)\right)\right]$$

Reducing Variance of Policy Gradient RL

2) Baselines (Control Variates)

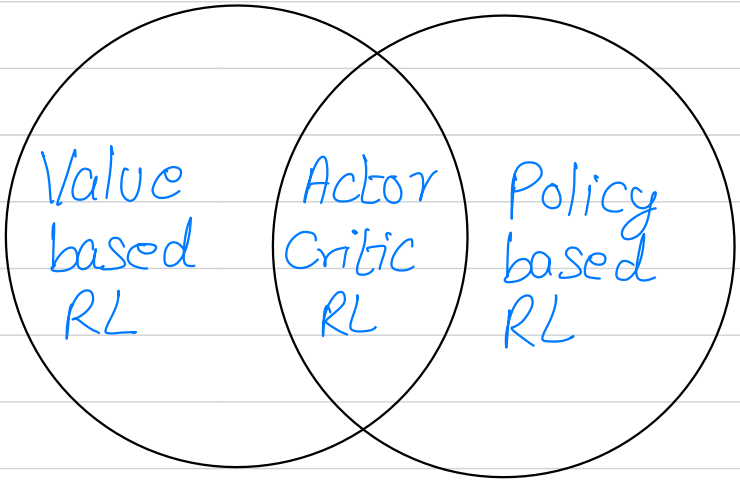
$$\nabla_\theta J(\theta) = E_{P[\tau_T;\theta]}\left[\sum_{t=0}^{T} \nabla_\theta \log\left(\pi(a_t|x_t,\theta)\right) \beta^t \left(\sum_{k=t}^{T} \beta^{k-t} r_k - h(x_t)\right)\right]$$

$$= E_{P[\tau_T;\theta]}\left[\sum_{t=0}^{T} \nabla_\theta \log\left(\pi(a_t|x_t,\theta)\right) \beta^t \left(\sum_{k=t}^{T} \beta^{k-t} r_k\right)\right]$$ } This is similar to X.

$$- E_{P[\tau_T;\theta]}\left[\sum_{t=0}^{T} \nabla_\theta \log\left(\pi(a_t|x_t,\theta)\right) \beta^t \left(h(x_t)\right)\right]$$ } This is similar to Y. Also, γ_y is ZERO. Why?

Actor-Critic RL

- Falls under policy gradient RL. But shares similarity with value-based method because it estimates the value functions.



- In policy gradient RL that we saw till now, we had to collect the entire trajectory before we can update the policy. This is similar to Monte-Carlo (MC) approach that we saw in module 3.

- Actor-Critic RL is the temporal difference (TD) version of the "vanilla" policy-gradient RL that we saw till now.

Remember the entire philosophy of TD approach:

- Step 1: Take an action, a. Collect reward, r, and observe next state, x'.
- Step 2: Make an estimate of the return starting from the next state. This estimate along with r will give an estimate of the return starting from current state x.
- Step 3: Use this estimate to update the policy.

$$\nabla_\theta J(\theta) = E_{P[\pi_T;\theta]}\left[\sum_{t=0}^{T} \nabla_\theta \log(\pi(a_t|x_t,\theta))\, \beta^t \left(\sum_{K=t}^{T} \beta^{K-t} r_K - h(x_t)\right)\right]$$

- How to use TD approach for policy gradient?

Answer: Consider this term. It is the return starting from time t, G_t, under current policy, π. We have,

$$G_t = \sum_{K=t}^{T} \beta^{K-t} r_K$$

$$= r_t + \beta r_{t+1} + \beta^2 r_{t+2} + \cdots$$

$$= r_t + \beta(r_{t+1} + \beta r_{t+2} + \cdots)$$

$$\approx r_t + \beta V^{\pi_\theta}(x_{t+1})$$

$$\nabla_\theta J(\theta) = E_{P[\tau_T;\theta]}\left[\sum_{t=0}^{T} \nabla_\theta \log\left(\pi(a_t|x_t,\theta)\right) \beta^t \left(\sum_{K=t}^{T} \beta^{K-t} r_K - h(x_t)\right)\right]$$

- How to use TD approach for policy gradient?

  Answer: Replace this term with $r_t + \beta V^{\pi_\theta}(x_{t+1})$.

- Choose baseline $h(x_t) = V^{\pi_\theta}(x_t)$.

- We get the following,

$$\nabla_\theta J(\theta) \approx E_{P[\tau_T;\theta]}\left[\sum_{t=0}^{T} \nabla_\theta \log\left(\pi(a_t|x_t,\theta)\right) \beta^t \left(r_t + \beta V^{\pi_\theta}(x_{t+1}) - V^{\pi_\theta}(x_t)\right)\right]$$

$$\nabla_\theta J(\theta) \approx E_{p[\tau_T;\theta]}\left[\sum_{t=0}^{T} \nabla_\theta \log\left(\pi(a_t|x_t,\theta)\right) \beta^t \left(r_t + \beta V^{\pi_\theta}(x_{t+1}) - V^{\pi_\theta}(x_t)\right)\right]$$

$$\approx \frac{1}{N}\sum_{n=1}^{N}\sum_{t=0}^{T} \nabla_\theta \log\left(\pi(a_{n,t}|x_{n,t},\theta)\right) \hat{A}_{n,t}$$

where,

$$\hat{A}_{n,t} = \beta^t \left(r_{n,t} + \beta V^{\pi_\theta}(x_{n,t+1}) - V^{\pi_\theta}(x_{n,t})\right)$$

where $\hat{A}_{n,t}$ is called the ADVANTAGE.

$$\nabla_\theta J(\theta) \approx E_{p[\tau_T;\theta]}\left[\sum_{t=0}^{T} \nabla_\theta \log\left(\pi(a_t|x_t,\theta)\right) \beta^t \left(r_t + \beta V^{\pi_\theta}(x_{t+1}) - V^{\pi_\theta}(x_t)\right)\right]$$

$$\approx \nabla_\theta \log\left(\pi(a_t|x_t,\theta)\right) \hat{A}_t$$

where,

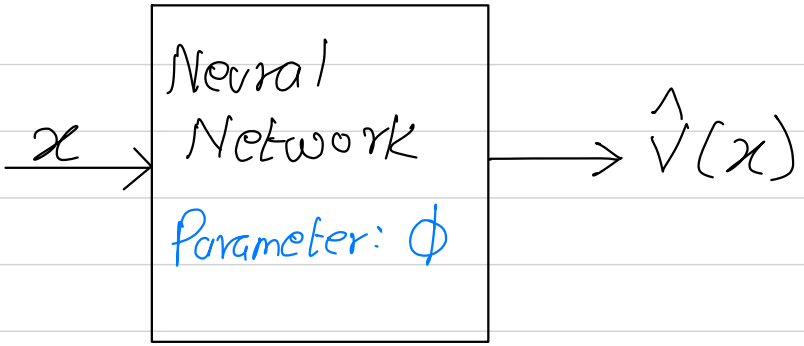
$$\hat{A}_t = \beta^t \left(r_t + \beta V^{\pi_\theta}(x_{t+1}) - V^{\pi_\theta}(x_t)\right)$$

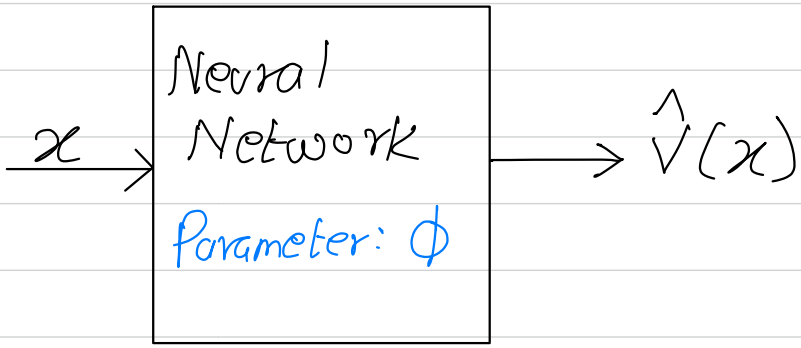
where $\hat{A}_t$ is called the ADVANTAGE.

But, one question still remains: How to compute $V^{\pi_\theta}(x)$?

Answer: Just the same way we were computing the Q-function in Deep Q-Learning. We approximate $V^{\pi_\theta}(x)$ with a neural network parameterized by ϕ,

$$V^{\pi_\theta}(x) \approx \hat{V}(x; \phi)$$





- How to train this neural network?

Answer:

- It is a regression problem. Use mean squared error as loss.
- Input: $x_{n,t}$ (or simply x).
- Target: $r_{n,t} + \beta \hat{V}(x_{n,t+1}; \phi)$ (or simply $r + \beta \hat{V}(x'; \phi)$ where x' is the next state).
  $\hat{V}(x_{n,t+1}; \phi)$ is obtained using current neural network.

Training DQN: Deep Policy Gradient RL

Algorithm 1: Psuedocode for Deep Policy Gradient RL

1 Initialize a neural network $\pi(\cdot\,;\,\theta)$ whose output is "probability like" (mostly a softmax). Then input to this neural network is the state. The number of ouptuts of the neural network is equal to the number of actions. $\pi(\cdot\,;\,\theta)$ represents the policy.

2 Set the maximum trajectory length, T, and the number of sample trajectories, N.

3 **for** *every episode until convergence* **do**

4 Sample N trajectories/episodes using the current policy $\pi(\cdot\,;\,\theta)$ and save the tuple (x, a, r) for every time slot and every trajectory in a buffer, B. The environment must be reset in the beginning of every trajectory/episode. We must collect (x, a, r) for T time slots for every episode (not more nor less). If the episode ends before T time slots, like in episodic case, the remaining time slots must be filled with dummy states, dummy actions, and **zero** reward.

5 Using the sampled trajectories, generate the training batch: $\left\{x_{n,t}\,,\, a_{n,t}\,,\quad T\hat{Q}_{n,t}\right\}$ for all $n \in \{1, \ldots, N\}$ and $t \in \{0, \ldots, T\}$ as shown in the table in the previous page. Note that $\hat{Q}_{n,t}$ is calculated using equation (4).

6 Use batch training data to update the parameters θ of the neural network by taking one gradient descent step.

Why descent?

Algorithm 2: Psuedocode for Actor-Critic RL

Initialize a neural network $\pi(\cdot\,;\,\theta)$ whose output is "probability like" (mostly a softmax). Then input to this neural network is the state. The number of ouptuts of the neural network is equal to the number of actions. $\pi(\cdot\,;\,\theta)$ represents the policy.

Initialize a neural network $\widehat{V}(\cdot;\phi)$ whose output is a scalar that is an estimate of the value function. Then input to this neural network is the state.

for *every episode until convergence* **do**

  Reset the environment to get the current state x.

  Set $w = 1$.

  for *every time slot of the episode until convergence* **do**

    Pick action, a, for current state, x, using the current policy $\pi(\cdot\,;\,\theta)$.

    Take action, a, and get reward, r, and next state, x'.

    Set the reward to go, $G = r + \beta\widehat{V}\left(x';\phi\right)$, and the advantage, $\widehat{A} = G - \widehat{V}(x;\phi)$.

    Training $\widehat{V}(\cdot;\phi)$: Create a batch of size one with input, x, and target, G Use this batch to train the neural network $\widehat{V}(\cdot;\phi)$ by taking one gradient descent step.

    Training $\pi(\cdot\,;\,\theta)$: Create a batch of size one with input, x, target, a, and sample weights, $w \cdot \widehat{A}$. Use this batch to train the neural network $\pi(\cdot\,;\,\theta)$ by taking one gradient descent step.

    Set $x \leftarrow x'$.

    Set $w \leftarrow \beta w$.

} Regression. Loss: MSE.

} Classification. Loss: Cross-entropy

Algorithm 2: Psuedocode for Actor-Critic RL

1 Initialize a neural network $\pi(\cdot\,;\,\theta)$ whose output is "probability like" (mostly a softmax). Then input to this neural network is the state. The number of ouptuts of the neural network is equal to the number of actions. $\pi(\cdot\,;\,\theta)$ represents the policy.

2 Initialize a neural network $\widehat{V}(\cdot;\phi)$ whose output is a scalar that is an estimate of the value function. Then input to this neural network is the state.

3 **for** *every episode until convergence* **do**

4 Reset the environment to get the current state x.

5 Set $w = 1$.

6 **for** *every time slot of the episode until convergence* **do**

7 Pick action, a, for current state, x, using the current policy $\pi(\cdot\,;\,\theta)$.

8 Take action, a, and get reward, r, and next state, $x^{'}$.

9 Set the reward to go, $\widehat{Q} = r + \beta\widehat{V}\left(x^{'};\phi\right)$, and the advantage, $\widehat{A} = \widehat{Q} - \widehat{V}(x;\phi)$.

10 Training $\widehat{V}(\cdot;\phi)$: Create a batch of size one with input, x, and target, $\widehat{Q}$. Use this batch to train the neural network $\widehat{V}(\cdot;\phi)$ by taking one gradient descent step.

11 Training $\pi(\cdot\,;\,\theta)$: Create a batch of size one with input, x, target, a, and sample weights, $w \cdot \widehat{A}$. Use this batch to train the neural network $\pi(\cdot\,;\,\theta)$ by taking one gradient descent step.

12 Set $x \leftarrow x^{'}$.

13 Set $w \leftarrow \beta w$.

> It is called actor-critic RL because:

actor network

i) The policy network $\pi(\cdot;\theta)$ is taking actions. Hence, actor.

Critic network

ii) The value function networ $\hat{V}(\cdot;\phi)$ is evaluating the goodness of the action by providing a baseline. Hence, critic.

Algorithm 2: Psuedocode for Actor-Critic RL

1 Initialize a neural network $\pi(\cdot\,;\,\theta)$ whose output is "probability like" (mostly a softmax). Then input to this neural network is the state. The number of ouptuts of the neural network is equal to the number of actions. $\pi(\cdot\,;\,\theta)$ represents the policy.

2 Initialize a neural network $\widehat{V}(\cdot;\phi)$ whose output is a scalar that is an estimate of the value function. Then input to this neural network is the state.

3 **for** *every episode until convergence* **do**

4 Reset the environment to get the current state x.

5 Set $w = 1$.

6 **for** *every time slot of the episode until convergence* **do**

7 Pick action, a, for current state, x, using the current policy $\pi(\cdot\,;\,\theta)$.

8 Take action, a, and get reward, r, and next state, x'.

9 Set the reward to go, $\widehat{Q} = r + \beta\widehat{V}\left(x';\phi\right)$, and the advantage, $\widehat{A} = \widehat{Q} - \widehat{V}(x;\phi)$.

10 Training $\widehat{V}(\cdot;\phi)$: Create a batch of size one with input, x, and target, $\widehat{Q}$. Use this batch to train the neural network $\widehat{V}(\cdot;\phi)$ by taking one gradient descent step.

11 Training $\pi(\cdot\,;\,\theta)$: Create a batch of size one with input, x, target, a, and sample weights, $w \cdot \widehat{A}$. Use this batch to train the neural network $\pi(\cdot\,;\,\theta)$ by taking one gradient descent step.

12 Set $x \leftarrow x'$.

13 Set $w \leftarrow \beta w$.

> The policy network $\pi(\cdot;\theta)$ is the final deliverable.

> Three points:

i) $\hat{A}_{n,t} = \beta^t\left(r_{n,t} + \beta V^{\pi_\theta}(x_{n,t+1}) - V^{\pi_\theta}(x_{n,t})\right)$

β^t can be ignored if minimizing the discounted reward is NOT our true goal (we discussed this before).

Remove line 13 if you want to ignore β^t.

Algorithm 2: Psuedocode for Actor-Critic RL

1 Initialize a neural network $\pi(\cdot\,;\,\theta)$ whose output is "probability like" (mostly a softmax). Then input to this neural network is the state. The number of ouptuts of the neural network is equal to the number of actions. $\pi(\cdot\,;\,\theta)$ represents the policy.

2 Initialize a neural network $\widehat{V}(\cdot;\phi)$ whose output is a scalar that is an estimate of the value function. Then input to this neural network is the state.

3 **for** *every episode until convergence* **do**

4 Reset the environment to get the current state x.

5 Set $w = 1$.

6 **for** *every time slot of the episode until convergence* **do**

7 Pick action, a, for current state, x, using the current policy $\pi(\cdot\,;\,\theta)$.

8 Take action, a, and get reward, r, and next state, x'.

9 Set the reward to go, $\widehat{Q} = r + \beta\widehat{V}\left(x';\phi\right)$, and the advantage, $\widehat{A} = \widehat{Q} - \widehat{V}(x;\phi)$.

10 Training $\widehat{V}(\cdot;\phi)$: Create a batch of size one with input, x, and target, $\widehat{Q}$. Use this batch to train the neural network $\widehat{V}(\cdot;\phi)$ by taking one gradient descent step.

11 Training $\pi(\cdot\,;\,\theta)$: Create a batch of size one with input, x, target, a, and sample weights, $w \cdot \widehat{A}$. Use this batch to train the neural network $\pi(\cdot\,;\,\theta)$ by taking one gradient descent step.

12 Set $x \leftarrow x'$.

13 Set $w \leftarrow \beta w$.

> The policy network $\pi(\cdot;\theta)$ is the final deliverable.

> Three points:

ii) Can we use replay buffer? No! Not atleast directly because policy gradient is an on-policy RL algo. Hence, we can't use an $\langle x, a, r, x' \rangle$ collected from an old policy.

Algorithm 2: Psuedocode for Actor-Critic RL

1 Initialize a neural network $\pi(\cdot\,;\,\theta)$ whose output is "probability like" (mostly a softmax). Then input to this neural network is the state. The number of ouptuts of the neural network is equal to the number of actions. $\pi(\cdot\,;\,\theta)$ represents the policy.

2 Initialize a neural network $\widehat{V}(\cdot;\phi)$ whose output is a scalar that is an estimate of the value function. Then input to this neural network is the state.

3 **for** *every episode until convergence* **do**

4 Reset the environment to get the current state x.

5 Set $w = 1$.

6 **for** *every time slot of the episode until convergence* **do**

7 Pick action, a, for current state, x, using the current policy $\pi(\cdot\,;\,\theta)$.

8 Take action, a, and get reward, r, and next state, x'.

9 Set the reward to go, $\widehat{Q} = r + \beta\widehat{V}\left(x';\phi\right)$, and the advantage, $\widehat{A} = \widehat{Q} - \widehat{V}(x;\phi)$.

10 Training $\widehat{V}(\cdot;\phi)$: Create a batch of size one with input, x, and target, $\widehat{Q}$. Use this batch to train the neural network $\widehat{V}(\cdot;\phi)$ by taking one gradient descent step.

11 Training $\pi(\cdot\,;\,\theta)$: Create a batch of size one with input, x, target, a, and sample weights, $w \cdot \widehat{A}$. Use this batch to train the neural network $\pi(\cdot\,;\,\theta)$ by taking one gradient descent step.

12 Set $x \leftarrow x'$.

13 Set $w \leftarrow \beta w$.

> The policy network $\pi(\cdot;\theta)$ is the final deliverable.

> Three points:

ii) Can we use target network?

Yes!

A Comment About Reward-to-Go and Advantage

> Consider the estimates of $\nabla_\theta J(\theta)$ shown below,

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{n=1}^{N} \sum_{t=0}^{T} \nabla_\theta \log\left(\pi(a_{n,t} | x_{n,t}, \theta)\right) \hat{\Theta}_{n,t}$$

where $\hat{\Theta}_{n,t} = \beta^t \sum_{k=t}^{T} \beta^{k-t} r_{n,k}$

Policy Gradient RL

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{n=1}^{N} \sum_{t=0}^{T} \nabla_\theta \log\left(\pi(a_{n,t} | x_{n,t}, \theta)\right) \hat{A}_{n,t}$$

where, $\hat{A}_{n,t} = \beta^t \left(\sum_{k=t}^{T} \beta^{k-t} r_{n,k} - h(x_{n,t}) \right)$

Policy Gradient RL with Baseline

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{n=1}^{N} \sum_{t=0}^{T} \nabla_\theta \log\left(\pi(a_{n,t} | x_{n,t}, \theta)\right) \hat{A}_{n,t}$$

where, $\hat{A}_{n,t} = \beta^t \left(r_{n,t} + \beta V^{\pi_\theta}(x_{n,t+1}) - V^{\pi_\theta}(x_{n,t}) \right)$

Actor-Critic RL

A Comment About Reward-to-Go and Advantage

- In many implementation of policy-gradient RL and actor-critic RL you will find that β^t, which is highlighted in GREEN in the previous page, is NOT included.

- This is because in many real-life applications (except say finance), we are NOT interested in discounted reward. But we still use discounted reward because for infinite horizon setting, discounted reward leads to finite return which is NOT possible if we use sum-of-reward.

A Comment About Reward-to-Go and Advantage

$$\hat{Q}_{n,t} = \beta^t \sum_{k=t}^{T} \beta^{k-t} r_{n,k}$$

$$\hat{A}_{n,t} = \beta^t \left(\sum_{k=t}^{T} \beta^{k-t} r_{n,k} - h(x_{n,t}) \right)$$

$$\hat{A}_{n,t} = \beta^t \left(r_{n,t} + \beta V^{\pi_\theta}(x_{n,t+1}) - V^{\pi_\theta}(x_{n,t}) \right)$$

- Now, whether we use β^t or NOT $\hat{Q}_{n,t}$ and $\hat{A}_{n,t}$ are still going to be finite. Hence, we can ignore β^t.

A Comment About Reward-to-Go and Advantage

$$\hat{Q}_{n,t} = \beta^t \sum_{k=t}^{T} \beta^{k-t} r_{n,k}$$

$$\hat{A}_{n,t} = \beta^t \left(\sum_{k=t}^{T} \beta^{k-t} r_{n,k} - h(x_{n,t}) \right)$$

$$\hat{A}_{n,t} = \beta^t \left(r_{n,t} + \beta V^{\pi_\theta}(x_{n,t+1}) - V^{\pi_\theta}(x_{n,t}) \right)$$

- Also $\beta^t < 1$ and β^t decreases with t. This in turn reduces $\hat{Q}_{n,t}$ and $\hat{A}_{n,t}$ as t increases which indirectly implies that the future actions are NOT that relevant because they will anyway contribute to a small net reward. In most settings future actions are as relevant as present. So, another reason to ignore β^t.

A Comment About Reward-to-Go and Advantage

$$\hat{Q}_{n,t} = \beta^t \sum_{k=t}^{T} \beta^{k-t} r_{n,k}$$

$$\hat{A}_{n,t} = \beta^t \left(\sum_{k=t}^{T} \beta^{k-t} r_{n,k} - b(x_{n,t}) \right)$$

$$\hat{A}_{n,t} = \beta^t \left(r_{n,t} + \beta V^{\pi_\theta}(x_{n,t+1}) - V^{\pi_\theta}(x_{n,t}) \right)$$

- That said, if our true objective is discounted reward, we MUST use β^t.


Thank you