Reinforcement Learning

Chapter 6: Actor Critic Methods

Ali Bereyhi

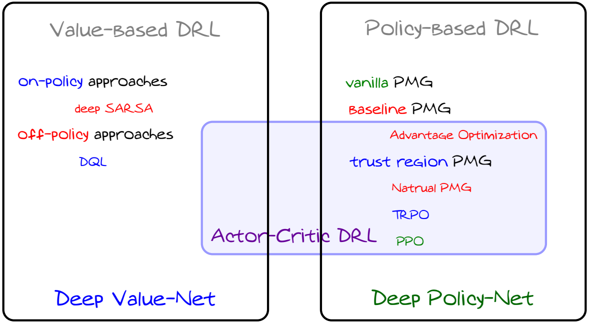
ali.bereyhi@utoronto.ca

Department of Electrical and Computer Engineering

University of Toronto

Fall 2025

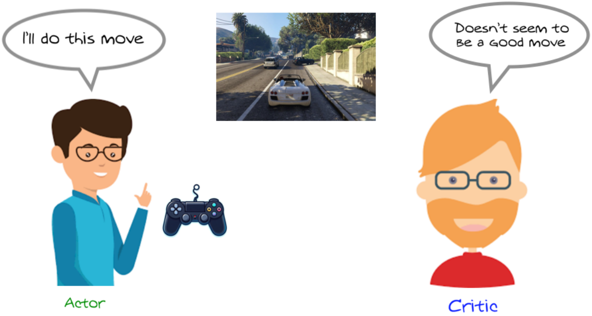
Deep RL: *Sort of Division*



Deep RL: *Sort of Division*

In actor-critic approaches we have both networks

- *an actor has a policy network*
  - ↳ *This network enables it to act at each particular state*
- *a critic has a value network*
  - ↳ *This network enables it to evaluate its policy*
  - ↳ *The evaluation will help improving the policy policy*



Deep RL: *Sort of Division*

Attention

For many people actor-critic $\equiv$ PGM: they usually argue that

- *to implement a PGM we need to estimate values*
- *we should do it by a value network*

So, any PGM is at the end actor-critic

That's practically true; however, in principle, we can

implement PGMs via basic Monte Carlo

So, we could also have a pure PGM, e.g., REINFORCE!

Implementing PGMs

Let's get back to PGMs: say we want to implement a PGM

- *We usually use sample advantages, i.e.,*

$$U_t = R_{t+1} + \gamma v_{\pi_{\boldsymbol{\theta}}} \left(S_{t+1} \right) - v_{\pi_{\boldsymbol{\theta}}} \left(S_t \right)$$

So, we need to know the value function $v_{\pi_{\boldsymbol{\theta}}} \left(\cdot \right)$ of our policy $\pi_{\boldsymbol{\theta}}$

+ *Well, why don't we evaluate it once and use it forever?*
- *Attention! We need this evaluation each time we update policy $\pi_{\boldsymbol{\theta}}$!*
+ *How exactly we do it then? You promised to tell us!*
- *Sure! Let's use what we have learned up to now*

Advantage PGM: *Implementing*

Let's look at the classic advantage optimization PGM

AdvantagePGM():

Initiate with $\boldsymbol{\theta}$ and learning rate α
while *interacting* **do**
  Set $\hat{\nabla} = \mathbf{0}$
  for *mini-batch* $b = 1 : B$ **do**
    Sample $S_0, A_0 \xrightarrow{R_1} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T$ *with policy* $\pi_{\boldsymbol{\theta}}$
    for $t = 0 : T-1$ **do**
      Compute sample advantage $U_t = R_{t+1} + \gamma v_{\pi_{\boldsymbol{\theta}}}(S_{t+1}) - v_{\pi_{\boldsymbol{\theta}}}(S_t)$
      Compute sample gradient $\hat{\nabla} \leftarrow \hat{\nabla} + U_t \nabla \log \pi_{\boldsymbol{\theta}}(A_t | S_t) / B$
    end for
  end for
  Update policy network $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \hat{\nabla}$
end while

To implement, we need to estimate $v_{\pi_{\boldsymbol{\theta}}}(S_t)$ for all trajectories in mini-batch

Estimating Values: *Monte-Carlo*

Say we are looking into one trajectory τ

$$\tau : S_0, A_0 \xrightarrow{R_1} S_1, A_1 \xrightarrow{R_2} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T$$

We know how to use this trajectory to compute value estimates: for each t

$$\hat{V}_t = \text{estimate of value for } S_t = G_t = \sum_{i=t}^{T} \gamma^i R_{i+1}$$

If we the same state happens multiple times in the trajectory: we count the number of times $S_t = S$ *appears in the trajectory and average estimates, i.e.,*

$$\hat{v}_{\pi_{\boldsymbol{\theta}}}(S) = \frac{1}{\mathcal{N}(S \in \tau)} \sum_{t=0}^{T-1} \mathbf{1}\{S_t = S\} \hat{V}_t$$

where $\mathcal{N}(S \in \tau)$ *is the number of times* S *has appeared in* τ

Estimating Values: *Monte-Carlo*

If we have a mini-batch $\mathbb{B}$ of trajectories

$$\tau : S_0, A_0 \xrightarrow{R_1} S_1, A_1 \xrightarrow{R_2} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T$$

We use the same approach

$$\hat{V}_t[\tau] = G_t[\tau] = \sum_{i=t}^{T} \gamma^i R_{i+1}[\tau]$$

We count the number of times $S_t = S$ appears in all trajectories and average the sample estimates, i.e.,

$$\hat{v}_{\pi_{\boldsymbol{\theta}}}(S) = \frac{1}{\mathcal{N}(S \in \mathbb{B})} \sum_{\tau \in \mathbb{B}} \sum_{t=0}^{T-1} \mathbf{1}\{S_t[\tau] = S\} \hat{V}_t[\tau]$$

where $\mathcal{N}(S \in \mathbb{B})$ the number of times S has appeared in $\mathbb{B}$

Advantage PGM: *With Value Estimates*

```
EstAdvantagePGM():
```

Initiate with $\boldsymbol{\theta}$ *and learning rate* α
while *interacting* **do**
  for *mini-batch* $b = 1 : B$ **do**
    Sample $S_0, A_0 \xrightarrow{R_1} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T$ *with policy* $\pi_{\boldsymbol{\theta}}$
  end for
  Estimate value of all observed states in the mini-batch as $\hat{v}_{\pi_{\boldsymbol{\theta}}}(S_t)$
  Set $\hat{\nabla} = \mathbf{0}$
  for $b = 1 : B$ **do**
    for $t = 0 : T-1$ **do**
      Compute sample advantages $U_t = R_{t+1} + \gamma \hat{v}_{\pi_{\boldsymbol{\theta}}}(S_{t+1}) - \hat{v}_{\pi_{\boldsymbol{\theta}}}(S_t)$
      Update sample gradient $\hat{\nabla} \leftarrow \hat{\nabla} + U_t \nabla \log \pi_{\boldsymbol{\theta}}(A_t | S_t) / B$
    end for
  end for
  Update policy network $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \hat{\nabla}$
end while

Advantage PGM: *With Value Estimates*

We could guess that this algorithm is not going to perform very impressive!

+ *And why is that?!*
– *For the exact same reasons we said at the beginning of Chapter 4*
  ↳ *We have lots of states*
  ↳ *Many of them are rarely observed in a small mini-batch*
  ↳ *The estimates can hence be very high variance*
  ↳ *Also we need to wait for the whole mini-batch to be ready*
  ↳ . . .
+ *So, what is the solution?*
– *You tell me!*
+ *We go for function approximation via value networks!*
– *You got it right!*

Recall: *Value Network*

Let's keep our trajectories here

$$\tau : S_0, A_0 \xrightarrow{R_1} S_1, A_1 \xrightarrow{R_2} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T$$

What we need is a simple v-network, as we only need the state values

$$\mathbf{x}(s) \longrightarrow \boxed{v_{\mathbf{w}}(\cdot)} \longrightarrow \hat{v}_\pi(s)$$

In Chapter 4, we saw that we could train it via sample returns, i.e.,

$$\textit{Dataset} = \left\{ \left(S_t[\tau], \hat{V}_t[\tau] \right) : \forall t \text{ and } \tau \right\}$$

and we train the network by minimizing the least-square loss

Value Network: *Training*

Let's keep our trajectories here

$$\tau : S_0, A_0 \xrightarrow{R_1} S_1, A_1 \xrightarrow{R_2} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T$$

This means that we compute the loss function

$$\mathcal{L}^v(\mathbf{w}) = \sum_{\tau} \sum_{t} \left(v_{\mathbf{w}}(S_t[\tau]) - \hat{V}_t[\tau] \right)^2$$

and update the wights of the v-network as

$$\mathbf{w} \leftarrow \operatorname*{argmin}_{\mathbf{w}} \mathcal{L}^v(\mathbf{w})$$

which we approximately solve using gradient descent

Basic Actor-Critic

This is going to end us with a basic actor-critic algorithm:

```
AC_v1():
```

Initiate with $\boldsymbol{\theta}$ *and* $\mathbf{w}$, *as well as a learning rate* α

while *interacting* **do**

  for *mini-batch* $b = 1 : B$ **do**

    Sample $S_0, A_0 \xrightarrow{R_1} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T$ *with policy* $\pi_{\boldsymbol{\theta}}$

    for $t = 0 : T - 1$ **do**

      Compute value estimate $\hat{V}_t$

      Compute sample advantages $U_t = R_{t+1} + \gamma v_{\mathbf{w}}(S_{t+1}) - v_{\mathbf{w}}(S_{t+1})$

      Update sample gradient $\hat{\nabla} \leftarrow \hat{\nabla} + U_t \nabla \log \pi_{\boldsymbol{\theta}}(A_t | S_t) / B$

    end for

  end for

  Update policy network $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \hat{\nabla}$

  Update $\mathbf{w}$ *by SGD using value estimates* $\hat{V}_t$

end while

Training Value Network: *TD Estimates*

$$\tau : S_0, A_0 \xrightarrow{R_1} S_1, A_1 \xrightarrow{R_2} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T$$

But now that we have a value network, we could also use TD: at step t, we set

$$\hat{V}_t = \text{estimate of value for } S_t = R_{t+1} + \gamma v_{\mathbf{w}} \left(S_{t+1} \right)$$

- *We can estimate the advantage using the current value network*

$$U_t = R_{t+1} + \gamma v_{\mathbf{w}} \left(S_{t+1} \right) - v_{\mathbf{w}} \left(S_{t+1} \right)$$

- *We then use least-squares update value network by TD sample estimates*

$$\mathcal{L}^v \left(\mathbf{w} \right) = \sum_{\tau} \sum_{t=0}^{T-1} \left(v_{\mathbf{w}} \left(S_t \left[\tau \right] \right) - \hat{V}_t \left[\tau \right] \right)^2$$

Training Value Network: *TD Estimates*

Let's write the update rule of the value network: we compute the gradient of loss and move in that direction

$$\nabla \mathcal{L}^v(\mathbf{w}) = 2 \sum_{t=0}^{T-1} \underbrace{\left(v_{\mathbf{w}}(S_t) - \hat{V}_t \right)}_{-\Delta_t} \nabla v_{\mathbf{w}}(S_t)$$

We used to call Δ_t the TD error, and set the learning rate to some $\beta/2$ to get

$$\mathbf{w} \leftarrow \mathbf{w} + \beta \sum_{t=0}^{T-1} \Delta_t \nabla v_{\mathbf{w}}(S_t)$$

Training Value Network: *TD Estimates*

Let's look at TD error: recall that our labels, i.e., sample estimates of values, are

$$\hat{V}_t = R_{t+1} + \gamma v_{\mathbf{w}} \left(S_{t+1} \right)$$

So the TD error is given by

$$\begin{aligned} \Delta_t &= \hat{V}_t - v_{\mathbf{w}} \left(S_t \right) \\ &= R_{t+1} + \gamma v_{\mathbf{w}} \left(S_{t+1} \right) - v_{\mathbf{w}} \left(S_t \right) \\ &= U_t \end{aligned}$$

This leads us to what we observed in Chapter 5

Recall: Advantage vs TD Error

Advantage is an estimator of TD error

A2C: *Basic Version*

A2C():

Initiate with $\boldsymbol{\theta}$ *and* $\mathbf{w}$, *as well as learning rates* α *and* β
while *interacting* **do**
    Start with zero gradients $\hat{\nabla}_{\mathbf{w}} = \hat{\nabla}_{\boldsymbol{\theta}} = \mathbf{0}$
    Sample $S_0, A_0 \xrightarrow{R_1} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T$ *with policy* $\pi_{\boldsymbol{\theta}}$
    Compute sample advantages $U_t = R_{t+1} + \gamma v_{\mathbf{w}}(S_{t+1}) - v_{\mathbf{w}}(S_t)$
    for $t = 0 : T-1$ **do**
        Compute sample policy gradient $\hat{\nabla}_{\boldsymbol{\theta}} \leftarrow \hat{\nabla}_{\boldsymbol{\theta}} + U_t \nabla \log \pi_{\boldsymbol{\theta}}(A_t | S_t)$
        Compute sample value gradient $\hat{\nabla}_{\mathbf{w}} \leftarrow \hat{\nabla}_{\mathbf{w}} + U_t \nabla v_{\mathbf{w}}(S_t)$
    end for
    Update policy network $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \hat{\nabla}_{\boldsymbol{\theta}}$
    Update value network $\mathbf{w} \leftarrow \mathbf{w} + \beta \hat{\nabla}_{\mathbf{w}}$
end while

This is the single-trajectory form of

Advantage Actor Critic $\equiv$ *A2C*

A2C: *Online Version*

Since we use TD, we can also update online, i.e., in each time step

```
OnlineA2C():
```

Initiate with $\boldsymbol{\theta}$ and $\mathbf{w}$, a random state S_0, $t = 0$ and learning rates α and β

while** interacting **do

  Sample A_t from $\pi_{\boldsymbol{\theta}}(\cdot|S_t)$

  Sample single step $S_t, A_t \xrightarrow{R_{t+1}} S_{t+1}$ from environment

  Compute sample advantage $U_t = R_{t+1} + \gamma v_{\mathbf{w}}(S_{t+1}) - v_{\mathbf{w}}(S_t)$

  Update policy network $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha U_t \nabla \log \pi_{\boldsymbol{\theta}}(A_t|S_t)$

  Update value network $\mathbf{w} \leftarrow \mathbf{w} + \beta U_t \nabla v_{\mathbf{w}}(S_t)$

  ***if** S_{t+1} is terminal **then** draw a random S_{t+1}*

  Set $t \leftarrow t + 1$

end while

But, that would be too noisy and hence quite unstable

A2C: *Mini-Batch Version*

We can further extend to mini-batch learning

`miniBatchA2C():`

Initiate with $\boldsymbol{\theta}$ *and* $\mathbf{w}$, *as well as learning rates* α *and* β

while *interacting* **do**

    Start with zero gradients $\hat{\nabla}_{\mathbf{w}} = \hat{\nabla}_{\boldsymbol{\theta}} = \mathbf{0}$

    for *mini-batch* $b = 1 : B$ **do**

        Sample $S_0, A_0 \xrightarrow{R_1} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T$ *with policy* $\pi_{\boldsymbol{\theta}}$

        Compute sample advantages $U_t = R_{t+1} + \gamma v_{\mathbf{w}}(S_{t+1}) - v_{\mathbf{w}}(S_t)$

        for $t = 0 : T-1$ **do**

            Compute sample policy gradient $\hat{\nabla}_{\boldsymbol{\theta}} \leftarrow \hat{\nabla}_{\boldsymbol{\theta}} + U_t \nabla \log \pi_{\boldsymbol{\theta}}(A_t | S_t)$

            Compute sample value gradient $\hat{\nabla}_{\mathbf{w}} \leftarrow \hat{\nabla}_{\mathbf{w}} + U_t \nabla v_{\mathbf{w}}(S_t)$

        end for

    end for

    Update policy network $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \hat{\nabla}_{\boldsymbol{\theta}}$

    Update value network $\mathbf{w} \leftarrow \mathbf{w} + \beta \hat{\nabla}_{\mathbf{w}}$

end while

Actor-Critic via Shared-Network

There is one extra obvious fact: *the policy and values that we learn are very much mutually related!*

+ *So, why don't we learn them together?!*
– *Actually we can!*

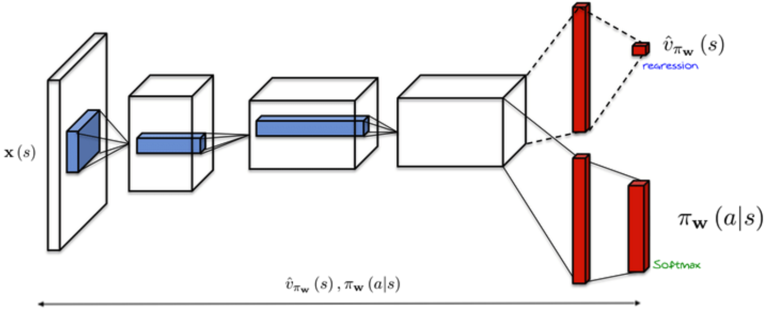
We can consider an actor-critic model, i.e.,

$$\mathbf{x}\left(s,a\right) \longrightarrow \boxed{\left(\pi,v\right)_{\mathbf{w}}\left(\cdot\right)} \longrightarrow \hat{v}_{\pi_{\mathbf{w}}}\left(s\right), \pi_{\mathbf{w}}\left(a|s\right)$$

and train it all together!

- *This model can be simply a DNN*
- *The DNN's output contains both policy and value*

Actor-Critic via Shared-Network: *Visualization*



Here, value and policy share same layers except the few last layers

Actor-Critic via Shared-Network: *Loss*

+ *But how can we train the loss in this network?*
– *We could let it to be proportional to sum of our both objectives*

We could define a new loss as

$$\begin{aligned}\mathcal{L}(\mathbf{w}) &= -\mathcal{J}(\pi_{\mathbf{w}}) + \xi \mathcal{L}^{v}(\mathbf{w}) \\ &= \sum_{\tau} \sum_{t=0}^{T-1} -U_t[\tau] \log \pi_{\mathbf{w}}(A_t[\tau] | S_t[\tau]) + \xi \left(v_{\mathbf{w}}(S_t[\tau]) - \hat{V}_t[\tau] \right)^2\end{aligned}$$

for some hyperparameter ξ: it's easy to see that in this case

$$\nabla \mathcal{L}(\mathbf{w}) = -\sum_{\tau} \sum_{t=0}^{T-1} U_t[\tau] \left[\nabla \log \pi_{\mathbf{w}}(A_t[\tau] | S_t[\tau]) + \xi \nabla v_{\mathbf{w}}(S_t[\tau]) \right]$$

A2C: *Shared-Network Version*

```
sharedNetA2C():
```

1: *Initiate shared network* $(\pi_{\mathbf{w}}, v_{\mathbf{w}})$ *with* $\mathbf{w}$
2: *Choose potentially scheduled value-weight* ξ *and learning rate* α
3: ***while*** *interacting* ***do***
4: *Start with zero gradients* $\hat{\nabla}_{\mathbf{w}} = \mathbf{0}$
5: ***for*** *mini-batch* $b = 1 : B$ ***do***
6: *Sample* $S_0, A_0 \xrightarrow{R_1} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T$ *with policy* $\pi_{\mathbf{w}}$
7: *Compute sample advantages* $U_t = R_{t+1} + \gamma v_{\mathbf{w}}(S_{t+1}) - v_{\mathbf{w}}(S_t)$
8: ***for*** $t = 0 : T - 1$ ***do***
9: *Compute sample gradient* $\hat{\nabla} \leftarrow \hat{\nabla} + U_t \left[\nabla \log \pi_{\mathbf{w}}(A_t | S_t) + \xi \nabla v_{\mathbf{w}}(S_t)\right]$
10: ***end for***
11: ***end for***
12: *Update shared network* $\mathbf{w} \leftarrow \mathbf{w} + \alpha \hat{\nabla}$
13: ***end while***