Introduction to Reinforcement Learning:

Urtzi Ayesta and Matthieu Jonckheere

CNRS-IRIT & Ikerbasque-Univ. Basque Country

Lecture 5

From tables to parametric approximations

Curse of dimensionality :
Memory problem: Huge state spaces (Go 10^{170}), continuous state space for many applications
But also we have to deal with an exploration problem: it becomes quickly impossible to estimate $q(s, a)$ for all pairs...

Alternative:
We hope for a better trade-off exploration/generalization by parametrizing the problem. First solution: parametrize the **value function**.

So we now look for a few (or many) parameters instead of the whole value function, hoping that it does generalize well...

Types of approximation

Find a good representation of the state space (central to the success of the RL solution)

- Linear combination of features
- Neural nets,
- Decision tree,
- Dictionary of functions (Fourier, wavelets,...)

The manifold assumption.
Hope with features is that a lower dimensional manifold represent well the important states where the problem "lives".

Basic example:

Linear combination.

Features: a representation (a smart function) of the state.

$$x(s) = (x_1(s), \ldots, x_d(s)).$$

Approximation:

$$\hat{V}(s, w) = x(s)^\top w = \sum x_i(s) w_i.$$

Theoretical point of view

- Theoretically, we need to fix a functional sub-space $\mathcal{F}$ where we shall work (for instance linear combination of features).
- Then, we want to find the best possible approximation $V \in \mathcal{F}$ of V^π or of V^*.
- Note that there is no reason a priori not to pay a price for approximating.
- The minimum price is

$$\min_{V \in \mathcal{F}} ||V^* - V||.$$

Two principles

- Given a norm on a functional space

$$\min_{V \in \mathcal{F}} ||V^* - V||.$$

- Other idea: Bellman residual.

$$\min_{V \in \mathcal{F}} ||TV - V||.$$

where T (or T^π) is the linear or non-linear Bellman operator.

Some results

Take the case with discount $0 < \gamma < 1$. Let V^* the optimal value function of a given MDP.

Proposition

- $$||V - V^*||_\infty \leq \frac{1}{1-\gamma} ||TV - V||_\infty,$$
- *If* $V_{BR} = \arg\min_{V \in \mathcal{F}} ||TV - V||_\infty$,
  $$||TV_{BR} - V_{BR}||_\infty \leq (1+\gamma) \inf_{V \in \mathcal{F}} ||V^* - V||_\infty.$$
- *Let* π_{BR} *the policy greedy w.r.t.* V_{BR}. *Then:*
  $$||V_{\pi_{BR}} - V^*||_\infty \leq \frac{(1+\gamma)}{1-\gamma} \inf_{V \in \mathcal{F}} ||V^* - V||_\infty.$$

But in practice...

- Though the infinity norm gives nice theoretical inequalities, it is usually impossible to work with it in practice (because we need to estimate values at too many points).

- So usually, some L_2 norms (possibly regularized with L_1) are preferred.
- But the nice "contraction properties" might vanish.

Practical schemes

Let us look at how to reach practical algorithms in L_2.

- Take the policy π fixed.
- For the evaluation problem, we want to find V such that:

$$\min_{V \in \mathcal{F}} ||V^\pi - V||_2,$$

$$||f||_2^2 = \sum_s f(s)^2 \mu_s.$$

- μ is a measure that has to do with the problem... (stationary measure, weights for states,...). To simplify , we take μ constant.

Optimisation principles

We define the objective function:

$$E(w) = \sum_{s} \Big(V^{\pi}(s) - \hat{V}(s, w) \Big)^2.$$

$$E(w) = E\Big(V^{\pi}(S) - \hat{V}(S, w) \Big)^2.$$

Optimisation principles

We define the objective function:

$$E(w) = \sum_{s} \Big(V^{\pi}(s) - \hat{V}(s, w) \Big)^2.$$

$$E(w) = E\Big(V^{\pi}(S) - \hat{V}(S, w) \Big)^2.$$

WHAT NEXT?

Optimisation principles

Clearly we do not know $V^\pi(s)$.

One can tries a gradient descent algorithm. (With the usual difficulties...).

$$\Delta w = -\alpha \nabla E(w) = \alpha E\Big((V^\pi(S) - \hat{V}(S,w)) \nabla \hat{V}(S,w) \Big).$$

Stochastic version

Stochastic gradient descent :

$$\Delta w = \alpha(V^{\pi}(S_t) - \hat{V}(S_t, w))\nabla \hat{V}(S_t, w).$$

In the case of linear combination $\hat{V}(s, w) = x(s)^{\top} w$:

$$\Delta w = \alpha(V^{\pi}(S_t) - \hat{V}(S_t, w))x(S_t).$$

Stochastic Gradient Descent with targets

Since we are in the model free world.

Recipe: We replace $V^{\pi}(S_t)$ by our favorite approximation.

MC

$$\Delta w = \alpha(G_t - \hat{V}(S_t, w))x(S_t).$$

TD

$$\Delta w = \alpha(R_t + \gamma \hat{V}(S_{t+1}, w) - \hat{V}(S_t, w))x(S_t).$$

$TD(\lambda)$

$$\Delta w = \alpha(G_t^{\lambda} - \hat{V}(S_t, w))x(S_t).$$

Stochastic Gradient Descent with targets

Since we are in the model free world.

Recipe: We replace $V^{\pi}(S_t)$ by our favorite approximation.

MC

$$\Delta w = \alpha(G_t - \hat{V}(S_t, w))x(S_t).$$

TD

$$\Delta w = \alpha(R_t + \gamma \hat{V}(S_{t+1}, w) - \hat{V}(S_t, w))x(S_t).$$

$TD(\lambda)$

$$\Delta w = \alpha(G_t^{\lambda} - \hat{V}(S_t, w))x(S_t).$$

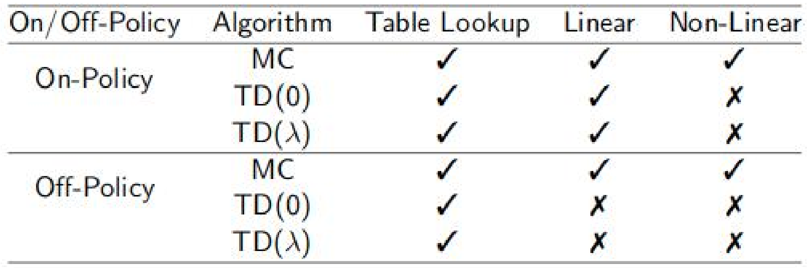
Convergence guarantees?

Convergence and properties

- **Careful:** For TD versions, this is not completely sound, because, we are not really computing a gradient there. (TD estimates ae biased).
- That can lead to divergence.
- Some modified versions (with corrections) allow to reach convergence.
- Tradeoff for bootstrapping and memory...

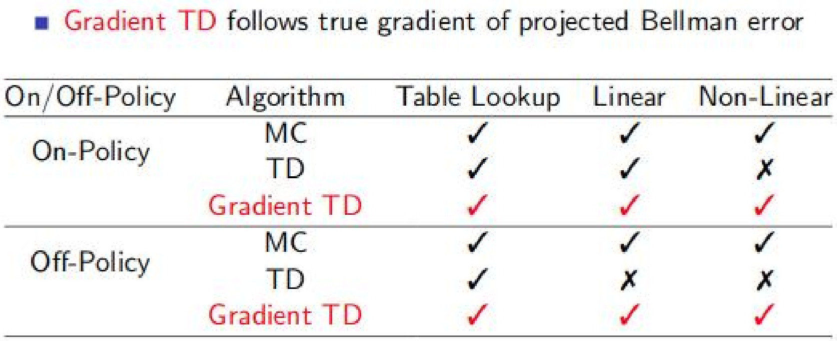
Convergence?

- Linear case. μ stationary distribution, early results of Tsitsiklis and Van Roy.
- Problems with off-policy.
  Baird example (in Barto and Sutton).



On/Off-Policy	Algorithm	Table Lookup	Linear	Non-Linear
On-Policy	MC	✓	✓	✓
	TD(0)	✓	✓	✗
	TD(λ)	✓	✓	✗
Off-Policy	MC	✓	✓	✓
	TD(0)	✓	✗	✗
	TD(λ)	✓	✗	✗

Improved results



- Gradient TD follows true gradient of projected Bellman error

On/Off-Policy	Algorithm	Table Lookup	Linear	Non-Linear
On-Policy	MC	✓	✓	✓
	TD	✓	✓	✗
	Gradient TD	✓	✓	✓
Off-Policy	MC	✓	✓	✓
	TD	✓	✗	✗
	Gradient TD	✓	✓	✓

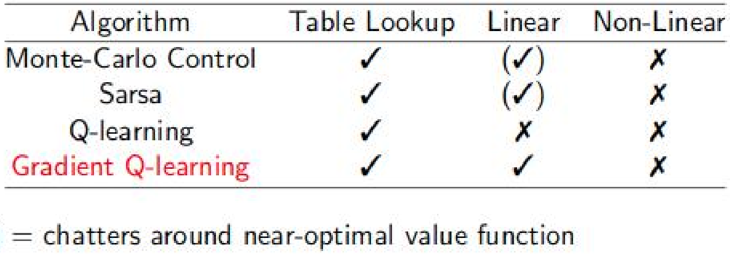
Figure: better results with finer methods (Slide D. Silver)

Control case.

One possible generic strategy: Generalized Policy Iteration + approximation.

- Of course, we use Q instead of V.
- Incremental method: SGD to estimate (the weights) $\hat{Q}$,
- ϵ-greedy w.r.t $\hat{Q}$ for exploration.

Convergence for control schemes



Algorithm	Table Lookup	Linear	Non-Linear
Monte-Carlo Control	✓	(✓)	✗
Sarsa	✓	(✓)	✗
Q-learning	✓	✗	✗
Gradient Q-learning	✓	✓	✗

(✓) = chatters around near-optimal value function

Figure: Convergence (Slide D. Silver)

Issues with Vanilla incremental methods

- Correlations along trajectories might play against approximation errors.
- Incremental methods are usually not the best ones. (for instance problems of divergence when using bootstrapping).
- They do not use efficiently all the information available.
- One can try to perform the best fitting given the data available.

Batch methods

- LSTD
- Neural networks with replay

Instead of doing small steps, we can try to break the problem into various bigger supervised learning problems.

Use one (several pieces of) or several trajectories.

A supervised learning framework

We define "training data" $S_1, G_1^\lambda, S_2, G_2^\lambda, \dots, S_n, G_n^\lambda$, that we are going to use as training set.
Least squares algorithms find parameter vector w minimising sum-squared error between $\hat{V}(S_t, w)$ and target values G_t^λ ,

$$LS(w) = \sum_{t=1}^{T} (G_t^\lambda - \hat{V}(S_t, w))^2.$$

Empirical version of

$$E((V^\pi(S) - \hat{V}(S, w))^2).$$

But...empirical version is biased for $\lambda \neq 1$ (i.e not MC).
There are ways to make the bias vanish using independent trajectories.

ANOTHER DIRECTION FOR APPROXIMATION

Different strategies with different advantages

- Scaling to bigger state space : Approximation of value functions (done)
- Scaling to bigger state space : Approximation of policies.
- Scaling to bigger state space : Approximation of value functions and policies.

Different strategies with different advantages

- **Modelling issues** Value functions approximation vs policy first depend on the problem. It can be simpler to parametrize one or the other e.g., Games. For some problems, the structure of the optimal value functions is known and that leads to good parametrization of value functions.
  for continuous state/action control, the policy has to be parametrized...

- **The max can be a problem.** Computing the max for a continuous control might be problematic.

Pros and cons

- **Convergence issues** Value function approximation with Q-learning might be very aggressive: you can change drastically from action to another because of the max term. On the contrary, gradient updates on the policy might much smoother with better convergence properties...

- **You can learn stochastic policies** This can be crucial for games or for partially observed systems.

Disadvantages:

- It could be **slow, high variance**...
- **Convergence to what?** Might not converge to a good local optimum...

$$x(s, right) = [1, 0],$$

$$x(s, left) = [0, 1].$$

Toy model of uncertainty due to representation/ missing information.

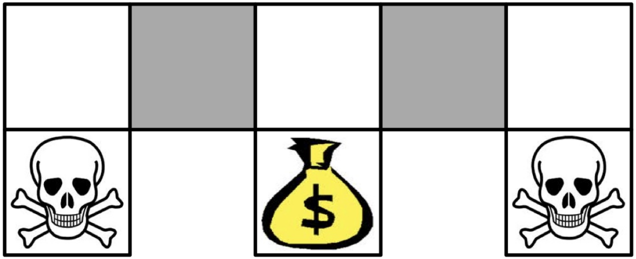


Figure: Image Barto and Sutton: Example with better stochastic policy

Partial information on the state. Assume the agent cannot differnetiate the grey states (because of the specific representations taken for instance.)

Toy model of uncertainty due to representation/ missing information.

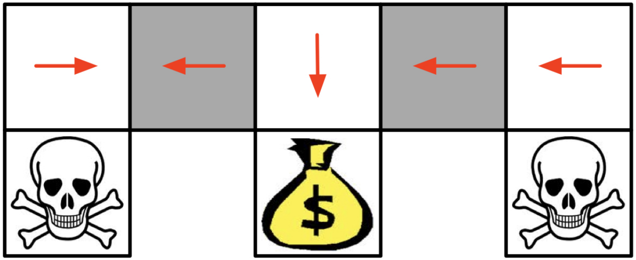


Figure: Image Barto and Sutton: Optimal deterministic policy

Toy model of uncertainty due to representation/ missing information.

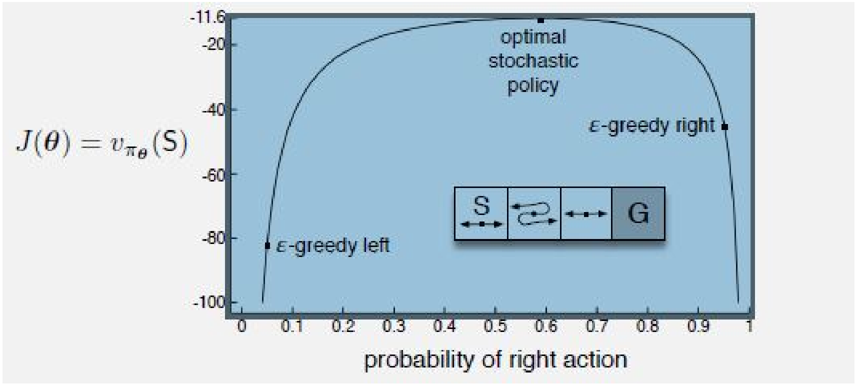


Figure: Image D. Silver: Example with better stochastic policy

Which objective functions for approximating the policy?

Something around the value function weighted per state.

$$J(\theta) = \sum_s \mu(s) V^{\pi^\theta}(s).$$

μ can be

- Some initial states for episodic problems,
- the stationary measure for continuing problems,
- Some arbitrary weights on states you are most interested in.

Principle

Instead on relying on (sample version of) Bellman equation and defining ϵ-greedy policy, here we aim at optimising a weighted sum of the per-state global objective.

What about exploration?

How to optimize

- Direct optimizations (simplex, genetic algorithms...)
- It can be a good idea to have a sequential method.
- Stochastic version of gradient ascent.

Finite differences if $J(\theta)$ is not differentiable:

$$\Delta\theta = \alpha\epsilon^{-1}\Big(J(\theta + \epsilon e_k) - J(\theta)\Big).$$

if differentiable SGD:

$$\Delta\theta = \alpha\nabla_\theta J(\theta).$$

Example of simple parametrizations

Soft-max or Boltzman distributions:

$$\pi_\theta(a|s) = \frac{e^{\theta^T.x(s,a)}}{\sum_c e^{\theta^T.x(s,c)}}.$$

Hence

$$\frac{\partial}{\partial\theta_i}\log(\pi_\theta(a|s)) = x_i(s,a) - \frac{\sum_c x_i(s,c)e^{\theta^T.x(s,c)}}{\sum_c e^{\theta^T.x(s,c)}}.$$

$$\boxed{\nabla_\theta \log(\pi_\theta(a|s)) = x(s,a) - E^{\pi^\theta}(x(s,A)).}$$

Example of simple parametrizations for continuous actions

Gaussian representations:
We define a mean depending on weights and features

$$\mu(s) = \theta^T . x(s),$$

Then, the action follows a Gaussian distribution

$$A \sim \mathcal{N}(\mu(s), 1).$$

and the density of taking action a, $\pi_\theta(a|s)$ updates as follows:

$$\boxed{\nabla_\theta \log(\pi_\theta(a|s)) = (a - \mu(s))x(s).}$$

Policy gradient

Define η the "on-policy" distribution, i.e, the distribution that one would see if playing many times the episodes with a given distribution and choosing at random a time in the episode...

Theorem (**The policy gradient theorem**)

Episodic version

$$\boxed{\nabla_\theta J(\theta) = c. \sum_s \eta(s) \sum_a \nabla_\theta \pi^\theta(a|s) q_{\pi^\theta}(s,a).}$$

Stationary version

Theorem (**The policy gradient theorem)**

$$\boxed{\nabla_\theta J(\theta) = c. \sum_s \mu(s) \sum_a \nabla_\theta \pi^\theta(a|s) q_{\pi^\theta}(s,a).}$$

with μ the stationary measure.

How do we use that?

$$\nabla_\theta J(\theta) = c.\sum_s \mu(s) \sum_a \pi^\theta(a|s) \frac{\nabla_\theta \pi^\theta(a|s)}{\pi^\theta(a|s)} q_{\pi^\theta}(s,a),$$

$$\boxed{\nabla_\theta J(\theta) = E^{\pi^\theta}\left[\nabla_\theta \log(\pi^\theta(A|S)) q_{\pi^\theta}(S,A)\right].}$$

With A, S some random variables following the on Policy distribution (or the stationary distribution). So in practice we just follow the trajectories of our policy.

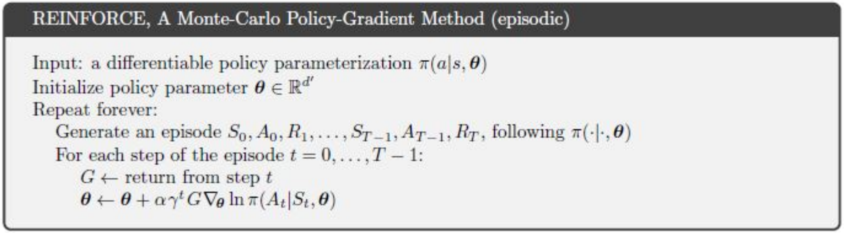
What shall we do next?

First idea

$$
\begin{aligned}
\nabla_\theta J(\theta) &= E^{\pi^\theta}\Big[\nabla_\theta \log(\pi^\theta(A_t|S_t))q_{\pi^\theta}(S_t, A_t)\Big], && (1)\\
&= E^{\pi^\theta}\Big[\nabla_\theta \log(\pi^\theta(A_t|S_t))E(G_t|A_t, S_t]\Big], && (2)\\
&= E^{\pi^\theta}\Big[\nabla_\theta \log(\pi^\theta(A_t|S_t))G_t\Big]. && (3)
\end{aligned}
$$

Replace the unknown $q_{\pi^\theta}(S_t, A_t)$ by an unbiased sample and follow SG Ascent.

$$\boxed{\theta_{t+1} = \theta_t + \alpha\nabla_\theta \log(\pi^\theta(A_t|S_t))G_t.}$$

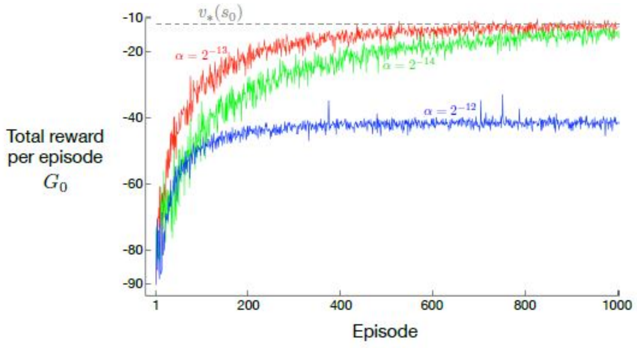


REINFORCE, A Monte-Carlo Policy-Gradient Method (episodic)

```
Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$
Repeat forever:
    Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot,\boldsymbol{\theta})$
    For each step of the episode $t = 0, \ldots, T-1$:
        $G \leftarrow$ return from step $t$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\gamma^t G \nabla_{\boldsymbol{\theta}} \ln \pi(A_t|S_t,\boldsymbol{\theta})$
```

Note that the θ updates should in theory wait for the complete computations of the targets, which is not done in practice.

Performance of Reinforce



Drawbacks

- Need well chosen step size (very small to get sufficient exploration).
- Very sensitive to step size.
- Big variance.

Baseline

Take any function (possibly random) $b(s,\theta)$ of the state space **not depending on actions**.

Theorem (**The policy gradient theorem with baseline)**

$$\boxed{\nabla_\theta J(\theta) = c.\sum_s \mu(s) \sum_a \nabla_\theta \pi^\theta(a|s)(q_{\pi^\theta}(s,a) - b(s,\theta)).}$$

Indeed,

$$\sum_a b(s,\theta)\nabla_\theta \pi^\theta(a|s) = b(s,\theta)\nabla_\theta \sum_a \pi^\theta(a|s) = b(s,\theta)\nabla_\theta 1 = 0.$$

Baseline

What baselines to choose?
Natural first choice is to normalize (rescale) the G_t to normalize the gradients steps...
So theoretically $b(s) = E(G_t)$ and replace G_t by

$$G_t^* = \frac{G_t - E(G_t)}{\sigma_G}.$$

A good choice is already $E(G_t) = V^{\pi^\theta}(s)$.
In practice, we need again to plug in an estimate...

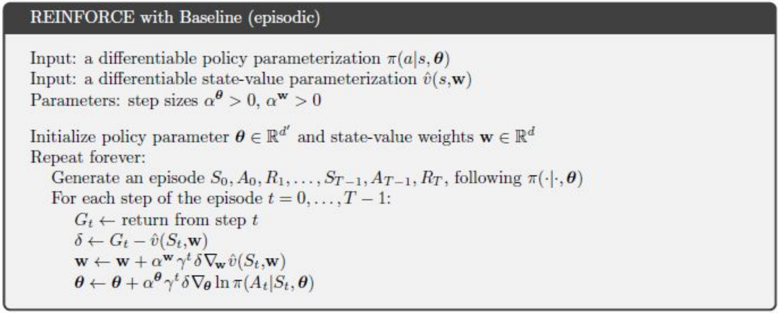
REINFORCE with baseline

Use another estimate of the state value function (through another approximation).

$b(S_t) = \hat{V}(S_t, w)$, with w some weights learned through another approximation scheme.

Advantage function:

$$A(s,a) = G_t - \hat{V}(S_t, w)$$

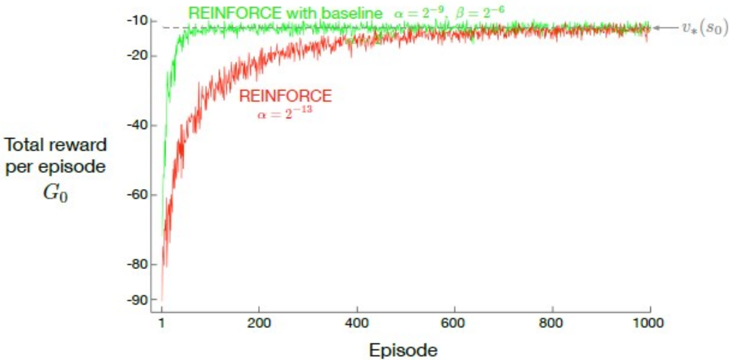


REINFORCE with Baseline (episodic)

```
Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$

Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$
Repeat forever:
    Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot,\boldsymbol{\theta})$
    For each step of the episode $t = 0, \ldots, T-1$:
        $G_t \leftarrow$ return from step $t$
        $\delta \leftarrow G_t - \hat{v}(S_t,\mathbf{w})$
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \gamma^t \delta \nabla_{\mathbf{w}} \hat{v}(S_t,\mathbf{w})$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} \gamma^t \delta \nabla_{\boldsymbol{\theta}} \ln \pi(A_t|S_t,\boldsymbol{\theta})$
```

Learning rate



Where are we?

- **Actor-only methods** (vanilla policy gradient) use parameterized family of policies. The gradient of a combination of state value function is estimated, and the parameters of the policies are updated in a direction of improvement.
  A possible drawback of such methods is that the gradient estimators may have **a large variance.** The tradeoff exploration exploitation is done through the gradient steps and could be difficult to tune.
  Furthermore, as the policy changes, a new gradient is estimated independently of past estimates (by sampling trajectories). Hence, there is no "accumulation of knowledge" and consolidation of older information.

- **Critic-only methods** (e.g., Q-learning, TD-learning) rely exclusively on value function approximation and aim at learning an approximate solution to the Bellman equation, which will then hopefully prescribe a near- optimal policy.

Actor critic: the best of both world?

- Still high variance problems and slow learning with REINFORCE
- Why not try bootstrapping?
- Architecture with two asymmetric "agents"
- $\Rightarrow$ Critic: approximation (neural network) for learning value functions,
- $\Rightarrow$ Actor: another approximation for policy.
- Replace Monte Carlo target by a TD target.
- Problem: We loose here the property of following the theoretical gradient, we somehow approximate the gradient.