ATCI: Reinforcement Learning

Function approximation

Mario Martin

CS-UPC

February 29, 2024

Goal of this lecture

- Methods we have seen so far work well when we have a *tabular* representation for each state, that is, when we represent value function with a lookup table.

Goal of this lecture

- Methods we have seen so far work well when we have a *tabular* representation for each state, that is, when we represent value function with a lookup table.
- This is not reasonable on most cases:
  - In Large state spaces: There are too many states and/or actions to store in memory (f.i. Backgammon: 10^{20} states, Go 10^{170} states)
  - and in continuous state spaces (f.i. robotic examples)

Goal of this lecture

- Methods we have seen so far work well when we have a *tabular* representation for each state, that is, when we represent value function with a lookup table.
- This is not reasonable on most cases:
  - In Large state spaces: There are too many states and/or actions to store in memory (f.i. Backgammon: 10^{20} states, Go 10^{170} states)
  - and in continuous state spaces (f.i. robotic examples)
- In addition, we want to generalize from/to similar states to speed up learning. It is too slow to learn the value of each state individually.

Goal of this lecture

- We'll see now methods to learn policies for large state spaces by using *function approximation to estimate value functions*:

$$V_\theta(s) \approx V^\pi(s) \quad (1)$$

$$Q_\theta(s,a) \approx Q^\pi(s,a) \quad (2)$$

- θ is the set of parameters of the function approximation method (with size much lower than $|S|$)
- Function approximation allow to generalize from seen states to unseen states and to save space.
- Now, instead of storing V values, we will update θ parameters using MC or TD learning so they fulfill (1) or (2).

Which Function Approximation?

- There are many function approximators, e.g.
  - Artificial neural network
  - Decision tree
  - Nearest neighbor
  - Fourier/wavelet bases
  - Coarse coding
- In principle, any function approximator can be used. However, the choice may be affected by some properties of RL:
  - Experience is not i.i.d. – Agent's action affect the subsequent data it receives
  - During control, value function V(s) changes with the policy (non-stationary)

Incremental methods

Which Function Approximation?

- Incremental methods allow to directly apply the control methods of MC, Q-learning and Sarsa, that is, back up is done using *"on-line"* sequence of data of the trial reported by the agent following the policy.
- Most popular method in this setting is **gradient descent**, because it adapts to changes in the data (non-stationary condition)

Gradient Descent

- Let $L(\theta)$ be a differentiable function of parameter vector θ, we want to minimize
- Define the gradient of $L(\theta)$ to be:

$$\nabla_\theta L(\theta) = \begin{bmatrix} \frac{\partial L(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial L(\theta)}{\partial \theta_n} \end{bmatrix}$$

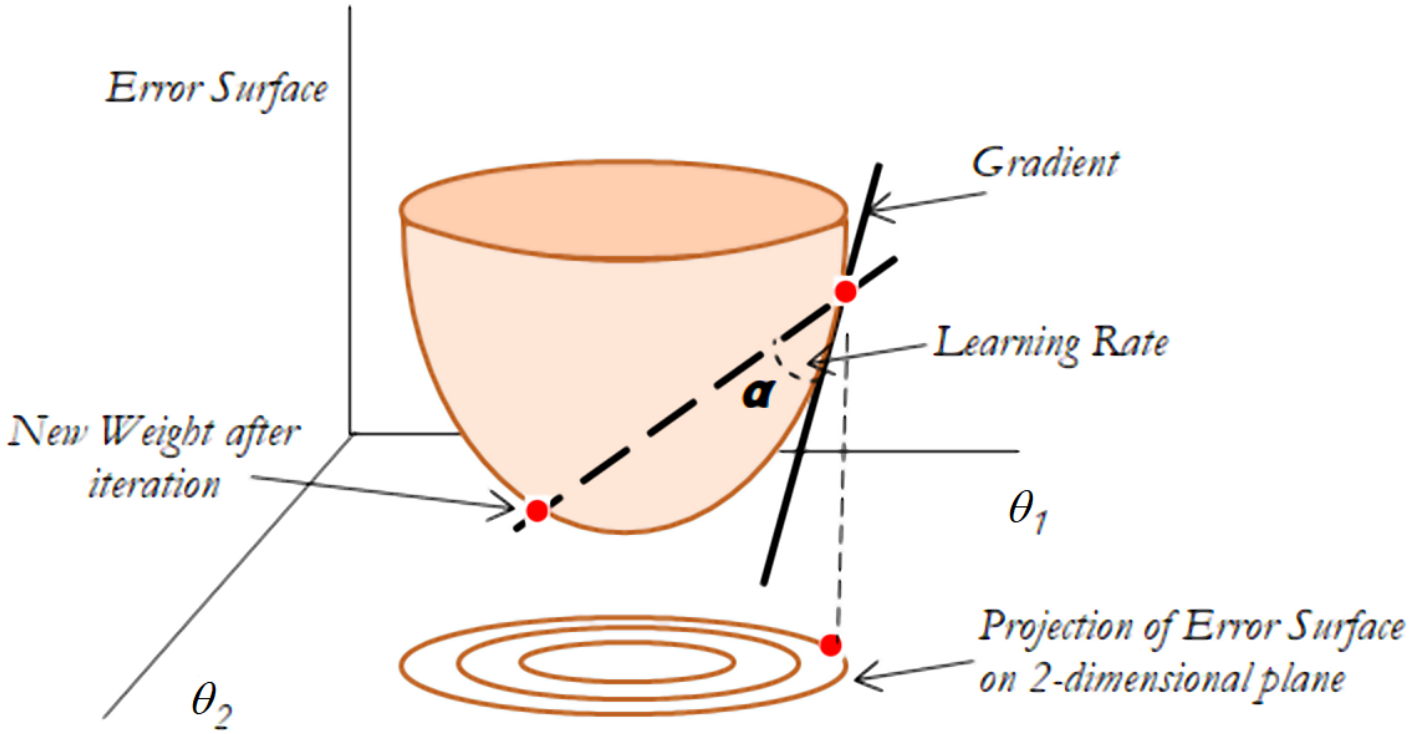
- To find a local minimum of $L(\theta)$, gradient descent method adjust the parameter in the direction of negative gradient:

$$\Delta\theta = -\frac{1}{2}\alpha\nabla_\theta L(\theta)$$

where is a stepsize parameter

Gradient Descent

$$\Delta\theta = -\frac{1}{2}\alpha\nabla_\theta L(\theta)$$



Value Function Approx. by SGD

Minimizing *Loss function* of the approximation

Goal: Find parameter vector θ minimizing mean-squared error between approximate value function $V_\theta(s)$ and true value function $V^\pi(s)$

$$L(\theta) = \mathbb{E}_\pi \left[(V^\pi(s) - V_\theta(s))^2 \right] = \sum_{s \in \mathcal{S}} \mu^\pi(s) \left[V^\pi(s) - V_\theta(s) \right]^2$$

where $\mu^\pi(s)$ is the time spent in state s while following π (probability visiting s following policy)

- Gradient descent finds a *local* minimum:

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

- **Stochastic gradient descent** (SGD) *samples* the gradient

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

Subsection 1

Linear approximation

Linear representation of the state

- Represent state by a feature vector:

$$\phi(s) = \begin{bmatrix} \phi_1(s) \\ \vdots \\ \phi_n(s) \end{bmatrix}$$

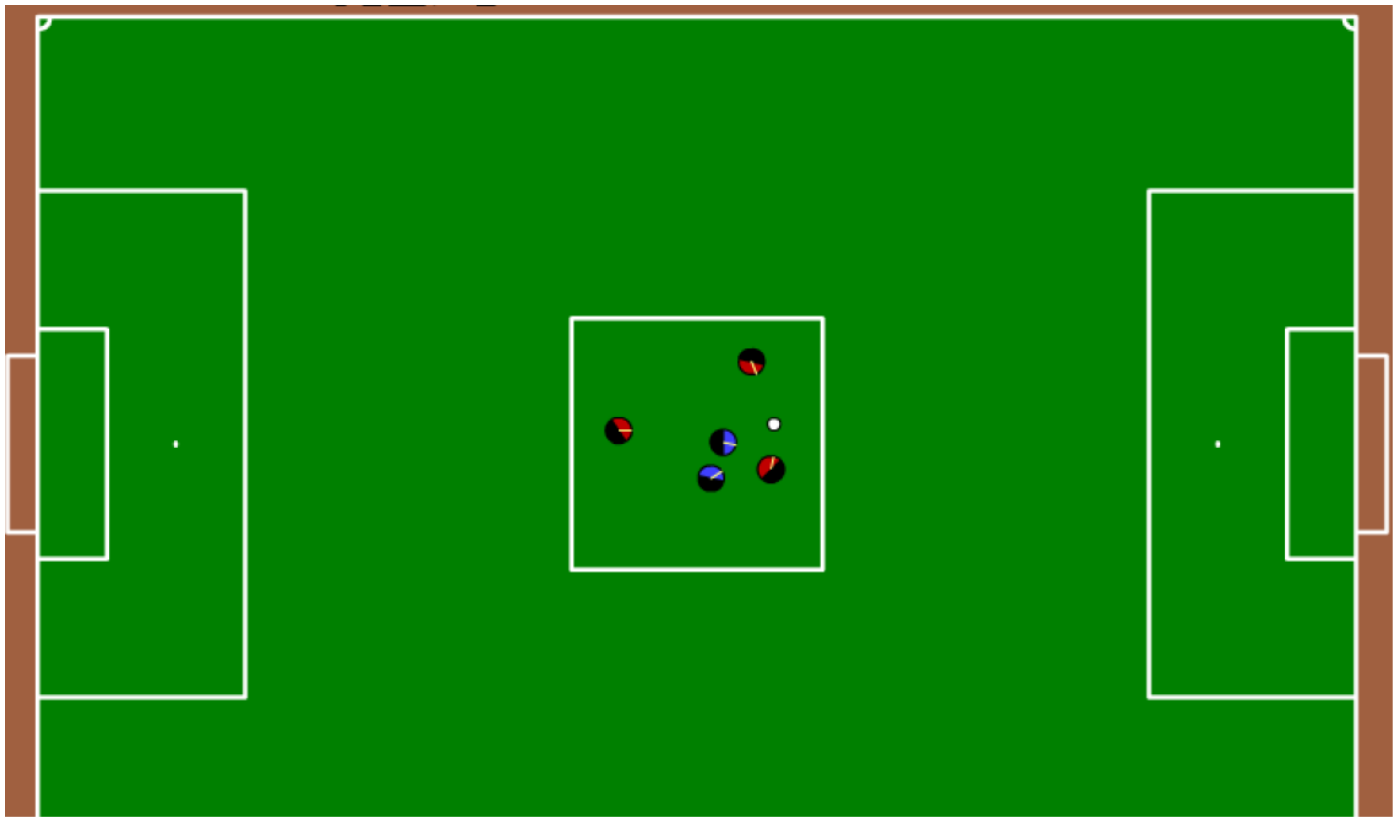
- Represent value function by a linear combination of features:

$$V_\theta(s) = \phi(s)^T \theta = \sum_{j=1}^{n} \phi_j(s)\theta_j \tag{3}$$

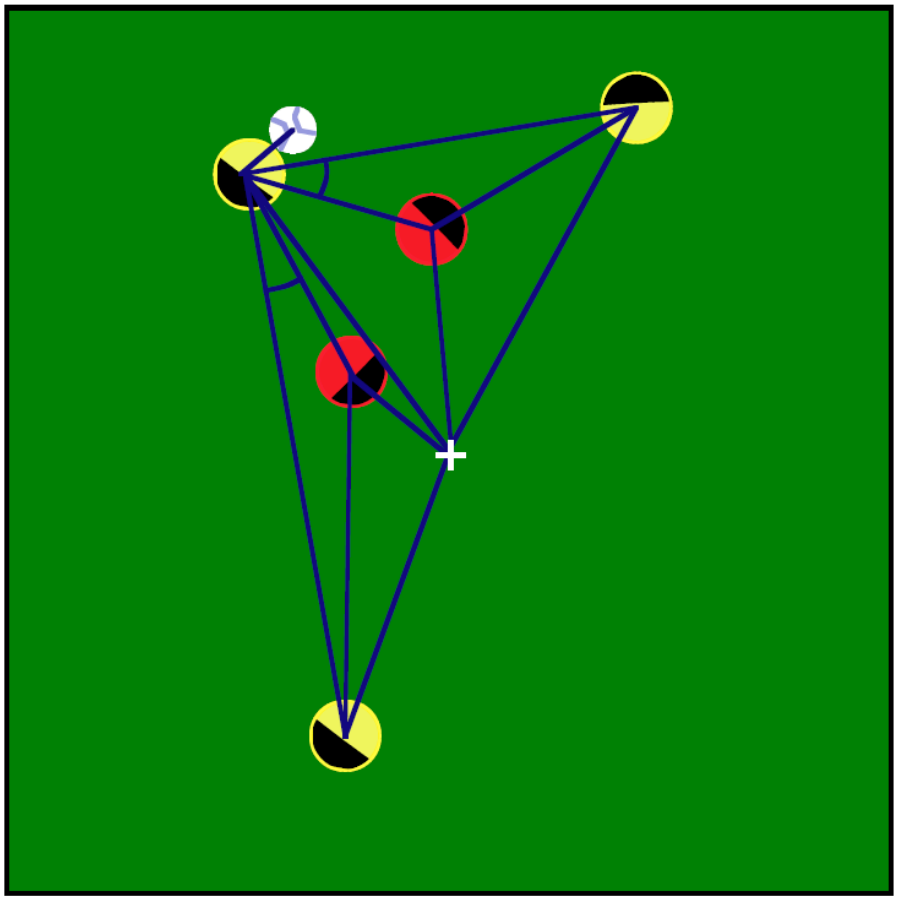
Linear representation of the state

- For example:
  - Distance of robot from landmarks

Example: RoboCup soccer keepaway (Stone, Sutton & Kuhlmann, 2005)



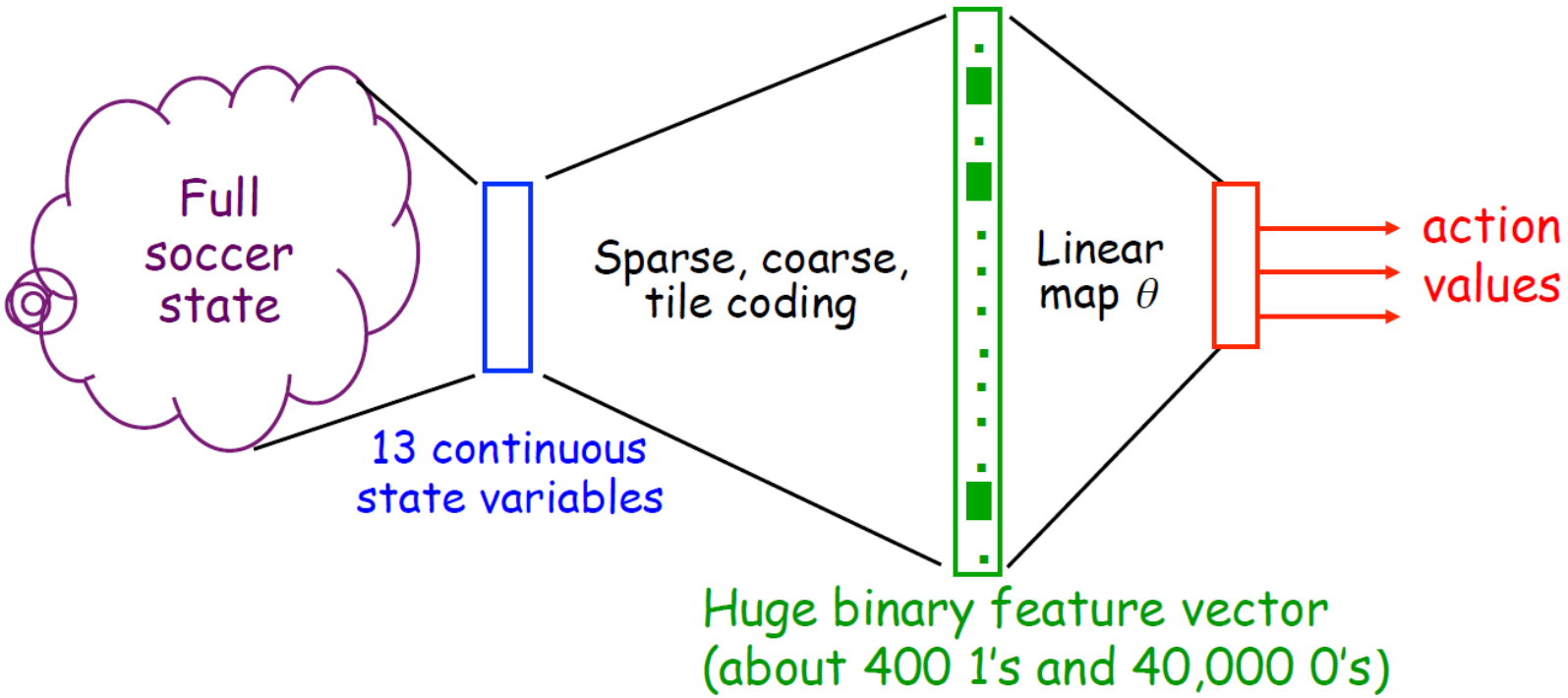
Example: RoboCup soccer keepaway (Stone, Sutton & Kuhlmann, 2005)



State is encoded in 13 continuous variables:

- 11 distances among the players, ball, and the center of the field
- 2 angles to takers along passing lanes

Example: RoboCup soccer keepaway (Stone, Sutton & Kuhlmann, 2005)



Linear representation of the state

- Table lookup is a special case of linear value function approximation. Using table lookup features:

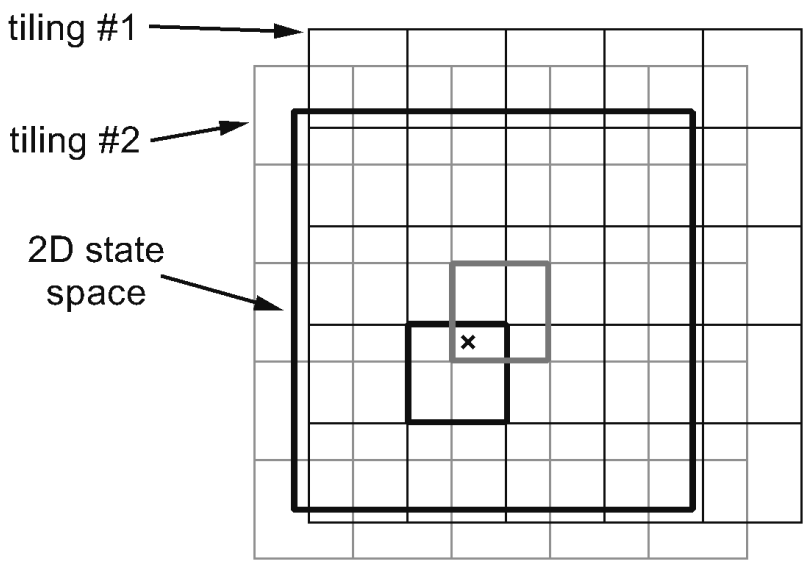
$$\phi^{table}(S) = \begin{bmatrix} 1(S = s_1) \\ \vdots \\ 1(S = s_n) \end{bmatrix}$$

- Parameter vector is exactly value of each individual state

$$V_\theta(S) = \begin{bmatrix} 1(S = s_1) \\ \vdots \\ 1(S = s_n) \end{bmatrix}^T \cdot \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$$

Coarse coding using Tiles

- *RBFs* return a real value for each feature. *Tiles* define a **binary feature** for each tile.
  - Binary features means weighted sum easy to compute
  - Number of features present at any time step is constant
  - Easy to compute indexes of features active



Shape of tiles ⇒ Generalization

#Tilings ⇒ Resolution of final approximation

- You can use irregular tilings or superposition of different tilings

Going back to SGD

First nice property of SGD in linear F.A. (in policy evaluation)

In the case of linear function approximation, objective function is *quadratic*:

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

so SGD converges to global optimum:

Notice equation (3).

Why it converges?

Going back to SGD

First nice property of SGD in linear F.A. (in policy evaluation)

In the case of linear function approximation, objective function is *quadratic*:

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

so SGD converges to global optimum:

Notice equation (3).

Why it converges? Quadratic problem (parabola) has no local minimum.

Going back to SGD

Second nice property of SGD in linear F.A.

Gradient vector of value function is vector of feature values:

$$\begin{aligned}\frac{\partial V_\theta(s)}{\partial \theta_i} &= \frac{\partial \sum_{j=1}^{n} \phi_j(s)\theta_j}{\partial \theta_i} \\ &= \phi_i(s)\end{aligned}$$

So, update rule is particularly simple:

$$\Delta\theta_i = \alpha(V^\pi(s) - V_\theta(s))\ \phi_i(s)$$

TD prediction algorithm for the linear case

Changes applying TD to the linear case:

1. Function approximation is now for the Q value function:

$$Q^{\pi}(s,a) \approx Q_{\theta}(s,a) = \phi(s,a)^{T}\theta = \sum_{j=1}^{n} \phi_j(s,a)\theta_j$$

2. Loss function is also now for Q value function:

$$L(\theta) = \mathbb{E}_{\pi}\left[(Q^{\pi}(s,a) - \phi(s,a)^{T}\theta)^2 \right]$$

TD prediction algorithm for the linear case

- In TD(0) we use Q of *next state* to estimate Q on the *current state* using Bellman equations. So, in general,

$$\begin{aligned}\Delta\theta_i =& \alpha(\quad Q^\pi(s,a) \quad - Q_\theta(s,a))\nabla_\theta Q_\theta(s,a) \\ =& \alpha(r + \gamma Q_\theta(s', \pi(s')) - Q_\theta(s,a))\nabla_\theta Q_\theta(s,a)\end{aligned}$$

- And, in particular, for the linear case:

$$\begin{aligned}\frac{\partial Q_\theta(s,a)}{\partial \theta_i} &= \frac{\partial(\sum_{j=1}^{n} \phi_j(s,a)\theta_j)}{\partial \theta_i} \\ &= \phi_i(s,a)\end{aligned}$$

- and so,

$$\Delta\theta_i = \alpha(r + \gamma Q_\theta(s', \pi(s')) - Q_\theta(s,a))\ \phi_i(s,a)$$

TD prediction algorithm for the linear case

Caution!

- No same guarantees that MC had when *bootstrapping* estimate of $Q(S_t, a)$ is used as the target
- Notice that TD targets are not independent of parameters. In TD(0):

$$r + \gamma \mathbf{Q}_\theta(s', \pi(s'))$$

  depends of θ
- Bootstrapping methods are not true gradient descent: they take into account the effect of changing θ on the estimate, **but ignore its effect on the target**. They include only a part of the gradient and, accordingly, we call them *semi-gradient methods.*

- However, it can be proved that **linear** TD(0) policy evaluation converges (close) to global optimum.

Subsection 2

Control algorithms for the linear case

TD(0) prediction algorithm for the linear case

- Like the Control methods we used in tabular learning algorithms, we will build algorithms that iterate the two following steps:
  1. Policy evaluation - Follow a method for approximate policy evaluation $Q_\theta \approx Q^\pi$
  2. Policy improvement - do policy improvement of the policy
- Depending on the Policy evaluation procedure used (MC, TD, etc.), we have a different method

Examples of Control using PI

Linear FA Monte Carlo

Initialize parameters θ as appropriate (e.g., $\theta = 0$)
repeat
  Generate trial using ϵ-greedy policy derived from Q_θ
  for each s_t in trial **do**
    $R_t \leftarrow$ long-term return from s_t
    $\theta \leftarrow \theta + \alpha(R_t - Q_\theta(s_t, a_t))\phi(s_t, a_t)$
  end for
until false

Examples of Control using PI

Linear FA Monte Carlo

Initialize parameters θ as appropriate (e.g., $\theta = 0$)
repeat
  Generate trial using ϵ-greedy policy derived from Q_θ
  for each s_t in trial **do**
    $R_t \leftarrow$ long-term return from s_t
    $\theta \leftarrow \theta + \alpha(R_t - Q_\theta(s_t, a_t))\phi(s_t, a_t)$
  end for
until false

Function $Q_\theta(s, a)$

Given θ, state s and action a
  return $\theta^T \phi(s, a)$

Select action following policy

Function $\pi_\theta(s)$

Given θ and state s

return $\arg\max_a \ \theta^T \phi(s, a)$

Function implementing ϵ-greedy

Given θ, $\epsilon \leq 1$ and state s

Select p number from uniform distribution in range $[0, 1]$
if $p \leq \epsilon$ **then**
$a \leftarrow$ Random action from $\mathcal{A}$
else
$a \leftarrow \pi_\theta(s)$
end if
return a

Examples of Control using PI

Linear FA Q-learning

initialize parameters θ arbitrarily (e.g. $\theta = 0$)
for each episode **do**
  Choose initial state s
  repeat
    Choose a from s using policy π_θ derived from Q_θ (e.g., ϵ-greedy)
    Execute action a, observe r, s'
    $\theta \leftarrow \theta + \alpha \left(r + \gamma Q_\theta(s', \pi_\theta(s')) - Q_\theta(s, a)\right) \phi(s, a)$
    $s \leftarrow s'$
  until s is terminal
end for

Convergence of Gradient methods

- We have examples of TD divergence even when exact solution is representable with linear function
- Fortunately, in practice, TD(0) works well... but we don't have guarantees
- Problem can be solved if we update parameters following an on-policy distribution (we have a proof of that). Good for Sarsa.
- Unfortunately convergence guarantees on TD incremental methods only work for linear approximation
- Main cause is that TD does not follow true gradient.

Deadly triad

- The risk of divergence arises whenever we combine three things:

  Function approximation: Significantly generalizing from large numbers of examples.

  Bootstrapping: Learning value estimates from other value estimates, as in dynamic programming and temporal-difference learning.

  Off-policy learning: Learning about a policy from data not due to that policy, as in Q-learning, where we learn about the greedy policy from data with a necessarily more exploratory policy.

- Any two without the third is ok.

Convergence of incremental Gradient methods for *control*

Algorithm	Table Lookup	Linear
Monte–Carlo Control	OK	OK
SARSA	OK	(OK)
Q–learning	OK	KO

(OK) = **chatters** around near–optimal value function

Conclusions and final notes about convergence

- Value-function approximation by stochastic gradient descent enables RL to be applied to arbitrarily large state spaces
- Most algorithms just carry over the Targets from the tabular case
- With bootstrapping (TD), we don't get true gradient descent methods
  - this complicates the analysis
  - but the linear, on-policy case is still guaranteed convergent
  - and learning is still much faster
- For continuous state spaces, coarse/tile coding is a good strategy

Conclusions and final notes about convergence

- Value-function approximation by stochastic gradient descent enables RL to be applied to arbitrarily large state spaces
- Most algorithms just carry over the Targets from the tabular case
- With bootstrapping (TD), we don't get true gradient descent methods
  - this complicates the analysis
  - but the linear, on-policy case is still guaranteed convergent
  - and learning is still much faster
- For continuous state spaces, coarse/tile coding is a good strategy
- Still some possible approaches: Gradient-TD (convergence in off-line linear FA) and Batch methods

Batch methods

Batch Reinforcement Learning

- Gradient descent is simple and appealing
  - It is computationally efficient (one update per sample)
  - ... But it is not sample efficient (does not take all profit from samples)
- We can do better at the cost of more computational time

Batch Reinforcement Learning

- Gradient descent is simple and appealing
  - It is computationally efficient (one update per sample)
  - ... But it is not sample efficient (does not take all profit from samples)
- We can do better at the cost of more computational time

- **Batch methods** seek to find the *best fitting value function* of given agent's experience ("training data") in a supervised way.

Fitted Q-learning: Non-linear approximation

Incremental Q-learning with FA

Q-learning with FA

initialize parameters θ arbitrarily (e.g. $\theta = 0$)
for each episode **do**
  Choose initial state s
  repeat
    Choose a from s using policy π_θ derived from Q_θ (e.g., ϵ-greedy)
    Execute action a, observe r, s'
    $Q_\theta(s,a) \leftarrow Q_\theta(s,a) + \alpha\,(r + \gamma Q_\theta(s', \pi_\theta(s')) - Q_\theta(s,a))\nabla_\theta Q_\theta(s,a))$
    $s \leftarrow s'$
  until s is terminal
end for

Problems with incremental Q-learning with FA

Essence of off-policy learning.

repeat

Choose a, execute it and observe r and s' (s, a, r, s') using any probabilistic policy

$Q_\theta(s,a) \leftarrow Q_\theta(s,a) + \alpha\left(r + \gamma Q_\theta(s', \pi_\theta(s')) - Q_\theta(s,a)\right)\nabla_\theta Q_\theta(s,a))$

$s \leftarrow s'$

until s is terminal

- Several problems with incremental off-policy TD learning
  - SGD does not converge because gradient does not follow true gradient
  - Target value is always changing and SGD does not converge
  - Data is not even close to iid (it is strongly correlated) so another problem for SGD convergence
- How to solve all these problems?

Generalizarion of off-policy learning

Let's generalize the method:

Generalizarion of off-policy learning.

Get $\mathcal{D} = \{\langle s, a, r, s' \rangle\}$ using any probabilistic policy
repeat
  Set $\mathcal{SD}$ to N samples randomly taken from $\mathcal{D}$
  for each sample i in $\mathcal{SD}$ **do**
    $y_i \leftarrow r + \gamma \max_a Q_\theta(s'_i a)$
  end for
  $\theta \leftarrow \arg\min_\theta \sum (Q_\theta(s_i, a_i) - y_i)^2$ *// Any ML regression method*
until convergence

Generalizarion of off-policy learning

- Notice several differences:
  1. Sample a set of N examples instead of only 1
  2. Don't use 1-step of gradient descent but compute exact solution (regression problem)

Generalizarion of off-policy learning

- Notice several differences:
  1. Sample a set of N examples instead of only 1
  2. Don't use 1-step of gradient descent but compute exact solution (regression problem)
- Each difference improves convergence
  1. Samples obtained randomly reduce correlation between them and stabilize Q value function for the regression learner
  2. Computation of exact solution avoid the true gradient problem

Fitted Q-learning

Fitted Q-learning

Given $\mathcal{D}$ of size T with examples $(s_t, a_t, r_{t+1}, s_{t+1})$, and regression algorithm, set N to zero and $Q_N(s, a) = 0$ for all a and s

repeat

$N \leftarrow N + 1$

Build training set $TS = \{\langle (s_t, a_t), r_{t+1} + \gamma \max_a Q_N(s_{t+1}, a)\rangle\}_{t=1}^{T}$

$Q_{N+1} \leftarrow$ regression algorithm on TS

until $Q_N \approx Q_{N+1}$ or $N >$ limit

return π based on greedy evaluation of Q_N

- Works specially well for forward Neural Networks as regressors (Neural Fitted Q-learning)