

# Lecture 4: Model-free control, thicc state spaces

## SUMS 707 - Basic Reinforcement Learning

Gabriela Moisesu-Pareja and Viet Nguyen

McGill University, Mila

February 4, 2021

# Recap of last lecture

Last time we discussed:

- Unknown environment: predict the value function, improve policy
- Model-free prediction: Temporal difference, Monte-Carlo methods
- Generalized policy iteration: evaluation step + improvement step

## Recap: TD and MC I

From this idea:

$$V^\pi(s_t) = \mathbb{E} [r_{t+1} + \gamma V^\pi(s_{t+1}) | \pi]$$

We said to ourselves, why not use the integrand as an estimation for the value function? We defined the TD target:

$$r_{t+1} + \gamma V(s_{t+1})$$

and we update  $V$ , our estimate for  $V^\pi$ , using:

$$V(s_t) \leftarrow V(s_t) + \alpha(r_{t+1} + \gamma V(s_{t+1}) - V(s_t))$$

If this converges, it converges exactly to  $V^\pi$ !

## Recap: TD and MC II

What if we don't bootstrap at all?

$$V^\pi(s_t) = \mathbb{E} [\mathcal{R}_t | \pi]$$

where  $\mathcal{R}_t$  is the return at time  $t \in [T]$ . With the *MC target*, we update:

$$V(s_t) \leftarrow V(s_t) + \alpha(\mathcal{R}_t - V(s_t))$$

We've seen how to implement every-visit MC. We've also seen that MC converges  $V$  to the true value function  $V^\pi$ .

On the convergence of TD and MC, check out *Algorithms for Reinforcement Learning* by Csaba Szepesvari.

## Recap: TD and MC, pros and cons

- TD can learn without the final outcome
  - TD can learn from incomplete sequences
  - MC needs complete sequences
  - TD works in continuing environments
  - MC only works for episodic environments

There are interesting pathological environments where one converges faster than the other.

In general, TD higher bias, lower variance, MC unbiased, very high variance. This arises from the fact that  $\mathcal{R}_t$  depends on *many* random actions, transitions and rewards.

# TD( $\lambda$ ) I

Unifies TD and MC. Recall the  $n$ -step return:

$$\mathcal{R}_t^{(n)} = r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n V(s_{t+n})$$

We have the  $n$ -step TD-learning update rule

$$V(s_t) \leftarrow V(s_t) + \alpha \left( \mathcal{R}_t^{(n)} - V(s_t) \right)$$

TD( $\lambda$ ) takes a weighted average of all these  $n$ -step returns

## TD( $\lambda$ ) II

TD( $\lambda$ ) takes a weighted average of all these  $n$ -step returns

$$\mathcal{R}_t^\lambda = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} \mathcal{R}_t^{(n)}$$

and updates

$$V(s_t) \leftarrow V(s_t) + \alpha \left( \mathcal{R}_t^\lambda - V(s_t) \right)$$

This is the forward view. The backward view uses *eligibility traces*, and enables computation on incomplete episodes, and online learning.

# Generalized policy iteration

We recall that the policy iteration algorithm cycles through two steps:

- Evaluate a policy (find  $V^\pi$ )
- Improve the policy (find  $\pi' \geq \pi$ )

EZ if we know the environment (solve with DP). When we don't know the environment, we can evaluate our policy with TD or MC.

But how do we improve the policy? We can do the greedy thing:

$$\pi'(s) = \arg \max_{a \in \mathcal{A}} Q(s, a)$$



## Maybe greedy might not be the way to go here...

- You open door 1 and get a reward of 0, you update your action-value function
- You open door 2 and get a reward of  $+1$ , you update your action-value function
- You are greedy. You open door 2 again and get a reward of  $+3$ . You update your action-value function
- You suddenly remember rewards are stochastic
- But you are greedy
- You open door 2 again and get a reward of  $+1$
- ...

# Plan for today

- Model-free control:
  - $\epsilon$ -greedy exploration
  - $\{\text{TD}, \text{MC}\} + \epsilon$ -greedy?
  - SARSA, Q-learning
  - Large state spaces, function approximation

# $\epsilon$ -greedy exploration

Extremely useful

- With probability  $1 - \epsilon$  we choose the greedy action
- With probability  $\epsilon$  we choose an action at random

$$\pi(a|s) = \begin{cases} \epsilon/m + 1 - \epsilon, & a = \arg \max_{a \in \mathcal{A}} Q(s, a) \\ \epsilon/m, & \text{otherwise} \end{cases}$$

# $\epsilon$ -greedy policy improvement

## Theorem

*For any  $\epsilon$ -greedy policy  $\pi$ , the  $\epsilon$ -greedy policy  $\pi'$  w.r.t.  $Q^\pi$  is an improvement,  $V^{\pi'}(s) \geq V^\pi(s)$*

Recall

$$Q^\pi(s, \pi'(s)) = \sum_{a \in \mathcal{A}} \pi'(a|s) Q^\pi(s, a)$$

But you know  $\pi'$  (given in the previous slide), so you can algebra your way to victory.

# MC Control with $\epsilon$ -greedy

- Sample the  $k^{th}$  episode using  $\pi$ :  $s_0, a_0, r_1, \dots, r_T, s_T$
- For each  $s_t$  and  $a_t$  in the episode,

$$Q \approx Q^\pi \quad \begin{aligned} N(s_t, a_t) &\leftarrow N(s_t, a_t) + 1 \\ Q(s_t, a_t) &\leftarrow Q(s_t, a_t) + \frac{1}{N(s_t, a_t)}(\mathcal{R}_t - Q(s_t, a_t)) \end{aligned}$$

- Improve the policy according to the updated  $Q$ :

$$\epsilon \leftarrow 1/k$$

$$\pi \leftarrow \epsilon\text{-greedy}(Q)$$

# Convergence guarantees

$$MC + \epsilon \text{ greedy}$$

## Theorem

*With the assumptions*

- *All state-action pairs are explored infinitely many times*
- *The policy converges on a greedy policy MC control converges to the optimal action-value function, that is,  $Q(s, a) \rightarrow Q^*(s, a)$ .*

# TD Control

MC is cool but the potential problems persist:

- MC methods in general have higher variance than TD methods
- TD can be performed online
- TD can learn from incomplete sequences

What if instead of looping MC control  $\leftrightarrow$   $\epsilon$ -greedy, we loop TD control  $\leftrightarrow$   $\epsilon$ -greedy? How would this look like?

# SARSA updates

After every time-step, recall TD learning updates for policy evaluation:

$$V(s_t) \leftarrow V(s_t) + \alpha (r_{t+1} + \gamma V(s_{t+1}) - V(s_t))$$

We mirror the same idea, but now with  $Q$ . Given that we're in state  $s$  we take action  $a$ , the environment transitions us to state  $s'$  and gives reward  $r$ , we take a subsequent action  $a'$ :

$$Q(s, a) \leftarrow Q(s, a) + \alpha (r + \gamma Q(s', a') - Q(s, a))$$

We emphasize that SARSA is an *on-policy* algorithm.

$$V^\pi = \mathbb{E} [r + \gamma V^\pi \mid \pi]$$

$$Q^\pi = \mathbb{E} [r + \gamma Q^\pi(s', a') \mid \pi],$$



# SARSA control I

We maintain  $Q$ , our estimate for  $Q^\pi$ . We play the  $\epsilon$ -greedy policy, in part., w.p.  $\epsilon$  we do something random, otherwise we take action

$$\pi(s) = \arg \max_a Q(s, a)$$

Every timestep,

- In state  $s_t$  we choose  $a_t$ , we obtain reward  $r_{t+1}$  and transition to state  $s_{t+1}$ , at which we take another action  $a_{t+1}$
- We update  $Q$  according to

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha (r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$$

- Now that our  $Q$  has changed, our  $\epsilon$ -greedy policy may have changed as well!

## Theorem

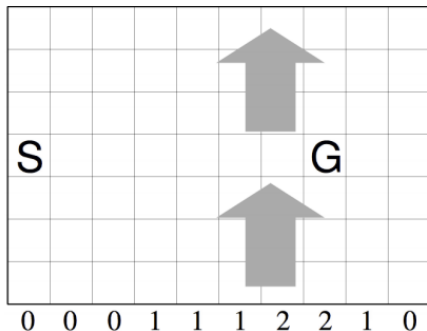
Given that:

- All state-action pairs are explored infinitely many times
- The policy converges on a greedy policy (we can decay  $\epsilon$ )
- Robbins-Monro learning rates:

$$\sum \alpha_t = \infty, \quad \sum \alpha_t^2 < \infty \quad \alpha_t = \frac{1}{t}$$

Then, SARSA converges to the optimal action-value function  $Q^*$ .

# SARSA on Windy Gridworld I



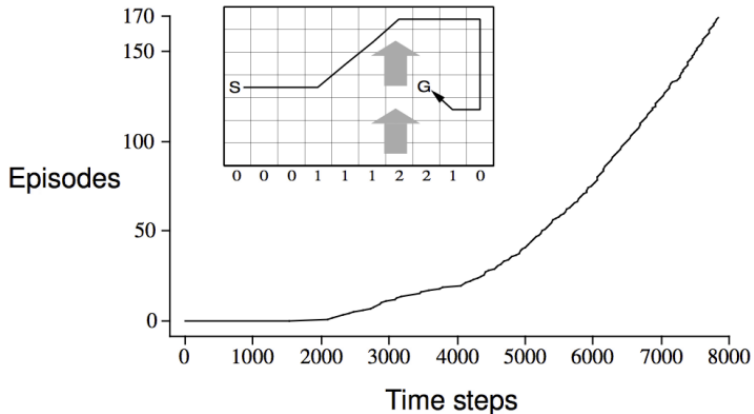
standard  
moves



king's  
moves

- Reward = -1 per time-step until reaching goal
- Undiscounted

# SARSA on Windy Gridworld II



# SARSA( $\lambda$ )

Similar to TD( $\lambda$ ), define  $n$ -step  $Q$ -return:

$$q_t^{(n)} = r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n Q(s_{t+n})$$

Exponentially weighing the  $n$ -step  $Q$ -returns:

$$q_t^\lambda = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} q_t^{(n)}$$

We update:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left( q_t^\lambda - Q(s_t, a_t) \right)$$

Just like TD( $\lambda$ ), one can view this algorithms backwards with *eligibility traces*, enabling online learning.

# Off-policy learning

When we update  $Q$  with SARSA, we are estimating  $Q^\pi$ , while following  $\pi$ .

Off-policy learning:

- Evaluate a target policy  $\pi$  to compute  $Q^\pi$
- We follow  $\mu$  instead of  $\pi$ ,

$$s_1, a_1, r_1, \dots, r_t, s_t \sim \mu$$

Why?

- Learn from observing humans or other agents
- ***Re-use experience generated from old policies***
- Learn about the optimal policy while following an exploratory policy
- Learn about multiple policies while following one policy

# Importance Sampling

$\frac{P}{Q}$

$$\begin{aligned}\mathbb{E}_{X \sim P} [f(X)] &= \sum f(X) P(X) \\ &= \sum Q(X) \frac{P(X)}{Q(X)} f(X) \\ &= \mathbb{E}_{X \sim Q} \left[ \frac{P(X)}{Q(X)} f(X) \right]\end{aligned}$$

# Off-policy MC

Recall:  $V^\pi(s_t) = \mathbb{E}[\mathcal{R}_t | \pi]$ , but in this case,  $\mathcal{R}_t$  is obtained following  $\pi$ .

If we want to learn about  $\pi$  from our history of playing  $\mu$ , we have to multiply  $\mathcal{R}_t$  by correction terms. Consider the entire trajectory from time  $t$  to time  $T$ , define:

$$\mathcal{R}_t^{\pi/\mu} = \frac{\pi(a_t|s_t)}{\mu(a_t|s_t)} \cdot \frac{\pi(a_{t+1}|s_{t+1})}{\mu(a_{t+1}|s_{t+1})} \cdot \dots \cdot \frac{\pi(a_T|s_T)}{\mu(a_T|s_T)} \mathcal{R}_t$$

We update

$$V(s_t) \leftarrow V(s_t) + \alpha \left( G_t^{\pi/\mu} - V(s_t) \right)$$

In practice, this is a very bad idea, extremely high variance, also cannot use when  $\mu = 0$  when  $\pi \neq 0$ .



# Off-policy TD

It is imperative to use TD learning when you're doing off-policy learning. TD targets generated from  $\mu$  to evaluate  $\pi$ , and weigh them by importance sampling:

$$V(s_t) \leftarrow V(s_t) + \alpha \left( \frac{\pi(a_t|s_t)}{\mu(a_t|s_t)} (r_{t+1} + \gamma V(s_{t+1})) - V(s_t) \right)$$

Essentially weighing “how much we trust the TD target by how much we would've taken that action under policy  $\pi$ ”

This method still increases variance, but much better than off-policy MC.

# Q-learning I

$$(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1} \sim \mu)$$

We consider off-policy learning of  $Q$  instead of  $V$ .

- We want to evaluate  $\pi$  (target policy) following  $\mu$  (behavior policy)
- In state  $s_{t+1}$ , we choose action  $a_{t+1} \sim \mu(\cdot|s_t)$
- We also consider an alternative successor  $a' \sim \pi(\cdot|s_t)$
- Update  $Q(s_t, a_t)$  towards *alternative action*:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha (r_{t+1} + \gamma Q(s_{t+1}, a') - Q(s_t, a_t))$$

Update towards “what would’ve happened if we followed the policy we cared about”

## Q-learning II

When we talk about the Q-learning algorithm we mean:

- Target  $\pi$  is *greedy*:

$$\pi(s_{t+1}) = \arg \max_{a'} Q(s_{t+1}, a')$$

- Behavior  $\mu$  is  $\epsilon$ -*greedy*
- Thus, the Q-learning target simplifies:

$$\begin{aligned} r_{t+1} + \gamma Q(s_{t+1}, a') &= r_{t+1} + \gamma Q(s_{t+1}, \arg \max_{a'} Q(s_{t+1}, a')) \\ &= r_{t+1} + \max_{a'} \gamma Q(s_{t+1}, a') \end{aligned}$$

The Q-learning algo updates as such:

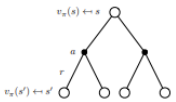

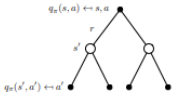
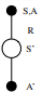

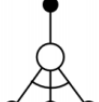
$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left( r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right)$$

## Theorem

*Q-learning control converges to the optimal action-value function, that is,*

$$Q(s, a) \rightarrow Q^*(s, a)$$

# DP and TD I

	Full Backup (DP)	Sample Backup (TD)
Bellman Expectation Equation for $v_{\pi}(s)$	 <p>Iterative Policy Evaluation</p>	 <p>TD Learning</p>
Bellman Expectation Equation for $q_{\pi}(s, a)$	 <p>Q-Policy Iteration</p>	 <p>Sarsa</p>
Bellman Optimality Equation for $q_{*}(s, a)$	 <p>Q-Value Iteration</p>	 <p>Q-Learning</p>

# DP and TD II

<i>Full Backup (DP)</i>	<i>Sample Backup (TD)</i>
Iterative Policy Evaluation $V(s) \leftarrow \mathbb{E}[R + \gamma V(S') \mid s]$	TD Learning $V(S) \stackrel{\alpha}{\leftarrow} R + \gamma V(S')$
Q-Policy Iteration $Q(s, a) \leftarrow \mathbb{E}[R + \gamma Q(S', A') \mid s, a]$	Sarsa $Q(S, A) \stackrel{\alpha}{\leftarrow} R + \gamma Q(S', A')$
Q-Value Iteration $Q(s, a) \leftarrow \mathbb{E}\left[R + \gamma \max_{a' \in \mathcal{A}} Q(S', a') \mid s, a\right]$	Q-Learning $Q(S, A) \stackrel{\alpha}{\leftarrow} R + \gamma \max_{a' \in \mathcal{A}} Q(S', a')$

# Large problems

- Backgammon:  $10^{20}$  states
- Chess:  $\approx 10^{46}$  states
- Go:  $10^{170}$  states
- Helicopter:  $\infty$



How do we scale up the model-free methods for prediction and control?

# Value function approximation

So far, we've only worked with tables:

- $\forall s \in \mathcal{S}$ , there is an entry  $V(s)$
- $\forall (s, a) \in \mathcal{S} \times \mathcal{A}$ , there is an entry  $Q(s, a)$

Instead of maintaining a table, we will *parametrize* our (action-)value functions:

$$V(s; \theta) \approx V^\pi(s)$$

$$Q(s, a; \theta) \approx Q^\pi(s, a)$$

With this method, we will be able to *generalize* from seen states to unseen states.

*Idea:* we **update**  $\theta$  using MC or TD learning!



# From an implementation perspective I

- We can implement  $V(\cdot; \theta)$  as

$$V : \mathcal{S} \rightarrow \mathbb{R}$$

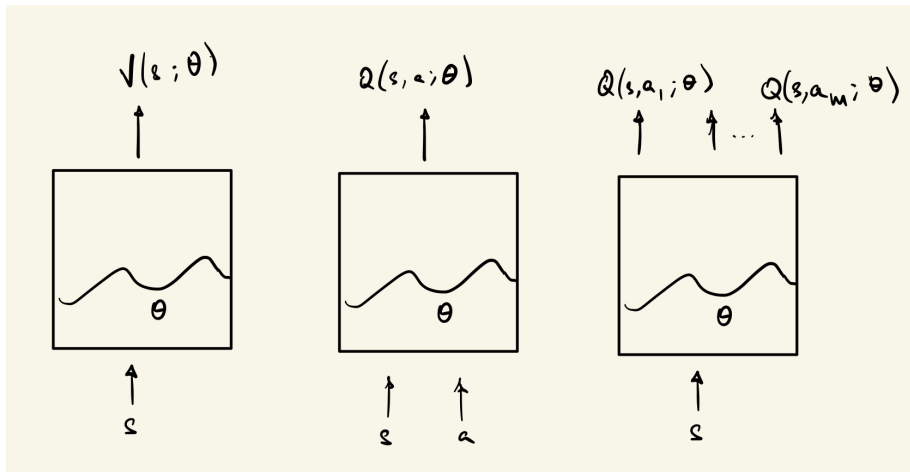
- We can implement  $Q(\cdot, \cdot; \theta)$  as

$$Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$$

- But most often, we will be implementing  $Q$  as

$$Q : \mathcal{S} \rightarrow \mathbb{R}^{|\mathcal{A}|}$$

## From an implementation perspective II



# Types of function approximators

- Linear parametrization
- Neural network approximation
- Decision trees
- Nearest neighbors
- Fourier, wavelets
- ...

In particular, we will study *differentiable* function approximators, namely linear features and neural networks.

Non iid data, non-stationary.

## Two methods

$$\theta \Rightarrow V(e; \theta) \approx \sqrt{\pi}(e).$$

- Incremental methods using gradient descent
- Batch methods using least-squares

# General value function approx. with gradient descent

Define the squared loss function:

$$J(\theta) = \mathbb{E}_{\pi} [(V^{\pi}(s) - V(s; \theta))^2]$$

We move in the direction of the negative gradient of the loss function:

$$\begin{aligned}\Delta\theta &= -\frac{1}{2}\alpha\nabla_{\theta}J(\theta) \\ &= \alpha\mathbb{E}_{\pi} [(V^{\pi}(s) - V(s; \theta))\nabla_{\theta}V(s; \theta)]\end{aligned}$$

Instead of computing this integral, we sample:


$$\Delta\theta = \alpha (V^{\pi}(s) - V(s; \theta)) \nabla_{\theta} V(s; \theta)$$

## No oracle? No problem

In our update, we assumed access to  $V^\pi$ , but in practice, there are no oracles (sadge).

We look at past lecture slides and realize that we've been estimating  $V^\pi$  the whole time!


- For MC, our target is  $\mathcal{R}_t$ :


$$\Delta\theta = \alpha(\mathcal{R}_t - V(s; \theta)) \nabla_\theta V(s; \theta)$$

- For TD(0), our target is  $r_{t+1} + \gamma V(s_{t+1}; \theta)$ :

$$\Delta\theta = \alpha(r_{t+1} + \gamma V(s_{t+1}; \theta) - V(s; \theta)) \nabla_\theta V(s; \theta)$$

- For TD( $\lambda$ ), our target is  $\mathcal{R}_t^\lambda$ :


$$\Delta\theta = \alpha(\mathcal{R}_t^\lambda - V(s; \theta)) \nabla_\theta V(s; \theta)$$

# Linear function approximation

We represent our estimation of the value function as

$$V(s; \theta) = \langle \theta, \varphi(s) \rangle = \varphi(s)^T \theta$$

$$\varphi: \mathcal{S} \rightarrow \mathbb{R}^d$$

where  $\varphi$  is some feature mapping. Our loss function simplifies:

$$J(\theta) = \mathbb{E} \left[ (V^\pi(s) - \varphi(s)^T \theta)^2 \right]$$

Our gradient w.r.t.  $\theta$  is:

$$\nabla_\theta V(s; \theta) = \varphi(s)$$

Thus, our updates look like:

$$\Delta \theta = \alpha (V^\pi(s) - \varphi(s)^T \theta) \varphi(s)$$

# Linear function approximation: MC

At the completion of an episode, we can imagine building a dataset  $\mathcal{D} = \{(s_t, \mathcal{R}_t) : t \in [T]\}$  which we treat as “training data”, we want to fit  $V$  such that it approximately predicts  $\mathcal{R}_t$  in state  $s_t$ , this is done with stoch. grad. descent:

$$\Delta\theta = \alpha(\mathcal{R}_t - \varphi(s_t)^T\theta)\varphi(s_t)$$

$\mathcal{R}_t$  is a noisy unbiased estimate of  $V^\pi(s_t)$

- MC converges to a local optimum
- Still converges when using non-linear function approximation



## Linear function approximation: TD

Although the TD target is a biased sample of the true value  $V^\pi(s_t)$ , we can still apply it in the same way as we did with MC.

We make our dataset  $\mathcal{D} = \{(s_t, r_{t+1} + \phi(s_{t+1})^T \theta) : t \in [T]\}$  and update:

$$\Delta \theta = \alpha (r_{t+1} + \phi(s_{t+1})^T \theta - \phi(s_t)^T \theta) \phi(s_t)$$

This converges close to the global optimum.

Now that you know the increments for  $\theta$  with MC and TD, can you write the increment for TD( $\lambda$ )? Recall the target in TD( $\lambda$ )...

## Remark

The dataset analogy here is purely to relate to supervised learning, however we are doing updates every timestep!

# Exercise!

Write  $\Delta\theta$  when you're using a 1 hidden layer neural network.

# Control in the function approximation setting

We now know how to approximate the value function in large state spaces (MC, TD). How do we improve our policies?

Again, we work with the action-value function.

- Policy evaluation:  $Q^\pi \approx Q(s, a; \theta)$
- $\epsilon$ -greedy policy improvement

# Action-value function approximation

We approximate:  $Q^\pi(s, a) \approx Q(s, a; \theta)$  Our loss is:

$$J(\theta) = \mathbb{E}_\pi [(Q^\pi(s, a)) - Q(s, a; \theta)^2]$$

Our stoch. grad. descent updates in full generality:

$$\Delta\theta = \alpha(Q^\pi(s, a) - Q(s, a; \theta))\nabla_\theta Q(s, a; \theta)$$

# Linear action-value function approximation

With feature map  $\varphi : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}^d$  now instead of  $\varphi : \mathcal{S} \rightarrow \mathbb{R}^d$ . We represent the action-value function by:

$$Q(s, a; \theta) = \langle \varphi(s, a), \theta \rangle = \varphi(s, a)^T \theta$$

$$Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$$

and our stoch. grad. descent updates simplify to:

$$\Delta \theta = \alpha (Q^\pi(s, a) - \varphi(s, a)^T \theta) \varphi(s, a)$$

# Estimating $Q^\pi$

With MC, we use the returns  $\mathcal{R}_t$  to estimate  $Q^\pi$ :

$$\Delta\theta = \alpha(\mathcal{R}_t - Q(s_t, a_t; \theta)) \nabla_\theta Q(s_t, a_t; \theta)$$

With TD(0), we bootstrap the TD target  $r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}; \theta)$ :

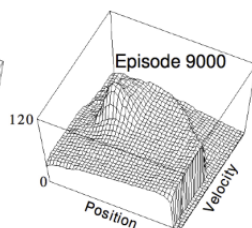
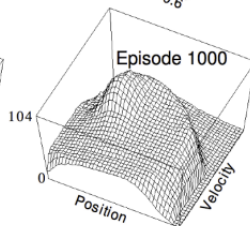
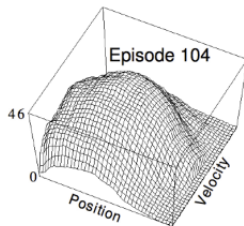
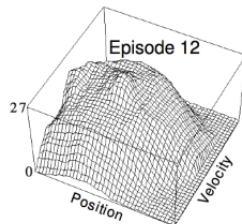
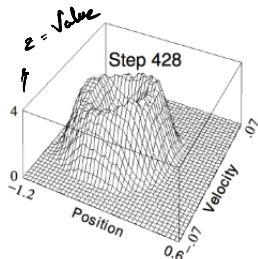
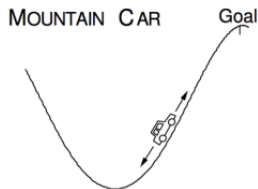
$$\Delta\theta = \alpha(r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}; \theta) - Q(s_t, a_t; \theta)) \nabla_\theta Q(s_t, a_t; \theta)$$

With TD( $\lambda$ ), we use the action value  $\lambda$ -return,  $q_t^\lambda$ :

$$\Delta\theta = \alpha(q_t^\lambda - Q(s_t, a_t; \theta)) \nabla_\theta Q(s_t, a_t; \theta)$$

This is the forward-view of TD( $\lambda$ ).

# Linear SARSA for Mountain Car





# Convergence of Prediction algorithms

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

# Convergence of Control algorithms

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

(✓) = chatters around near-optimal value function

- Our incremental methods are not very sample efficient
- Seeks to find the best fitting value function given the agent's experience (we imagine the agent to remember everything that it has done up until the current timestep)
- Recycle past experiences

# Least Squares

Given our function approximation  $V(s; \theta)$ , we have a hypothetical dataset:

$$\mathcal{D} = \{(s_t, V_t^\pi) : t \in [T]\}$$

We want to find  $\theta$  such that our estimate  $V$  best fits this dataset. This is exactly the problem setup of *least-squares*, we minimize the sum of squared errors:

$$LS(\theta) = \sum_{t=1}^T (V_t^\pi - V(s_t; \theta))^2$$

# Experience replay

We store a dataset  $\mathcal{D}$  as defined previously,

- We sample  $(s, V^\pi) \sim \mathcal{D}$
- We update with stoch. grad. descent:

$$\Delta\theta = \alpha(V^\pi - V(s; \theta))\nabla_\theta V(s, \theta)$$

From theory: this process converges to the least squares solution:

$$\theta^\pi = \arg \min_{\theta} LS(\theta)$$

- Take action  $a_t$  according to  $\epsilon$ -greedy policy
- Store transition  $(s_t, a_t, r_{t+1}, s_{t+1})$  in  $\mathcal{D}$
- Sample a *batch*  $S = (s, a, r, s') \sim \mathcal{D}$
- Compute Q-learning targets w.r.t. old, fixed parameters  $\theta^-$
- Optimize MSE between Q-network and Q-learning targets:

$$MSE(\theta_i) = \mathbb{E} \left[ \left( r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i) \right)^2 \right]$$

# Next lecture

What do we do when we don't know the dynamics of the environment?

- Policy gradients
- Actor-critic methods

# References

- Reinforcement Learning: Theory and Algorithms by Alekh Agarwal, Nan Jiang, Sham M. Kakade
- Algorithms for Reinforcement Learning by Csaba Szepesvári
- Reinforcement Learning: An Introduction by Andrew Barto and Richard S. Sutton
- “Introduction to Reinforcement Learning” Lectures by David Silver
- “CS 598 - Statistical Reinforcement Learning” Notes by Nan Jiang