# Lecture 3 and 4: Dynamic programming and reinforcement learning

Reinforcement Learning and Decision Making MLSALT7, Lent 2016

Matthew W. Hoffman, Zoubin Ghahramani, Carl Edward Rasmussen

Department of Engineering
University of Cambridge

http://mlg.eng.cam.ac.uk/teaching/mlsalt7/

# Dynamic programming for control

# The Bellman operator

Recall in the last lecture that we wrote the value function recursively as

$$v^{\pi}(x) = r(x) + \gamma \int p^{\pi}(z|x)\, v^{\pi}(z)\, dz$$

which for discrete/tabular MDPs can be written as

$$\boldsymbol{v}^{\pi} = \mathbf{r} + \gamma \mathbf{P}^{\pi} \boldsymbol{v}^{\pi}.$$

Alternatively we can introduce the Bellman operator $\mathcal{T}^{\pi}$, the discrete version of which can be applied to any vector $\boldsymbol{v}$ resulting in

$$\mathcal{T}^{\pi} \boldsymbol{v} = \mathbf{r} + \gamma \mathbf{P}^{\pi} \boldsymbol{v}$$

for which $\boldsymbol{v}^{\pi} = \mathcal{T}^{\pi} \boldsymbol{v}^{\pi}$ is the unique fixed point.

# The Bellman operator is a $\gamma$-contraction

We can look at the max-norm $\|\boldsymbol{v}\|_\infty = \max_i v_i$ of the Bellman operator:

$$\begin{aligned}
\|\mathcal{T}^\pi \boldsymbol{v}_1 - \mathcal{T}^\pi \boldsymbol{v}_2\|_\infty &= \|\mathbf{r} + \gamma \mathbf{P}^\pi \boldsymbol{v}_1 - \mathbf{r} - \gamma \mathbf{P}^\pi \boldsymbol{v}_2\|_\infty \\
&= \gamma \|\mathbf{P}^\pi (\boldsymbol{v}_1 - \boldsymbol{v}_2)\|_\infty \\
&\leqslant \gamma \|\boldsymbol{v}_1 - \boldsymbol{v}_2\|_\infty
\end{aligned}$$

For $\gamma \in [0, 1)$ this shows that $\mathcal{T}^\pi$ is a contraction under this norm and as a result due to the Banach fixed-point theorem, $\mathcal{T}^\pi$ has a unique fixed-point.

This states that given any two vectors applying the operator to them will take them both closer towards something. That something can be found by repeatedly applying the contraction:

$$\boldsymbol{v}^{i+1} = \mathcal{T}^\pi \boldsymbol{v}^i$$

which we have defined as the value function!

# Convergence of the Bellman operator cont'd

In slightly more detail we can look at the max-norm comparing the iterative procedure and the fixed-point:

$$
\begin{aligned}
\|\boldsymbol{v}^{i+1} - \boldsymbol{v}^{\pi}\|_\infty &= \|\mathcal{T}^{\pi}\boldsymbol{v}^{i} - \mathcal{T}^{\pi}\boldsymbol{v}^{\pi}\|_\infty && \text{(the def'n of } \mathcal{T}^{\pi} \text{ and } \boldsymbol{v}^{\pi}) \\
&\leqslant \gamma\|\boldsymbol{v}^{i} - \boldsymbol{v}^{\pi}\|_\infty && \text{(the } \gamma\text{-contraction)} \\
&\vdots \\
&\leqslant \gamma^{i+1}\|\boldsymbol{v}^{0} - \boldsymbol{v}^{\pi}\|_\infty \to 0
\end{aligned}
$$

# Acting according to a value function

Let's say we have an arbitrary value function $\boldsymbol{v}$. How should we act optimally under this value function?

We can define a policy:

$$\pi'(x) = \arg\max_a \left[ r_x + \gamma \mathbf{P}_{ax}^\top \boldsymbol{v} \right]$$

If $\boldsymbol{v} = \boldsymbol{v}^\pi$ is value function associated with policy $\pi$, do $\pi$ and $\pi'$ coincide?

No! The equation above looks similar to the Bellman operator introduced earlier, but because of the arg max above, $v_x^{\pi'} \geqslant v_x^\pi$ for all $x \in \mathcal{X}$.

# The Optimal Bellman operator and value iteration

Define the optimal Bellman operator $\mathcal{T}^*$ as

$$(\mathcal{T}^* \boldsymbol{v})_x = \max_a \left[ r_x + \gamma \mathbf{P}_{ax}^\top \boldsymbol{v} \right].$$

This can similarly be shown to be a $\gamma$-contraction which has as its fixed-point the optimal value function $\boldsymbol{v}^* = \mathcal{T}^* \boldsymbol{v}^*$.

The iterative procedure:

1. start from some arbitrary $\boldsymbol{v}$
2. and iterate $\boldsymbol{v} \leftarrow \mathcal{T}^* \boldsymbol{v}$ until convergence

is known as value iteration, due to Bellman (1957). Convergence can be shown in exactly the same way as the convergence for $\mathcal{T}^\pi$

What does the policy look like in this case? How would things change if the reward was defined as $r_{axz}$?

# Backward induction

Value iteration is also known as backward induction. Why?

Let's assume that rather than considering an infinite-horizon problem we want to solve

$$a_0 = \arg\max_a \mathbb{E}_{a_{1:T}, x_{1:T}} \Big[ \sum_{t=1}^{T} r(x_t) \Big| x_0 \Big]$$

How could we use backward induction to solve this?

The key is in the name. We can start by solving the problem optimally at step $T$:

$$v_T(x) = \max_a \left[ r_x + \mathbf{P}_{ax}^\top \mathbf{r} \right]$$

$$v_{T-1}(x) = \max_a \left[ r_x + \mathbf{P}_{ax}^\top \boldsymbol{v}_T \right]$$

and performing induction backward in time...

until ultimately $a_0 = \arg\max \boldsymbol{v}_1$. This can be written recursively as $\boldsymbol{v}_t = \mathcal{T}^* \boldsymbol{v}_{t+1}$ where $\boldsymbol{v}_{T+1} = \mathbf{r}$.

# Policy evaluation and improvement

We have already shown how to compute the value of a policy by finding the fixed-point $\boldsymbol{v}^\pi = \mathcal{T}^\pi \boldsymbol{v}^\pi$. This is known as policy evaluation.

But now given the value of a policy we an improve upon that value (as shown in an earlier slide) by setting

$$\pi(x) \leftarrow \arg\max_a \Big[ \underbrace{r_x + \mathbf{P}_{ax}^\top \boldsymbol{v}^\pi}_{Q^\pi(x,a)} \Big]$$

This is known as policy improvement. The function $Q^\pi$ is often known as a state/action value-function and is not strictly necessary here since it follows directly from $v^\pi$.

After we have evaluated and updated the policy $\pi$ how can we use this to find the optimal policy?

# Policy iteration

This interleaved process of policy evaluation and improvement is known as policy iteration:

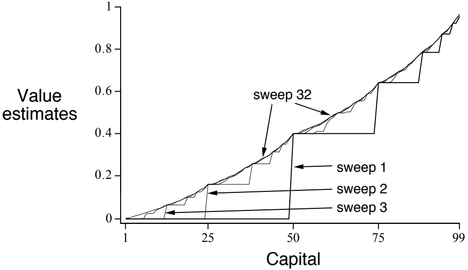
1. Initialize $\boldsymbol{v}$ arbitrarily
2. Iterate:
   1. $\pi(x) \leftarrow \arg\max_a \left[r_x + \mathbf{P}_{ax}^\top \boldsymbol{v}\right]$
   2. $\boldsymbol{v} \leftarrow \lim_{n\to\infty} (\mathcal{T}^\pi)^n \boldsymbol{v}$

# Generalized policy iteration

item most of these techniques can be gathered as generalized policy iteration where

- some improvement is made to the value function $\boldsymbol{v}$, and
- the policy $\pi$ is updated to be greedy with respect to $\boldsymbol{v}$

One particular instance is where the evaluation step is ended early; succesive evaluation steps may not change the policy.



# Reinforcement learning

# RL vs DP

The approaches introduced previously, based on dynamic programming, relied on knowing the model $p(z|x, a)$ in order to perform the necessary integrals.

- now we will only assume that we can sample $z$ conditioned on $(x, a)$
- further we will only assume we can sample a single $z$!

# The temporal difference (TD) error

Let's return to the recursive definition of the value function:

$$v^{\pi}(x) = r(x) + \gamma \sum_{z} p^{\pi}(z|x)\, v^{\pi}(z)$$

if we can sample from $p^{\pi}(z|x)$ then this can be approximated as

$$\approx r(x) + \gamma \frac{1}{N} \sum_{i} v^{\pi}(z^{i}) \quad \text{for } z^{i} \sim p^{\pi}(\cdot|x)$$

but we will only assume we can take a single sample. IE we have no "reset switch" that will allow us to draw multiple samples from a single $x$. Once we've taken an action and moved to $z$ that's it.

$$\approx r(x) + \gamma v^{\pi}(z) \quad \text{for } z \sim p^{\pi}(\cdot|x)$$

The difference between the left- and right-side is known as the temporal difference error. It should be zero in expectation!

# TD for policy evaluation

Temporal difference methods use the TD-error (essentially) as a noisy gradient (think stochastic approximation)

Given a policy $\pi$ and an arbitrary value function $\boldsymbol{v}$ we can compute $\boldsymbol{v}^{\pi}$ by iterating:

1. sample the next state $z \sim p^{\pi}(\cdot|x)$
2. update the value function,

$$v_x \leftarrow v_x + \alpha\Big[\underbrace{r_x + \gamma v_z}_{\text{1-step Monte Carlo approx}} - \overbrace{v_x}^{\text{current prediction}}\Big]$$

3. and repeat: $x \leftarrow z$

But this is just policy evaluation. Can we combine this with policy improvement?

# TD for control

In order to apply TD for control we first have to note that we must directly learn Q rather than a value function. Why?

For DP-based methods policy improvement is

$$\pi(x) = \arg\max_a Q(x, a) = \arg\max_a r_x + \mathbf{P}_{ax}^\top \boldsymbol{v}$$

only $v$ is needed because integration with respect to $p^\pi(z|x)$ can be performed. This doesn't work for RL.

Finally: the integration above also allows us to consider all outcomes of taking action $a$. For RL we must instead explore by injecting noise into our action selection.

# On-policy TD: SARSA

1. initialize $Q(x, a)$ arbitrarily
2. select action $a$ arbitrarily
3. iterate
   1. sample $x' \sim p(\cdot|x, a)$
   2. choose action $a'$ from Q "$\epsilon$-greedily",

$$a' = \begin{cases} \arg\max_{a'} Q(x', a') & \text{w.p. } 1 - \epsilon, \\ \text{Uniform}(\mathcal{A}) & \text{otherwise.} \end{cases}$$

   3. update the value function

$$Q(x, a) \leftarrow Q(x, a) + \alpha \Big[ r_x + \gamma \underbrace{Q(x', a')}_{\text{"on-policy" because } a' \text{ used here}} - Q(x, a) \Big]$$

   4. $x \leftarrow x'$; $a \leftarrow a'$

Note: the funny name stands for "State-Action-Reward-State-Action"

# Off-policy TD: Q-learning

1. initialize $Q(x, a)$ arbitrarily
2. select action $a$ arbitrarily
3. iterate
    1. sample $x' \sim p(\cdot|x, a)$
    2. choose action $a'$ from Q "$\epsilon$-greedily",
    3. update the value function

$$Q(x, a) \leftarrow Q(x, a) + \alpha\big[r_x + \gamma \underbrace{\max_{a''} Q(x', a'')}_{\text{"off-policy" due to } a''} - Q(x, a)\big]$$

    4. $x \leftarrow x'$; $a \leftarrow a'$

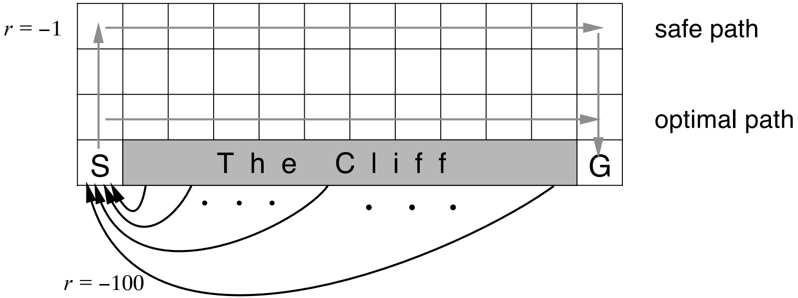
# On- vs off-policy

What is the difference between on- and off-policy methods?

- updates are performed without using actions selected by the exploratory policy for off-policy
- Q-learning can not care about rewards it gets while it's learning
    - we WILL choose bad actions $a'$ due to the noisy exploration
    - but this does not affect the value function learned by Q-learning due to the max

# Cliff-world

no noise, but the optimal path passes by a cliff with high penalty for falling



# Cliff-world results