Reinforcement Learning

Chapter 5: RL via Policy Gradient

Ali Bereyhi

ali.bereyhi@utoronto.ca

Department of Electrical and Computer Engineering

University of Toronto

Fall 2025

Natural Policy Gradient: *Main Challenges*

$$\boxed{\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \sqrt{\frac{2d_{\max}}{\nabla_k^\mathsf{T} \mathbf{H}_k^{-1} \nabla_k}} \mathbf{H}_k^{-1} \nabla_k}$$

If we update with this rule: *we could see*

1. *the new point $\boldsymbol{\theta}_{k+1}$ does not fulfill what we expect, i.e.,*
   - ↳ *it might do no improvement*

     $$\mathcal{J}\left(\pi_{\boldsymbol{\theta}_{k+1}}\right) \leqslant \mathcal{J}\left(\pi_{\boldsymbol{\theta}_k}\right)$$

   - ↳ *it might violate the constraint*

     $$\bar{D}_{\mathrm{KL}}\left(\pi_{\boldsymbol{\theta}_{k+1}} \| \pi_{\boldsymbol{\theta}_k}\right) > d_{\max}$$

\+ *But, didn't we solve the optimization problem?!*

– *Well! We did it approximately not exactly*

Natural Policy Gradient: *Main Challenges*

$$\boxed{\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \sqrt{\frac{2d_{\max}}{\nabla_k^\mathsf{T} \mathbf{H}_k^{-1} \nabla_k}} \mathbf{H}_k^{-1} \nabla_k}$$

If we update with this rule: *we need to*

2. *compute* Hessian *of* $\bar{D}_{\mathrm{KL}}\left(\pi_{\boldsymbol{\theta}} \| \pi_{\boldsymbol{\theta}_k}\right)$

   ↳ *we need to compute all second order derivatives, i.e., for all choices of* i *and* j

$$\frac{\partial^2}{\partial \theta_i \partial \theta_j} \bar{D}_{\mathrm{KL}}\left(\pi_{\boldsymbol{\theta}} \| \pi_{\boldsymbol{\theta}_k}\right)$$

   ↳ *say we use ResNet-50 with* 2.6×10^7 *trainable parameters*

      ↳ *we need to compute about* 6.6×10^{14} *derivatives*

      ↳ *this in principle changes the complexity in orders of magnitude*

Natural Policy Gradient: *Main Challenges*

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \sqrt{\frac{2d_{\max}}{\nabla_k^{\mathsf{T}} \mathbf{H}_k^{-1} \nabla_k}} \mathbf{H}_k^{-1} \nabla_k$$

Say we computed the Hessian: *we need to*

3. *invert the Hessian of* $\mathbf{H}_k \in \mathbb{R}^{D \times D}$
   - ↳ *the complexity scales as* $\mathcal{O}\left(D^{\xi}\right)$
     - ↳ $\xi = 3$ *for classical Gauss-Jordan algorithm*
     - ↳ *it can reduces to* $\xi \approx 2.4$ *if we use more optimized algorithms like CW*
   - ↳ *at the end, this is computationally very expensive*

TRPO: *Backtracking Line*

The first algorithmic approach proposed by Schulman et. al was

$$\textit{Trust Region Policy Optimization} \equiv \textit{TRPO}$$

It uses two simple ideas to overcome the mentioned issues

- *Backtracking line challenge to get rid of the first issue*
- *Conjugate gradient to overcome the other two*

Let's take a look

TRPO: *Backtracking Line*

$$\boxed{\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \sqrt{\frac{2d_{\max}}{\nabla_k^\mathsf{T} \mathbf{H}_k^{-1} \nabla_k}} \mathbf{H}_k^{-1} \nabla_k}$$

Through analysis it turns out that: the direction of natural policy gradient is effective; however, the step size might be overshooting

+ *Why don't we scale it back?*
- *Sure! We can do this efficiently via backtracking line*

TRPO: *Backtracking Line*

`BacktrackLine():`

1: *Choose some* $\alpha < 1$, *set* $i = 0$ *and start with some* $\delta > d_{\max}$

2: ***while*** $\delta > d_{\max}$ ***do***

3: *Replace* $\boldsymbol{\theta}_{k+1}$ *with*

$$\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k + \alpha^i \sqrt{\frac{2d_{\max}}{\nabla_k^\mathsf{T} \mathbf{H}_k^{-1} \nabla_k}} \mathbf{H}_k^{-1} \nabla_k$$

4: *Set* $\delta \leftarrow \bar{D}_{\mathrm{KL}} \left(\pi_{\boldsymbol{\theta}_{k+1}} \| \pi_{\boldsymbol{\theta}_k} \right)$

5: *Update* $i \leftarrow i + 1$

6: ***end while***

+ *But we are only checking the constraint?!*

– It turns out that *this could also guarantee policy improvement*

TRPO: *Conjugate Gradient*

The next trick in TRPO is to write down the update in a form that can be computed via conjugate gradient: let's take a look at the update rule

$$\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k + \alpha^i \sqrt{\frac{2d_{\max}}{\nabla_k^\mathsf{T} \mathbf{H}_k^{-1} \nabla_k}} \mathbf{H}_k^{-1} \nabla_k$$

We can define the vector

$$\mathbf{y}_k = \mathbf{H}_k^{-1} \nabla_k$$

It is then easy to say that

$$\begin{aligned} \nabla_k^\mathsf{T} \mathbf{H}_k^{-1} \nabla_k = \nabla_k^\mathsf{T} \mathbf{H}_k^{-1} \mathbf{I} \nabla_k &= \nabla_k^\mathsf{T} \mathbf{H}_k^{-1} \underbrace{\mathbf{H}_k \mathbf{H}_k^{-1}}_{\mathbf{I}} \nabla_k \\ &= \underbrace{\nabla_k^\mathsf{T} \mathbf{H}_k^{-1}}_{\mathbf{y}_k^\mathsf{T}} \mathbf{H}_k \underbrace{\mathbf{H}_k^{-1} \nabla_k}_{\mathbf{y}_k} = \mathbf{y}_k^\mathsf{T} \mathbf{H}_k \mathbf{y}_k \end{aligned}$$

TRPO: *Conjugate Gradient*

If we have $\mathbf{y}_k$, *we could update more easily*

$$\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k + \alpha^i \sqrt{\frac{2d_{\max}}{\mathbf{y}_k^\top \mathbf{H}_k \mathbf{y}_k}} \mathbf{y}_k$$

Let's see if there is any efficient way to find $\mathbf{y}_k$ *at least approximately*

$$\mathbf{y}_k = \mathbf{H}_k^{-1} \nabla_k \rightsquigarrow \mathbf{H}_k \mathbf{y}_k = \nabla_k$$

Now, let's define $\mathbf{g}(\boldsymbol{\theta}) = \nabla \mathcal{L}_k(\boldsymbol{\theta})$: *obviously, we have*

$$\begin{aligned} \nabla_k &= \mathbf{g}(\boldsymbol{\theta}_k) \\ \mathbf{H}_k &= \nabla \mathbf{g}(\boldsymbol{\theta}) \big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_k} \end{aligned}$$

TRPO: *Conjugate Gradient*

Let's use these facts to expand our equation

$$\begin{aligned} \mathbf{y}_k = \mathbf{H}_k^{-1} \nabla_k \rightsquigarrow \mathbf{H}_k \mathbf{y}_k &= \nabla_k \\ \nabla \mathbf{g}\left(\boldsymbol{\theta}_k\right) \mathbf{y}_k &= \mathbf{g}\left(\boldsymbol{\theta}_k\right) \\ \nabla\left(\mathbf{g}\left(\boldsymbol{\theta}\right) \mathbf{y}_k\right)\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_k} &= \mathbf{g}\left(\boldsymbol{\theta}_k\right) \end{aligned}$$

The above functional equation can be solved for $\mathbf{y}_k$ via conjugate gradient algorithm[1], even without knowing the complete $\mathbf{H}_k = \nabla \mathbf{g}\left(\boldsymbol{\theta}_k\right)$!

In practice, we do the following

- *Compute the gradient estimator $\hat{\nabla}_k$*
- *Compute a sample Hessian $\hat{\mathbf{H}}_k$*
- *Solve $\hat{\mathbf{H}}_k \mathbf{y}_k = \hat{\nabla}_k$ via conjugate gradient*

[1] *You could check this tutorial if you are interested to know more about that*

TRPO: *Comments on Estimating Hessian*

As long as we need only an estimate, we can estimate Hessian by sampling: if we extend our derivative in Assignment 3, we will see

$$
\begin{aligned}
\mathbf{H}_k &= \nabla^2 \bar{D}_{\mathrm{KL}} \left(\pi_{\boldsymbol{\theta}} \| \pi_{\boldsymbol{\theta}_k} \right) |_{\boldsymbol{\theta}_k} \\
&= \int_s \int_a d_{\boldsymbol{\theta}_k}(s) \nabla \pi_{\boldsymbol{\theta}_k}(a|s) \nabla \log \pi_{\boldsymbol{\theta}_k}(a|s)^{\mathsf{T}} \\
&= \int_s \int_a \underbrace{d_{\boldsymbol{\theta}_k}(s) \pi_{\boldsymbol{\theta}_k}(a|s)}_{\text{distribution}} \underbrace{\nabla \log \pi_{\boldsymbol{\theta}_k}(a|s) \nabla \log \pi_{\boldsymbol{\theta}_k}(a|s)^{\mathsf{T}}}_{\text{sample outer product}} \\
&= \mathbb{E}_{\pi_{\boldsymbol{\theta}_k}} \left\{ \nabla \log \pi_{\boldsymbol{\theta}_k}(A|S) \nabla \log \pi_{\boldsymbol{\theta}_k}(A|S)^{\mathsf{T}} \right\}
\end{aligned}
$$

This is the Fisher information matrix and can be estimated by sampling

TRPO

`TRPO():`

Initiate with $\boldsymbol{\theta}$ and dampening factor $\alpha < 1$
while** interacting **do
    for** mini-batch $b = 1 : B$ **do
        Sample $S_0, A_0 \xrightarrow{R_1} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T$ by policy $\pi_{\boldsymbol{\theta}}$
        *Compute sample $U_t = R_{t+1} + \gamma v_{\pi_{\boldsymbol{\theta}}}(S_{t+1}) - v_{\pi_{\boldsymbol{\theta}}}(S_t)$ **for** $t = 0 : T-1$*
        Compute sample gradient as $\hat{\nabla}_b = \sum_t U_t \nabla \log \pi_{\boldsymbol{\theta}}(A_t | S_t)$
    end for
    Compute estimator as $\hat{\nabla} = \mathtt{mean}(\hat{\nabla}_1, \ldots, \hat{\nabla}_B)$ and a Hessian estimator $\hat{\mathbf{H}}$
    Solve $\hat{\mathbf{H}}\mathbf{y} = \hat{\nabla}$ for $\mathbf{y}$ via conjugate gradient with multiple iterations
    Backtrack on a line: find minimum integer i such that

$$\boldsymbol{\theta}' \leftarrow \boldsymbol{\theta} + \alpha^i \sqrt{\frac{2 d_{\max}}{\mathbf{y}^\mathsf{T} \hat{\mathbf{H}} \mathbf{y}}}\, \mathbf{y}$$

    satisfies $\bar{D}_{\mathrm{KL}}(\pi_{\boldsymbol{\theta}'} \| \pi_{\boldsymbol{\theta}}) \leqslant d_{\max}$
    Update $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}'$
end while

Back to Trust Region PGM

`AdvantageGD():`

1: *Start with some initial* $\boldsymbol{\theta}_0$
2: ***for*** $k = 1 : K$ ***do***
3: *Compute the surrogate function* $\mathcal{L}_k(\pi_{\boldsymbol{\theta}})$
4: *Update the parameters as*

$$\boldsymbol{\theta}_{k+1} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \mathcal{L}_k(\pi_{\boldsymbol{\theta}}) \quad \textit{subject to} \quad \pi_{\boldsymbol{\theta}} \textit{ and } \pi_{\boldsymbol{\theta}_k} \textit{ are close}$$

5: ***end for***

+ *Was this whole "closeness" metric worth it?*
– *Well! Maybe not!*

Back to Trust Region PGM: *Alternative Formulation*

Let's check back what was our concern: we wanted to maximize

$$\mathcal{L}_k\left(\pi_{\boldsymbol{\theta}}\right) = \mathbb{E}_{\pi_{\boldsymbol{\theta}_k}} \left\{ u_{\pi_{\boldsymbol{\theta}_k}}\left(S, A\right) \frac{\pi_{\boldsymbol{\theta}}\left(A|S\right)}{\pi_{\boldsymbol{\theta}_k}\left(A|S\right)} \right\}$$

while making sure that

$$\mathbb{Var}\left\{ \hat{\mathcal{L}}_k\left(\pi_{\boldsymbol{\theta}}\right) \right\} \propto \frac{\pi_{\boldsymbol{\theta}}\left(A|S\right)}{\pi_{\boldsymbol{\theta}_k}\left(A|S\right)}$$

does not explode!

+ *Why don't we check the ratio of policies for "closeness"?*
- *Sounds like a good idea!*

Trust Region PGM: *Ratio-Limited Policy Optimization*

Let's assume $\mathcal{C}(\cdot)$ is a function that limits its argument into a restricted interval of variation: then we can define

$$\tilde{\mathcal{L}}_k(\pi_{\boldsymbol{\theta}}) = \mathbb{E}_{\pi_{\boldsymbol{\theta}_k}} \left\{ u_{\pi_{\boldsymbol{\theta}_k}}(S, A)\, \mathcal{C}\left(\frac{\pi_{\boldsymbol{\theta}}(A|S)}{\pi_{\boldsymbol{\theta}_k}(A|S)} \right) \right\}$$

If we optimize this surrogate function, we proximally satisfy what we want

`LimitedRatioAdvantageGD():`

1: *Start with some initial* $\boldsymbol{\theta}_0$
2: **for** $k = 1 : K$ **do**
3: *Compute the surrogate function* $\mathcal{L}_k(\pi_{\boldsymbol{\theta}})$
4: *Update the parameters as*

$$\boldsymbol{\theta}_{k+1} = \operatorname*{argmax}_{\boldsymbol{\theta}} \tilde{\mathcal{L}}_k(\pi_{\boldsymbol{\theta}})$$

5: ***end for***

Proximal Policy Optimization

A common form of this approach is used in

$$\textit{Proximal Policy Optimization} \equiv \textit{PPO}$$

In this algorithm, we set

$$\tilde{\mathcal{L}}_k\left(\pi_{\boldsymbol{\theta}}\right) = \mathbb{E}_{\pi_{\boldsymbol{\theta}_k}}\left\{\mathcal{L}_k^{\mathrm{Clip}}\left(S, A, \boldsymbol{\theta}\right)\right\}$$

where $\mathcal{L}_k^{\mathrm{Clip}}\left(S, A, \boldsymbol{\theta}\right)$ *is importance sample of advantage with clipped ratio,i.e.,*

$$\mathcal{L}_k^{\mathrm{Clip}}\left(S, A, \boldsymbol{\theta}\right) = \min\left\{u_{\pi_{\boldsymbol{\theta}_k}}\left(S, A\right)\frac{\pi_{\boldsymbol{\theta}}\left(A|S\right)}{\pi_{\boldsymbol{\theta}_k}\left(A|S\right)}, \ell_\varepsilon\left(u_{\pi_{\boldsymbol{\theta}_k}}\left(S, A\right)\right)\right\}$$

for the clipping function

$$\ell_\varepsilon\left(x\right) = \begin{cases}\left(1+\varepsilon\right)x & x > 0\\ \left(1-\varepsilon\right)x & x \leqslant 0\end{cases}$$

Proximal Policy Optimization

+ *This clipping looks quite complicated! How does it restrict the domain of variation?*
– *It is indeed complicated, but we may understand it by a simple example*

Say we have only one sample trajectory with single state S and action A: we hence estimate the restricted surrogate as

$$\tilde{\mathcal{L}}_k(\pi_{\boldsymbol{\theta}}) \approx \mathcal{L}_k^{\mathrm{Clip}}(S, A, \boldsymbol{\theta})$$

Now, say that this sample pair gives sample advantage $u_{\pi_{\boldsymbol{\theta}_k}}(S, A)$: this can be

- *a positive advantage*
- *a negative advantage*

Let's see output of our restricted surrogate in each case

Proximal Policy Optimization

+ *What happens when we have a positive sample advantage?*

– *In this case, we have*

$$\mathcal{L}_k^{\mathrm{Clip}}\left(S, A, \boldsymbol{\theta}\right) = u_{\pi_{\boldsymbol{\theta}_k}}\left(S, A\right) \min\left\{\frac{\pi_{\boldsymbol{\theta}}\left(A|S\right)}{\pi_{\boldsymbol{\theta}_k}\left(A|S\right)}, 1+\varepsilon\right\}$$

Since the advantage is positive, surrogate is optimized by $\boldsymbol{\theta}$ that increases the ratio: the clipping operator lets us do it only up to some $\boldsymbol{\theta}$ that

$$\frac{\pi_{\boldsymbol{\theta}}\left(A|S\right)}{\pi_{\boldsymbol{\theta}_k}\left(A|S\right)} \leqslant 1+\varepsilon$$

if the ratio happens to be more, it clips it by $1+\varepsilon$

Proximal Policy Optimization

+ *What happens when we have a negative sample advantage?*
- *In this case, we have*

$$\mathcal{L}_k^{\text{Clip}}\left(S, A, \boldsymbol{\theta}\right) = u_{\pi_{\boldsymbol{\theta}_k}}\left(S, A\right) \max\left\{\frac{\pi_{\boldsymbol{\theta}}\left(A|S\right)}{\pi_{\boldsymbol{\theta}_k}\left(A|S\right)}, 1-\varepsilon\right\}$$

Since the advantage is negative, surrogate is maximized by $\boldsymbol{\theta}$ that reduces the ratio: the clipping operator lets us do it only up to some $\boldsymbol{\theta}$ that

$$\frac{\pi_{\boldsymbol{\theta}}\left(A|S\right)}{\pi_{\boldsymbol{\theta}_k}\left(A|S\right)} \geqslant 1-\varepsilon$$

if the ratio happens to lie below, it clips it by $1-\varepsilon$

Proximal Policy Optimization

Moral of Story

Clipping will keep the maximizer of the restricted surrogate such that the new policy described by the maximizer of the surrogate has controlled variation as compared to $\pi_{\boldsymbol{\theta}_k}$. This controlled variation is tuned by ε

Doing so we are still keeping our new policy within a trust region; however,

- *We don't need to check KL-divergence*
- *We don't need to estimate Hessian*
- *We don't need to implement conjugate gradient algorithm*
- *We don't need backtracking line*

Or in a nutshell: the life becomes much easier ☺

PPO Algorithm

`PPO():`

Initiate with $\boldsymbol{\theta}$ *and learning* $\alpha < 1$

while *interacting* ***do***

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

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

    Compute sample $U_t = R_{t+1} + \gamma v_{\pi_{\boldsymbol{\theta}}}(S_{t+1}) - v_{\pi_{\boldsymbol{\theta}}}(S_t)$ ***for*** $t = 0 : T-1$

  end for

  Compute the restricted surrogate

$$\tilde{\mathcal{L}}(\pi_{\mathbf{x}}) = \texttt{mean}_b \left[\sum_{t=1}^{T} \min \left\{ U_t \frac{\pi_{\mathbf{x}}(A_t | S_t)}{\pi_{\boldsymbol{\theta}}(A_t | S_t)}, \ell_\varepsilon(U_t) \right\} \right]$$

  for $i = 1 : I$ *potentially* $I = 1$ ***do***

    Update $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \nabla \tilde{\mathcal{L}}(\pi_{\mathbf{x}}) |_{\mathbf{x} = \boldsymbol{\theta}}$

  end for

end while

Sample Reuse with TRPO and PPO

+ *Very nice! You did a great job; however, you did not mention anything about sample efficiency!*
  ↳ *With TRPO and PPO, we can make sure that our updated policy will be within the vicinity of previous policy*
  ↳ *But, we still sample a mini-batch, apply SGD and drop it!*
– *Well! As long as we are using TRPO and PPO, we can reuse our previous samples for some time! This can help us with sample efficiency*

In practice, we can use experience buffer here as well

- *We collect multiple sample trajectory and save them into into a buffer*
- *We treat the buffer as a dataset and break it into mini-batches*
- *We go multiple epochs over this dataset*

⋆ *We remove old trajectories periodically as our policy is getting far gradually*

Sample Reuse with TRPO and PPO

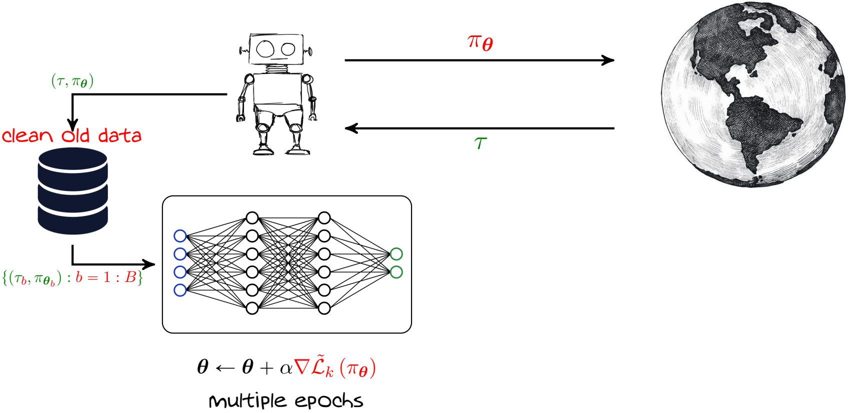
There is a tiny change that we need to consider in this case: when we compute the surrogate function, we should do the importance sampling with the policy that we sampled the trajectory with

For instance, say we sample B trajectories from the buffer

- *It might be that each trajectory has been sampled by one policy $\pi_{\boldsymbol{\theta}_b}$*
- ⚠ *They are all close policies as we clean buffer periodically*
- *If we use PPO, we could compute the surrogate as*

$$\tilde{\mathcal{L}}\left(\pi_{\mathbf{x}}\right) = \texttt{mean}_b \left[\sum_{t=1}^{T} \min \left\{ U_t \frac{\pi_{\mathbf{x}}\left(A_t | S_t\right)}{\pi_{\boldsymbol{\theta}_b}\left(A_t | S_t\right)}, \ell_\varepsilon\left(U_t\right) \right\} \right]$$

Sample Reuse with TRPO and PPO: *Visualization*



PPO Algorithm: *Sample Efficient Example*

`PPO():`

Initiate with $\boldsymbol{\theta}$, learning $\alpha < 1$, and an experience buffer with limited size
while** interacting **do
  Sample $\tau : S_0, A_0 \xrightarrow{R_1} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T$ by policy $\pi_{\boldsymbol{\theta}}$
  if** experience buffer is full **then
    Remove oldest sample
  end if
  Save sample $(\tau, \pi_{\boldsymbol{\theta}})$ into experience buffer as most recent
  for** $i = 1 : I$ potentially for multiple epochs of buffer **do
    Sample a mini-batch with B trajectories from experience buffer
    Compute the restricted surrogate

$$\tilde{\mathcal{L}}(\pi_{\mathbf{x}}) = \mathtt{mean}_b \left[\sum_{t=1}^{T} \min \left\{ U_t \frac{\pi_{\mathbf{x}}(A_t|S_t)}{\pi_{\boldsymbol{\theta}_b}(A_t|S_t)}, \ell_\varepsilon(U_t) \right\} \right]$$

    Update $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \nabla \tilde{\mathcal{L}}(\pi_{\mathbf{x}}) |_{\mathbf{x}=\boldsymbol{\theta}}$
  end for
end while

Sample Reuse with TRPO and PPO: *Final Notes*

Though we use experience reply as in DQL, we should note

- *In DQL, we are not very restricted with memory update*
  - ↳ *We could keep all samples and reuse them*
  - ↳ *This was because we could go totally off-policy with DQL*
- *In policy optimization, we are strictly restricted with memory update*
  - ↳ *We could not use very old samples efficiently*
    - ↳ *If we use them, we will have large variance*
  - ↳ *We can only mildly go off-policy*
    - ↳ *We keep a sample and squeeze the its most possible juice*

Important Conclusion

In terms of sample efficiency, we always have

$$\textit{Policy Gradient Methods} \ll \textit{DQL}$$

But they could become more stable than DQL as they directly control the policy

Last Stop: *Actor-Critic Approaches*

We are finished with PGMs

- ✓ *We know how to train efficiently a policy network*
  - ↳ *We can use TRPO and PPO pretty much in any problem*
- ✗ *But we assumed that we have access to the value function*
  - ↳ *This is not really the case in practice!*

We now go for the last chapter, where we learn to

approximate the value function via a value network

This will complete our box of tools and we are ready to solve any RL problem!

Efficient Implementation: *TorchRL*

TorchRL

In larger RL projects, you might find it easier to have access to some pre-implemented modules: TorchRL does that for you

- *It's a library implemented in PyTorch*
- *It contains lots of useful modules, e.g., to implement experience replay*
- *It does not give you implemented algorithms*
  - ↳ *Instead, it gives you modules that you need to implement the algorithm*
- *It's compatible with Gymnasium*

Since we often use PyTorch for DL implementations and Gymnasium to implement environment, TorchRL is a very efficient toolbox

Torch RL: *Sample Modules*

TorchRL

Some sample lines of code

```
from torchrl.collectors import SyncDataCollector
from torchrl.data.replay_buffers import ReplayBuffer
from torchrl.data.replay_buffers.samplers import SamplerWithoutReplacement
from torchrl.data.replay_buffers.storages import LazyTensorStorage
```

Some Resources

- *Take a look at the introductory presentation by Vincent Moens*
- *Go over its documentation at TorchRL page*