An Introduction to Reinforcement Learning

Part III: Approximation and Policy Gradient Methods

Christoph Belak

Technische Universität Berlin

Stochastic Numerics and Statistical Learning Workshop
KAUST, May 24, 2022

SARSA (On-Policy Temporal Difference Control)

- ▷ Initialize q arbitrarily
- ▷ Initialize S_0 (e.g. randomly)
- ▷ Choose action A_0 using ϵ-greedy policy derived from q
- ▷ Loop over $n = 0, 1, 2, \ldots$
  - ▷ Take action A_n and observe $r(S_n, A_n)$ and S_{n+1}
  - ▷ Choose action A_{n+1} using ϵ-greedy policy derived from q
  - ▷ Update q according to

$$q(S_n, A_n) \leftarrow q(S_n, A_n) + \alpha \Big[r(S_n, A_n) + \beta q(S_{n+1}, A_{n+1}) - q(S_n, A_n) \Big]$$

Q-Learning (Off-Policy Temporal Difference Control)

- ▷ Initialize q arbitrarily
- ▷ Initialize S_0 (e.g. randomly)
- ▷ Choose action A_0 using ϵ-greedy policy derived from q
- ▷ Loop over $n = 0, 1, 2, \ldots$
  - ▷ Take action A_n and observe $r(S_n, A_n)$ and S_{n+1}
  - ▷ Choose action A_{n+1} using ϵ-greedy policy derived from q
  - ▷ Update q according to

$$q(S_n, A_n) \leftarrow q(S_n, A_n) + \alpha \Big[r(S_n, A_n) + \beta \max_{a \in \mathbb{A}(s)} q(S_{n+1}, a) - q(S_n, A_n) \Big]$$

Approximation Methods

The algorithms we have seen so far fall into the class of **tabular methods**.

The algorithms we have seen so far fall into the class of **tabular methods**.

- They only work if both state and action space are finite

The algorithms we have seen so far fall into the class of **tabular methods**.

- They only work if both state and action space are finite
- They become inefficient if the number of state-action pairs grows large

The algorithms we have seen so far fall into the class of **tabular methods**.

- They only work if both state and action space are finite
- They become inefficient if the number of state-action pairs grows large

So what do we do?

The algorithms we have seen so far fall into the class of **tabular methods**.

- ▷ They only work if both state and action space are finite
- ▷ They become inefficient if the number of state-action pairs grows large

So what do we do?

- ▷ We **approximate** $\mathcal{V}$ or Q by parametric classes of functions v_θ or q_θ

The algorithms we have seen so far fall into the class of **tabular methods**.

- They only work if both state and action space are finite
- They become inefficient if the number of state-action pairs grows large

So what do we do?

- We **approximate** $\mathcal{V}$ or Q by parametric classes of functions v_θ or q_θ
- Linear methods: approximation using basis functions

The algorithms we have seen so far fall into the class of **tabular methods**.

- They only work if both state and action space are finite
- They become inefficient if the number of state-action pairs grows large

So what do we do?

- We **approximate** $\mathcal{V}$ or Q by parametric classes of functions v_θ or q_θ
- Linear methods: approximation using basis functions
- Nonlinear methods: approximation using e.g. neural networks

The algorithms we have seen so far fall into the class of **tabular methods**.

- They only work if both state and action space are finite
- They become inefficient if the number of state-action pairs grows large

So what do we do?

- We **approximate** $\mathcal{V}$ or Q by parametric classes of functions v_θ or q_θ
- Linear methods: approximation using basis functions
- Nonlinear methods: approximation using e.g. neural networks
- Instead of updating the estimate in each state, we update the parameters of the function class

The algorithms we have seen so far fall into the class of **tabular methods**.

- They only work if both state and action space are finite
- They become inefficient if the number of state-action pairs grows large

So what do we do?

- We **approximate** $\mathcal{V}$ or Q by parametric classes of functions v_θ or q_θ
- Linear methods: approximation using basis functions
- Nonlinear methods: approximation using e.g. neural networks
- Instead of updating the estimate in each state, we update the parameters of the function class
- This allows us to consider continuous state and action spaces (finally!)

The algorithms we have seen so far fall into the class of **tabular methods**.

- They only work if both state and action space are finite
- They become inefficient if the number of state-action pairs grows large

So what do we do?

- We **approximate** $\mathcal{V}$ or Q by parametric classes of functions v_θ or q_θ
- Linear methods: approximation using basis functions
- Nonlinear methods: approximation using e.g. neural networks
- Instead of updating the estimate in each state, we update the parameters of the function class
- This allows us to consider continuous state and action spaces (finally!)
- But makes it much harder to study convergence

We fix a parametric class of functions $v_\theta : \mathbb{S} \to \mathbb{R}, \theta \in \Theta$. Our aim is to approximate another function $V : \mathbb{S} \to \mathbb{R}$ in the sense of minimizing

$$\frac{1}{2} \int_{\mathbb{S}} |V(s) - v_\theta(s)|^2 \mu(\mathrm{d}s)$$

for some measure μ.

We fix a parametric class of functions $v_\theta : \mathbb{S} \to \mathbb{R}, \theta \in \Theta$. Our aim is to approximate another function $V : \mathbb{S} \to \mathbb{R}$ in the sense of minimizing

$$\frac{1}{2}\int_{\mathbb{S}} |V(s) - v_\theta(s)|^2 \mu(\mathrm{d}s)$$

for some measure μ. For example, we may consider $V \triangleq \mathcal{V}_\pi$ and μ the distribution of some random variable S. In this case, we therefore consider

$$\min_{\theta\in\Theta} \frac{1}{2}\mathbb{E}\Big[|\mathcal{V}_\pi(S) - v_\theta(S)|^2\Big].$$

We fix a parametric class of functions $v_\theta : \mathbb{S} \to \mathbb{R}, \theta \in \Theta$. Our aim is to approximate another function $V : \mathbb{S} \to \mathbb{R}$ in the sense of minimizing

$$\frac{1}{2}\int_{\mathbb{S}} |V(s) - v_\theta(s)|^2 \mu(\mathrm{d}s)$$

for some measure μ. For example, we may consider $V \triangleq \mathcal{V}_\pi$ and μ the distribution of some random variable S. In this case, we therefore consider

$$\min_{\theta\in\Theta} \frac{1}{2}\mathbb{E}\Big[|\mathcal{V}_\pi(S) - v_\theta(S)|^2\Big].$$

Gradient descent suggests that we solve this problem iteratively starting from a parameter configuration $\theta_0 \in \Theta$ and choosing

$$\theta_{t+1} = \theta_t - \frac{1}{2}\alpha\nabla\Big[\mathbb{E}\big[|\mathcal{V}_\pi(S) - v_\theta(S)|^2\big]\Big]$$

We fix a parametric class of functions $v_\theta : \mathbb{S} \to \mathbb{R}, \theta \in \Theta$. Our aim is to approximate another function $V : \mathbb{S} \to \mathbb{R}$ in the sense of minimizing

$$\frac{1}{2} \int_{\mathbb{S}} |V(s) - v_\theta(s)|^2 \mu(\mathrm{d}s)$$

for some measure μ. For example, we may consider $V \triangleq \mathcal{V}_\pi$ and μ the distribution of some random variable S. In this case, we therefore consider

$$\min_{\theta \in \Theta} \frac{1}{2} \mathbb{E}\Big[|\mathcal{V}_\pi(S) - v_\theta(S)|^2 \Big].$$

Gradient descent suggests that we solve this problem iteratively starting from a parameter configuration $\theta_0 \in \Theta$ and choosing

$$\begin{aligned} \theta_{t+1} &= \theta_t - \frac{1}{2} \alpha \nabla \Big[\mathbb{E}\big[|\mathcal{V}_\pi(S) - v_\theta(S)|^2 \big] \Big] \\ &= \theta_t + \alpha \mathbb{E}\Big[\big(\mathcal{V}_\pi(S) - v_\theta(S) \big) \nabla v_\theta(S) \Big] \end{aligned}$$

for all $t \in \mathbb{N}_0$, where ∇ denotes the gradient with respect to θ.

For the gradient descent step, we need to evaluate

$$\mathbb{E}\Big[\big(\mathcal{V}_\pi(S) - v_\theta(S)\big)\nabla v_\theta(S)\Big],$$

which can be (read: is) too costly to evaluate.

For the gradient descent step, we need to evaluate

$$\mathbb{E}\Big[\big(\mathcal{V}_\pi(S) - v_\theta(S)\big)\nabla v_\theta(S)\Big],$$

which can be (read: is) too costly to evaluate. In **stochastic gradient descent**, we therefore consider the update rule

$$\theta_{t+1} = \theta_t + \alpha\Big(\mathcal{V}_\pi(S) - v_\theta(S)\Big)\nabla v_\theta(S), \qquad t \in \mathbb{N}_0.$$

For the gradient descent step, we need to evaluate

$$\mathbb{E}\Big[\big(\mathcal{V}_\pi(S) - v_\theta(S)\big)\nabla v_\theta(S)\Big],$$

which can be (read: is) too costly to evaluate. In **stochastic gradient descent**, we therefore consider the update rule

$$\theta_{t+1} = \theta_t + \alpha\Big(\mathcal{V}_\pi(S) - v_\theta(S)\Big)\nabla v_\theta(S), \qquad t \in \mathbb{N}_0.$$

Intuition: In the mean, we move in the direction of steepest descent.

For the gradient descent step, we need to evaluate

$$\mathbb{E}\Big[\big(\mathcal{V}_\pi(S) - v_\theta(S)\big)\nabla v_\theta(S)\Big],$$

which can be (read: is) too costly to evaluate. In **stochastic gradient descent**, we therefore consider the update rule

$$\theta_{t+1} = \theta_t + \alpha\Big(\mathcal{V}_\pi(S) - v_\theta(S)\Big)\nabla v_\theta(S), \qquad t \in \mathbb{N}_0.$$

Intuition: In the mean, we move in the direction of steepest descent.

A critical issue for us is that we do not know $\mathcal{V}_\pi$ exactly. We could replace $\mathcal{V}_\pi(S)$ by an **unbiased estimate** such as

$$\sum_{n=0}^{\infty} \beta^n r(S_n^\pi, A_n^\pi) \qquad \text{with } S_0^\pi \sim S,$$

For the gradient descent step, we need to evaluate

$$\mathbb{E}\Big[\big(\mathcal{V}_\pi(S) - v_\theta(S)\big)\nabla v_\theta(S)\Big],$$

which can be (read: is) too costly to evaluate. In **stochastic gradient descent**, we therefore consider the update rule

$$\theta_{t+1} = \theta_t + \alpha\Big(\mathcal{V}_\pi(S) - v_\theta(S)\Big)\nabla v_\theta(S), \qquad t \in \mathbb{N}_0.$$

Intuition: In the mean, we move in the direction of steepest descent.

A critical issue for us is that we do not know $\mathcal{V}_\pi$ exactly. We could replace $\mathcal{V}_\pi(S)$ by an **unbiased estimate** such as

$$\sum_{n=0}^{\infty} \beta^n r(S_n^\pi, A_n^\pi) \qquad \text{with } S_0^\pi \sim S,$$

or we could use a **temporal difference target**, that is

$$r(S_0^\pi, A_0^\pi) + \beta v_\theta(S_1^\pi) \qquad \text{with } S_0^\pi \sim S.$$

Using the temporal difference target leads to what we call **semi-gradient** versions of SARSA and Q-Learning.

Using the temporal difference target leads to what we call **semi-gradient** versions of SARSA and Q-Learning.

Update rule for **Semi-Gradient SARSA**:

$$\theta \leftarrow \theta + \alpha \Big[r(S_n, A_n) + \beta q_\theta(S_{n+1}, A_{n+1}) - q_\theta(S_n, A_n) \Big] \nabla q_\theta(S_n, A_n)$$

Using the temporal difference target leads to what we call **semi-gradient** versions of SARSA and Q-Learning.

Update rule for **Semi-Gradient SARSA**:

$$\theta \leftarrow \theta + \alpha \Big[r(S_n, A_n) + \beta q_\theta(S_{n+1}, A_{n+1}) - q_\theta(S_n, A_n) \Big] \nabla q_\theta(S_n, A_n)$$

Update rule for **Semi-Gradient Q-Learning**:

$$\theta \leftarrow \theta + \alpha \Big[r(S_n, A_n) + \beta \max_{a \in \mathbb{A}(S_{n+1})} q_\theta(S_{n+1}, a) - q_\theta(S_n, A_n) \Big] \nabla q_\theta(S_n, A_n)$$

Now what about **convergence**?

Now what about **convergence**?

- Oh boy...

Now what about **convergence**?

- Oh boy...
- Semi-gradient SARSA, linear case: does not diverge, bounded error

Now what about **convergence**?

- Oh boy...
- Semi-gradient SARSA, linear case: does not diverge, bounded error
- Semi-gradient SARSA, nonlinear case: counterexample

Now what about **convergence**?

- Oh boy...
- Semi-gradient SARSA, linear case: does not diverge, bounded error
- Semi-gradient SARSA, nonlinear case: counterexample
- Semi-gradient Q-Learning: much more complicated, may diverge

Now what about **convergence**?

- Oh boy...
- Semi-gradient SARSA, linear case: does not diverge, bounded error
- Semi-gradient SARSA, nonlinear case: counterexample
- Semi-gradient Q-Learning: much more complicated, may diverge
- Apparently, on-policy methods often perform better than off-policy methods, but off-policy methods tend to find better policies

Now what about **convergence**?

- Oh boy...
- Semi-gradient SARSA, linear case: does not diverge, bounded error
- Semi-gradient SARSA, nonlinear case: counterexample
- Semi-gradient Q-Learning: much more complicated, may diverge
- Apparently, on-policy methods often perform better than off-policy methods, but off-policy methods tend to find better policies

The issue with Q-Learning:

Now what about **convergence**?

- Oh boy...
- Semi-gradient SARSA, linear case: does not diverge, bounded error
- Semi-gradient SARSA, nonlinear case: counterexample
- Semi-gradient Q-Learning: much more complicated, may diverge
- Apparently, on-policy methods often perform better than off-policy methods, but off-policy methods tend to find better policies

The issue with Q-Learning: the **deadly triad**

Now what about **convergence**?

- Oh boy...
- Semi-gradient SARSA, linear case: does not diverge, bounded error
- Semi-gradient SARSA, nonlinear case: counterexample
- Semi-gradient Q-Learning: much more complicated, may diverge
- Apparently, on-policy methods often perform better than off-policy methods, but off-policy methods tend to find better policies

The issue with Q-Learning: the **deadly triad**

- Function approximation

Now what about **convergence**?

- Oh boy...
- Semi-gradient SARSA, linear case: does not diverge, bounded error
- Semi-gradient SARSA, nonlinear case: counterexample
- Semi-gradient Q-Learning: much more complicated, may diverge
- Apparently, on-policy methods often perform better than off-policy methods, but off-policy methods tend to find better policies

The issue with Q-Learning: the **deadly triad**

- Function approximation
- Bootstrapping, i.e. using a biased target

Now what about **convergence**?

- Oh boy...
- Semi-gradient SARSA, linear case: does not diverge, bounded error
- Semi-gradient SARSA, nonlinear case: counterexample
- Semi-gradient Q-Learning: much more complicated, may diverge
- Apparently, on-policy methods often perform better than off-policy methods, but off-policy methods tend to find better policies

The issue with Q-Learning: the **deadly triad**

- Function approximation
- Bootstrapping, i.e. using a biased target
- Off-policy learning

Now what about **convergence**?

- Oh boy...
- Semi-gradient SARSA, linear case: does not diverge, bounded error
- Semi-gradient SARSA, nonlinear case: counterexample
- Semi-gradient Q-Learning: much more complicated, may diverge
- Apparently, on-policy methods often perform better than off-policy methods, but off-policy methods tend to find better policies

The issue with Q-Learning: the **deadly triad**

- Function approximation
- Bootstrapping, i.e. using a biased target
- Off-policy learning

There are a few approaches which try to overcome the triad: TD(λ), replay, Double Q-Learning, true-gradient RL methods, Emphatic-TD, ...

Policy Gradient Methods

Up to now: **Action-Value Methods**

Up to now: **Action-Value Methods**

- We learn the value of actions by estimating $Q_\pi(s, a) \approx q_\theta(s, a)$

Up to now: **Action-Value Methods**

- We learn the value of actions by estimating $Q_\pi(s, a) \approx q_\theta(s, a)$
- We select actions based on estimated action values via $\max_{a \in \mathbb{A}(s)} q_\theta(s, a)$

Up to now: **Action-Value Methods**

- We learn the value of actions by estimating $Q_\pi(s, a) \approx q_\theta(s, a)$
- We select actions based on estimated action values via $\max_{a \in \mathbb{A}(s)} q_\theta(s, a)$

Alternative: **Policy Gradient Methods**

Up to now: **Action-Value Methods**

- We learn the value of actions by estimating $Q_\pi(s, a) \approx q_\theta(s, a)$
- We select actions based on estimated action values via $\max_{a \in \mathbb{A}(s)} q_\theta(s, a)$

Alternative: **Policy Gradient Methods**

- Consider parametrized policy $\pi_\theta(a|s)$

Up to now: **Action-Value Methods**

- We learn the value of actions by estimating $Q_\pi(s, a) \approx q_\theta(s, a)$
- We select actions based on estimated action values via $\max_{a \in \mathbb{A}(s)} q_\theta(s, a)$

Alternative: **Policy Gradient Methods**

- Consider parametrized policy $\pi_\theta(a|s)$
- Try to learn the parameter θ for which π_θ approximates an optimal policy

Up to now: **Action-Value Methods**

- We learn the value of actions by estimating $Q_\pi(s, a) \approx q_\theta(s, a)$
- We select actions based on estimated action values via $\max_{a \in \mathbb{A}(s)} q_\theta(s, a)$

Alternative: **Policy Gradient Methods**

- Consider parametrized policy $\pi_\theta(a|s)$
- Try to learn the parameter θ for which π_θ approximates an optimal policy
- This could be done by some gradient ascend method

Up to now: **Action-Value Methods**

- We learn the value of actions by estimating $Q_\pi(s,a) \approx q_\theta(s,a)$
- We select actions based on estimated action values via $\max_{a\in\mathbb{A}(s)} q_\theta(s,a)$

Alternative: **Policy Gradient Methods**

- Consider parametrized policy $\pi_\theta(a|s)$
- Try to learn the parameter θ for which π_θ approximates an optimal policy
- This could be done by some gradient ascend method

We restrict to finite horizon problems

$$\mathcal{V}_\pi(s) \triangleq \mathbb{E}_s\left[\sum_{n=0}^{N} r(S_n^\pi, A_n^\pi)\right]$$

and keep the initial state s fixed.

Example: Parametrization for **finite action spaces**

$$\pi_\theta(a|s) \triangleq \frac{\exp\{h_\theta(s,a)\}}{\sum_{a' \in \mathbb{A}(s)} \exp\{h_\theta(s,a')\}},$$

where h_θ, $\theta \in \Theta$, is a parametrized function class.

Example: Parametrization for **finite action spaces**

$$\pi_\theta(a|s) \triangleq \frac{\exp\{h_\theta(s,a)\}}{\sum_{a' \in \mathbb{A}(s)} \exp\{h_\theta(s,a')\}},$$

where h_θ, $\theta \in \Theta$, is a parametrized function class.

Example: Parametrization for **continuous action spaces**

$$\pi_\theta(a|s) \triangleq \frac{1}{\sqrt{2\pi\sigma_\theta(s)^2}} \exp\Big\{-\frac{1}{2}\frac{(a-\mu_\theta(s))^2}{\sigma_\theta(s)^2}\Big\},$$

where $(\mu_\theta, \sigma_\theta)$, $\theta \in \Theta$, is a parametrized function class with $\sigma_\theta > 0$.

How do we find an optimal θ? E.g. by stochastic gradient ascend with respect to

$$J(\theta) \triangleq \mathcal{V}_{\pi_\theta}(s) = \mathbb{E}_s\Big[\sum_{n=0}^{N} r(S_n, A_n)\Big],$$

where $(S, A) \triangleq (S^{\pi_\theta}, A^{\pi_\theta})$. Thus, we need to compute $\nabla J(\theta)$.

How do we find an optimal θ? E.g. by stochastic gradient ascend with respect to

$$J(\theta) \triangleq \mathcal{V}_{\pi_\theta}(s) = \mathbb{E}_s\Big[\sum_{n=0}^{N} r(S_n, A_n)\Big],$$

where $(S, A) \triangleq (S^{\pi_\theta}, A^{\pi_\theta})$. Thus, we need to compute $\nabla J(\theta)$.

Policy Gradient Theorem

It holds that

$$\nabla J(\theta) = \mathbb{E}_s\Big[\sum_{n=0}^{N} \nabla \log\big(\pi_\theta(A_n|S_n)\big) \sum_{k=n}^{N} r(S_k, A_k)\Big].$$

The Policy Gradient Theorem suggests that the update rule is

$$\theta \leftarrow \theta + \alpha \sum_{n=0}^{N} \nabla \log\big(\pi_\theta(A_n|S_n)\big) \sum_{k=n}^{N} r(S_k, A_k).$$

The Policy Gradient Theorem suggests that the update rule is

$$\theta \leftarrow \theta + \alpha \sum_{n=0}^{N} \nabla \log\big(\pi_\theta(A_n|S_n)\big) \sum_{k=n}^{N} r(S_k, A_k).$$

We may in fact just split up the sum over n to obtain the update

$$\theta \leftarrow \theta + \alpha \nabla \log\big(\pi_\theta(A_n|S_n)\big) \sum_{k=n}^{N} r(S_k, A_k).$$

The Policy Gradient Theorem suggests that the update rule is

$$\theta \leftarrow \theta + \alpha \sum_{n=0}^{N} \nabla \log\big(\pi_\theta(A_n|S_n)\big) \sum_{k=n}^{N} r(S_k, A_k).$$

We may in fact just split up the sum over n to obtain the update

$$\theta \leftarrow \theta + \alpha \nabla \log\big(\pi_\theta(A_n|S_n)\big) \sum_{k=n}^{N} r(S_k, A_k).$$

This algorithm is referred to as **REINFORCE**.

The Policy Gradient Theorem suggests that the update rule is

$$\theta \leftarrow \theta + \alpha \sum_{n=0}^{N} \nabla \log\big(\pi_\theta(A_n|S_n)\big) \sum_{k=n}^{N} r(S_k, A_k).$$

We may in fact just split up the sum over n to obtain the update

$$\theta \leftarrow \theta + \alpha \nabla \log\big(\pi_\theta(A_n|S_n)\big) \sum_{k=n}^{N} r(S_k, A_k).$$

This algorithm is referred to as **REINFORCE**.

Remarks:

The Policy Gradient Theorem suggests that the update rule is

$$\theta \leftarrow \theta + \alpha \sum_{n=0}^{N} \nabla \log\big(\pi_\theta(A_n|S_n)\big) \sum_{k=n}^{N} r(S_k, A_k).$$

We may in fact just split up the sum over n to obtain the update

$$\theta \leftarrow \theta + \alpha \nabla \log\big(\pi_\theta(A_n|S_n)\big) \sum_{k=n}^{N} r(S_k, A_k).$$

This algorithm is referred to as **REINFORCE**.

Remarks:

- As a stochastic gradient ascent algorithm with unbiased target, it converges to a local maximum

The Policy Gradient Theorem suggests that the update rule is

$$\theta \leftarrow \theta + \alpha \sum_{n=0}^{N} \nabla \log\big(\pi_\theta(A_n|S_n)\big) \sum_{k=n}^{N} r(S_k, A_k).$$

We may in fact just split up the sum over n to obtain the update

$$\theta \leftarrow \theta + \alpha \nabla \log\big(\pi_\theta(A_n|S_n)\big) \sum_{k=n}^{N} r(S_k, A_k).$$

This algorithm is referred to as **REINFORCE**.

Remarks:

- As a stochastic gradient ascent algorithm with unbiased target, it converges to a local maximum
- Tends to have high variance, but versions with control variates are available, e.g. REINFORCE with baseline in which

$$\sum_{k=n}^{N} r(S_k, A_k) \qquad \text{is replaced by} \qquad \sum_{k=n}^{N} r(S_k, A_k) - v_\vartheta(S_n)$$

REINFORCE:

$$\theta \leftarrow \theta + \alpha \sum_{k=n}^{N} r(S_k, A_k) \nabla \log\big(\pi_\theta(A_n|S_n)\big)$$

REINFORCE:

$$\theta \leftarrow \theta + \alpha \sum_{k=n}^{N} r(S_k, A_k) \nabla \log\big(\pi_\theta(A_n|S_n)\big)$$

REINFORCE with baseline:

$$\theta \leftarrow \theta + \alpha_\theta \Big[\sum_{k=n}^{N} r(S_k, A_k) - v_\vartheta(S_n)\Big] \nabla \log\big(\pi_\theta(A_n|S_n)\big)$$

REINFORCE:

$$\theta \leftarrow \theta + \alpha \sum_{k=n}^{N} r(S_k, A_k) \nabla \log\big(\pi_\theta(A_n|S_n)\big)$$

REINFORCE with baseline:

$$\vartheta \leftarrow \vartheta + \alpha_\vartheta \Big[\sum_{k=n}^{N} r(S_k, A_k) - v_\vartheta(S_n) \Big] \nabla v_\vartheta(S_n)$$

$$\theta \leftarrow \theta + \alpha_\theta \Big[\sum_{k=n}^{N} r(S_k, A_k) - v_\vartheta(S_n) \Big] \nabla \log\big(\pi_\theta(A_n|S_n)\big)$$

REINFORCE:

$$\theta \leftarrow \theta + \alpha \sum_{k=n}^{N} r(S_k, A_k) \nabla \log\big(\pi_\theta(A_n|S_n)\big)$$

REINFORCE with baseline:

$$\vartheta \leftarrow \vartheta + \alpha_\vartheta \Big[\sum_{k=n}^{N} r(S_k, A_k) - v_\vartheta(S_n)\Big] \nabla v_\vartheta(S_n)$$

$$\theta \leftarrow \theta + \alpha_\theta \Big[\sum_{k=n}^{N} r(S_k, A_k) - v_\vartheta(S_n)\Big] \nabla \log\big(\pi_\theta(A_n|S_n)\big)$$

Both methods use an unbiased target involving all future rewards. If N is large, this might be inefficient. Dare we use a biased target one more time?

REINFORCE with baseline:

$$\vartheta \leftarrow \vartheta + \alpha_\vartheta \Big[\sum_{k=n}^{N} r(S_k, A_k) - v_\vartheta(S_n) \Big] \nabla v_\vartheta(S_n)$$

$$\theta \leftarrow \theta + \alpha_\theta \Big[\sum_{k=n}^{N} r(S_k, A_k) - v_\vartheta(S_n) \Big] \nabla \log\big(\pi_\theta(A_n|S_n)\big)$$

REINFORCE with baseline:

$$\vartheta \leftarrow \vartheta + \alpha_\vartheta \Big[\sum_{k=n}^{N} r(S_k, A_k) - v_\vartheta(S_n) \Big] \nabla v_\vartheta(S_n)$$

$$\theta \leftarrow \theta + \alpha_\theta \Big[\sum_{k=n}^{N} r(S_k, A_k) - v_\vartheta(S_n) \Big] \nabla \log\big(\pi_\theta(A_n|S_n)\big)$$

Actor-Critic:

$$\vartheta \leftarrow \vartheta + \alpha_\vartheta \Big[r(S_n, A_n) + v_\vartheta(S_{n+1}) - v_\vartheta(S_n) \Big] \nabla v_\vartheta(S_n)$$

$$\theta \leftarrow \theta + \alpha_\theta \Big[r(S_n, A_n) + v_\vartheta(S_{n+1}) - v_\vartheta(S_n) \Big] \nabla \log\big(\pi_\theta(A_n|S_n)\big)$$

Concluding remarks:

Concluding remarks:

- Being derived from stochastic gradient ascent, policy gradient methods tend to converge to local optima only

Concluding remarks:

- Being derived from stochastic gradient ascent, policy gradient methods tend to converge to local optima only
- REINFORCE has an unbiased target, but needs all future rewards

Concluding remarks:

- Being derived from stochastic gradient ascent, policy gradient methods tend to converge to local optima only
- REINFORCE has an unbiased target, but needs all future rewards
- Actor-Critic is more data efficient, but introduces a bias

Concluding remarks:

- Being derived from stochastic gradient ascent, policy gradient methods tend to converge to local optima only
- REINFORCE has an unbiased target, but needs all future rewards
- Actor-Critic is more data efficient, but introduces a bias
- Neither value-based methods nor policy gradient methods are universally superior

Concluding remarks:

- Being derived from stochastic gradient ascent, policy gradient methods tend to converge to local optima only
- REINFORCE has an unbiased target, but needs all future rewards
- Actor-Critic is more data efficient, but introduces a bias
- Neither value-based methods nor policy gradient methods are universally superior
- The choice depends e.g. on what is easier to approximate, the value function or the policy

Concluding remarks:

- Being derived from stochastic gradient ascent, policy gradient methods tend to converge to local optima only
- REINFORCE has an unbiased target, but needs all future rewards
- Actor-Critic is more data efficient, but introduces a bias
- Neither value-based methods nor policy gradient methods are universally superior
- The choice depends e.g. on what is easier to approximate, the value function or the policy
- We have now seen a couple of possibilities to make reinforcement learning algorithms feasible for larger state and action spaces

Concluding remarks:

- Being derived from stochastic gradient ascent, policy gradient methods tend to converge to local optima only
- REINFORCE has an unbiased target, but needs all future rewards
- Actor-Critic is more data efficient, but introduces a bias
- Neither value-based methods nor policy gradient methods are universally superior
- The choice depends e.g. on what is easier to approximate, the value function or the policy
- We have now seen a couple of possibilities to make reinforcement learning algorithms feasible for larger state and action spaces
- What we still do not know is how to obtain all the data for all the financial optimization problems we want to solve

Concluding remarks:

- Being derived from stochastic gradient ascent, policy gradient methods tend to converge to local optima only
- REINFORCE has an unbiased target, but needs all future rewards
- Actor-Critic is more data efficient, but introduces a bias
- Neither value-based methods nor policy gradient methods are universally superior
- The choice depends e.g. on what is easier to approximate, the value function or the policy
- We have now seen a couple of possibilities to make reinforcement learning algorithms feasible for larger state and action spaces
- What we still do not know is how to obtain all the data for all the financial optimization problems we want to solve
- Blanka, we need your help!