50.021 – AI

Alex

Week 06: Q-learning I

Key takeaways:

Be able to explain the main ideas behind:

- explain basic Q-learning
- explain SARSA
- explain the difference between SARSA and Q-learning
- name three tricks used in the nature paper for atari games
- explain the idea behind n-step bootstrap methods

1 Q-function

idea of Q-function

Idea: $Q^{\pi}(s, a)$ is the expected future reward when using action a in state s (not following any policy!), and after that continuing with policy π. This means as a definition:

The Bellman equations for the Q-function:

deterministic policy:

$$\begin{aligned} Q^\pi(s,a) &= \sum_{s'} P(s'|s,a)R(s,a,s') + \gamma \sum_{s'} P(s'|s,a)V^\pi(s') \\ &= \sum_{s'} P(s'|s,a)R(s,a,s') + \gamma \sum_{s'} P(s'|s,a)Q^\pi(s', a' = \pi(s')) \end{aligned}$$

stochastic policy:

$$\begin{aligned} Q^\pi(s,a) &= \sum_{s'} P(s'|s,a)R(s,a,s') + \gamma \sum_{s'} P(s'|s,a)V^\pi(s') \\ &= \sum_{s'} P(s'|s,a)R(s,a,s') + \gamma \sum_{s'} P(s'|s,a) \sum_{a'} \pi(a'|s')Q^\pi(s',a') \end{aligned}$$

The Bellman optimality for the Q-function and the optimal policy π^*:

$$\begin{aligned} Q^{\pi^*}(s,a) &= \sum_{s'} P(s'|s,a)R(s,a,s') + \gamma \sum_{s'} P(s'|s,a)V^{\pi^*}(s') \\ &= \sum_{s'} P(s'|s,a)R(s,a,s') + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q^{\pi^*}(s',a') \\ V^{\pi^*}(s) &= \max_a Q^{\pi^*}(s,a) \end{aligned}$$

The difference is the same as for values: for the optimal policy you use $\max_a T(a)$, for a given policy use $T(a = \pi(s))$ or $\sum_a T(a)\pi(a|s)$
Q-iteration: Computes the Q-function under the optimal policy π^*.

$$\begin{aligned} Q_{k+1}(s,a) &= \sum_{s'} P(s'|s,a)R(s,a,s') + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q_k(s',a') \\ V^{\pi^*}(s) &= \max_a Q^{\pi^*}(s,a) \\ \pi^*(s) &= \operatorname{argmax}_a Q^{\pi^*}(s,a) \end{aligned}$$

2 the start of Q-learning

Goal: Learn Q-function $Q^{\pi^*}(s,a)$ for optimal policy π^*

Remember:

$$Q^{\pi^*}(s,a) = \sum_{s'} P(s'|s,a)R(s,a,s') + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q^{\pi^*}(s',a')$$

This translates to:

Q = expected reward + $\gamma\times$ Q for new states s' averaged with probability to land in s'.

What is when we have observed an experience (s, a, r, s') and want to learn from it?

Note: $a \sim \pi(a|s)$, $s' \sim P(s'|a, s)$, $r = R(s, a, s')$

We can approximate expected reward by r:

$$\sum_{s'} P(s'|s,a)R(s,a,s') \approx r$$

We can replace *Q for new states s' averaged ...* by Q in our observed state s':

$$\gamma \sum_{s'} P(s'|s,a) \max_{a'} Q^{\pi^*}(s',a') \approx \gamma \max_{a'} Q_{\text{current}}(s',a'), \text{ so:}$$

$$Q^{\pi^*}(s,a) = \sum_{s'} P(s'|s,a)R(s,a,s') + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q^{\pi^*}(s',a')$$

$$Q_{\text{current}}(s,a) \approx r + \gamma \max_{a'} Q_{\text{current}}(s',a')$$

in an iterative fashion:

$$s, a \rightsquigarrow r, s'$$

$$Q_{\text{new}}(s,a) \approx r + \gamma \max_{a'} Q_{\text{old}}(s',a')$$

However we want a slow update, so weight it with α

$$\begin{aligned} Q_{\text{new}}(s,a) &= (1-\alpha)Q_{\text{old}}(s,a) + \alpha \left(r + \gamma \max_{a'} Q_{\text{old}}(s',a')\right) \\ &= Q_{\text{old}}(s,a) + \alpha \left(r + \gamma \max_{a'} Q_{\text{old}}(s',a') - Q_{\text{old}}(s,a)\right) \end{aligned}$$

What is the idea?

$$r + \gamma \max_{a'} Q_{\text{old}}(s',a') > Q_{\text{old}}(s,a) \Rightarrow \text{ increase } Q_{\text{old}}(s,a)$$

$$r + \gamma \max_{a'} Q_{\text{old}}(s',a') < Q_{\text{old}}(s,a) \Rightarrow \text{ decrease } Q_{\text{old}}(s,a)$$

Compare observed reward r plus $\gamma\times$ your estimated Q from the observed new state s' (from experience (s, a, r, s')) against you your estimated $Q(s, a)$ from the observed old state s and update iteratively.

Reinforcement:

- more reward then currently estimated $\Rightarrow$ increase $Q_{\text{old}}(s, a)$
- less reward then currently estimated $\Rightarrow$ decrease $Q_{\text{old}}(s, a)$

Analogously can be executed for the value function.

Q-Learning by TD(0)-Learning for the Q-function

$$\begin{aligned} Q_{\text{new}}(s,a) &= (1-\alpha)Q_{\text{old}}(s,a) + \alpha\left(r + \gamma \max_{a'} Q_{\text{old}}(s',a')\right) \\ &= Q_{\text{old}}(s,a) + \alpha\left(r + \gamma \max_{a'} Q_{\text{old}}(s',a') - Q_{\text{old}}(s,a)\right) \end{aligned}$$

As algorithm?

- init $Q(s,a) = 0$, choose start state s
- run while loop:
  - choose action $a \approx_\epsilon \text{argmax}_a Q(s,a)$
  - observe reward r and new state s' to obtain (s,a,r,s')
  - update $Q(s,a) = Q_{\text{old}}(s,a) + \alpha\,(r + \gamma \max_{a'} Q_{\text{old}}(s',a') - Q_{\text{old}}(s,a))$
  - set oldstate to new state $s = s'$

$a \approx_\epsilon \text{argmax}_a Q(s,a)$ is what ? – so called ϵ-greedy exploration

$$a = \begin{cases} \text{random} & \text{with prob } \epsilon \\ \text{argmax}_a Q(s,a) & \text{else} \end{cases}$$

Idea: Do not follow strictly your current estimate of best action. Allow a random action with some probability to explore new options.

Limitation: above works for discrete states s. Not if states are continuous.

Q-learning as above finds the Q^{π^*} for the optimal policy π^*

3 SARSA: Q-function estimation for a given policy

above: Q^{π^*}. Here goal Q^π for a given policy π.

Remember:

$$Q^\pi(s,a) = \sum_{s'} P(s'|s,a)R(s,a,s') + \gamma \sum_{s'} P(s'|s,a)Q^\pi(s', a' = \pi(s'))$$

When we have an experience s,a,r,s', then $s' \sim P(s'|s,a)$. If we assume also that $a' = \pi(s')$, then:

$$\sum_{s'} P(s'|s,a)R(s,a,s') \approx r$$

$$\gamma \sum_{s'} P(s'|s,a)Q^\pi(s', a' = \pi(s')) \approx \gamma Q^\pi(s',a')$$

Therefore we can use $r + \gamma Q^\pi(s', a')$ to update $Q^\pi(s, a)$ softly:

$$\begin{aligned} Q^\pi_{\text{new}}(s,a) &= (1-\alpha)Q^\pi_{\text{old}}(s,a) + \alpha(r + \gamma Q^\pi_{\text{old}}(s',a')) \\ &= Q^\pi_{\text{old}}(s,a) + (r + \gamma Q^\pi_{\text{old}}(s',a') - Q^\pi_{\text{old}}(s,a)) \end{aligned}$$

The difference between Q-learning and SARSA is: SARSA is a so-called on-policy method.

on-policy is defined as computing quantities for the given policy π. While off policy denotes the optimization for a different policy (usually some greedy policy or an ϵ-greedy policy.) There are two variants: SARSA for policy evaluation and SARSA for policy control (= learning the optimal policy π^*).

SARSA for policy evaluation with a given policy π

goal Q^π for a given policy π.

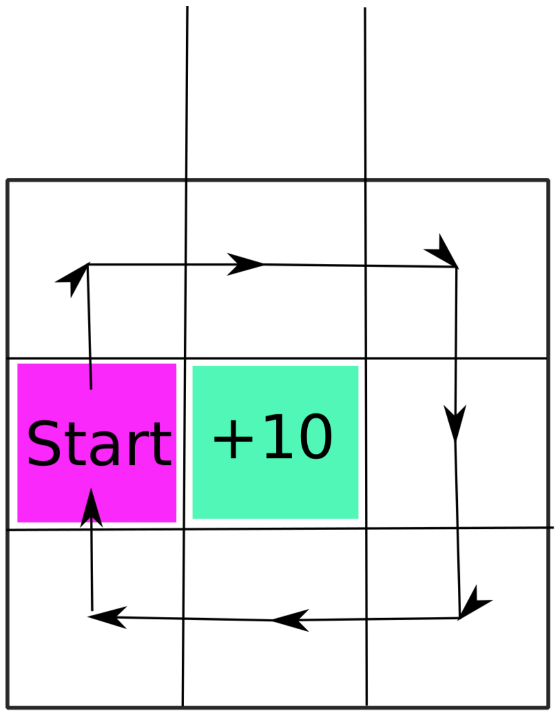
Given an experience $s, a = \pi(s), r, s', a' = \pi(s')$,
initialize $\forall(s,a)\ Q^\pi_{\text{new}}(s,a) = 0$, then iterate until convergence:

- update Q

$$Q^\pi_{\text{new}}(s,a) = Q^\pi_{\text{old}}(s,a) + (r + \gamma Q^\pi_{\text{old}}(s',a') - Q^\pi_{\text{old}}(s,a))$$

- update $s = s'$, $a = a'$

3.1 When one can use SARSA to optimize for the optimal policy?



SARSA does not try to learn the optimal policy if you choose $a \sim \pi(a|s)$ for a fixed π!

How to learn the optimal policy with sarsa? One needs to combine sarsa with iteratively changing the policy towards the greedy policy. This can be done by replacing $a \sim \pi(a|s)$ by doing in every step:

$$\begin{aligned} a &\approx_\epsilon \text{argmax}_a Q_t(s,a) \text{ and} \\ a' &\approx_\epsilon \text{argmax}_a Q_t(s',a) \end{aligned}$$

When you combine SARSA with choosing $a \approx_\epsilon \text{argmax}_a Q_t(s,a)$ and $a' \approx_\epsilon \text{argmax}_a Q_t(s',a)$, then it will converge to the optimal policy provided that $\epsilon = \epsilon(t) \xrightarrow{t} 0$ and that all state-action-pairs are visited an infinite number of times.

SARSA for policy control – obtaining π^*

goal Q^{π^*} for the optimal policy π^*.

Given an experience $s, a \approx_\epsilon \text{argmax}_a Q_t(s,a), r, s', a' \approx_\epsilon \text{argmax}_a Q_t(s', a')$,
initialize $\forall (s,a)\ Q^\pi_{\text{new}}(s,a) = 0$, then iterate until convergence:

- update Q

$$Q^\pi_{\text{new}}(s,a) = Q^\pi_{\text{old}}(s,a) + (r + \gamma Q^\pi_{\text{old}}(s',a') - Q^\pi_{\text{old}}(s,a))$$

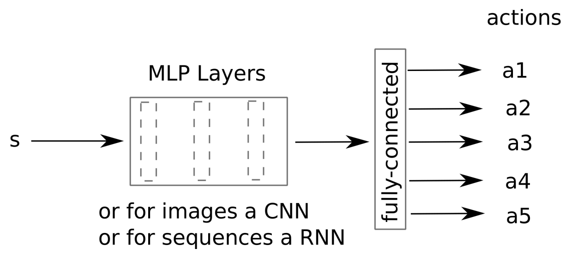
- update $s = s'$, $a = a'$

See algorithm on page 106 for the version of sarsa which converges to the optimal policy.

3.2 When SARSA obtains a different result from Q-Learning while using ϵ-greedy policy?

The famous example is the cliffworld on page 108. Q-learning assumes using the optimal policy. Whenever the ϵ-greedy policy has significant risks, and thus is a lot different from following the optimal policy, then SARSA will give a different solution.

4 Towards Deep Q-learning for continuous states with neural nets and the like



How to optimize in that case? We need a loss to compute a gradient.
Can start from:

$$\begin{aligned} s, a &\rightsquigarrow r, s' \\ Q_{\text{new}}(s,a) &\approx r + \gamma \max_{a'} Q_{\text{old}}(s', a') \\ \Rightarrow 0 &\approx r + \gamma \max_{a'} Q_{\text{old}}(s',a') - Q_{\text{new}}(s,a) \end{aligned}$$

This looks like a regression problem, and it can be approached (this is a simplification by)

$$L((s,a,r,s')) = (r + \gamma \max_{a'} Q_w(s',a') - Q_w(s,a))^2$$

as a loss for a single experience (s, a, r, s'). Now can do minibatch-gradient and SGD training or anything similar.

Mnih et al. Nature paper `https://www.nature.com/articles/nature14236` has mnay modifications – all for the goal to make learning more stable.
Please take a look at this paper.

4.1 Fix Q-Target

Consider:

$$L((s, a, r, s')) = (r + \gamma \max_{a'} Q_w(s', a') - Q_w(s, a))^2$$

Using the same Q_w for computing the expected reward and for the current $Q_w(s, a)$ was found to be unstable, a deeper look shows that one should use two Q_w functions: Q_{old} – the target Q, and Q_{new} the Q for optimization

$$0 \approx (r + \gamma \max_{a'} Q_{\text{old}}(s', a') - Q_{\text{new}}(s, a))^2$$

The first change is to consider

$$w = \text{argmin}_w (r + \gamma \max_{a'} Q_{\text{target}}(s', a') - Q_w(s, a))^2$$

where we keep Q_{target} constant for K episodes. Only every K episodes we update it by

$$Q_{\text{target}} = Q_w \text{ if } n_episode\%K == 0$$

We fix the target for a while to avoid the situation that rapid changes in Q_w (due to the SGD Gradient updates) make the learning unstable.

There is an alternative to that which also often works is to run at every iteration for a small $\beta \approx 0$:

$$Q_{\text{target}} = (1 - \beta) * Q_{\text{target}} + \beta Q_w$$

which updates the target only a bit every episode.

4.2 Robust losses

The Huber loss:

$$l(y, z) = 0.5(y - z)^2 1[|y - z| < 1] + |y - z| 1[|y - z| \geq 1]$$

has smaller gradients for large deviations than the MSE loss.

Also: clip gradients to lie in $[-1, +1]$

4.3 Experience Replay

Q-learning is an off-policy methods: it optimizes for a different policy than the one used to generate samples.
As such it is possible to generate a stack of N experiences (s, a, r, s'). For most games, they use $N = 10^6$. At training time one does the following:

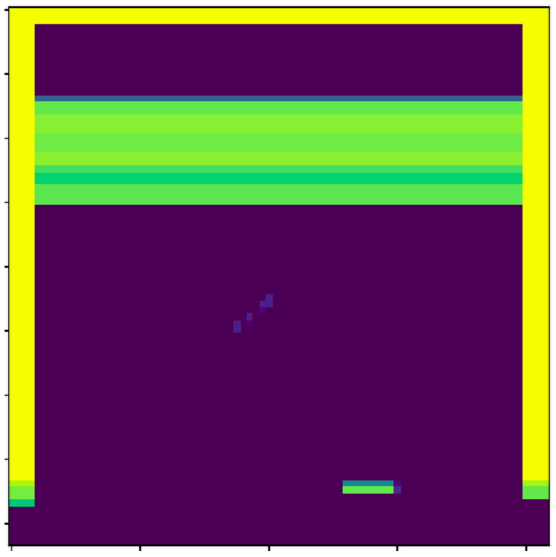
- play an episode, insert all experiences (s, a, r, s') into the memory
- randomly draw one (s, a, r, s') for gradient descent

4.4 Image preprocessing

the Atari output has shape $(3, 210, 160)$ per frame. They do the following:

- covert to gray, reshape to 84 size. Results in $(1, 84, 84)$

This is not sufficient.
Trick is: collect every fourth frame one frame, and assemble 4 of this into one input of shape $(4, 84, 84)$ – so in total collect 4 frames out of 16 into one single input. Guess why not using a single frame?



4.5 use a CNN for Q

Standard CNN architecure to process $(4, 84, 84)$ inputs.

5 n-step bootstrap methods

Idea:

$$Q^\pi(s,a) = \sum_{s'} P(s'|s,a)R(s,a,s') + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q^\pi(s',a')$$

is approximated by

$$Q_{\text{new}}(s,a) \approx r + \gamma \max_{a'} Q_{\text{old}}(s',a')$$

In this approximation we take the reward of one step r and an estimate for the next step. In the beginning of the learning, the estimate $Q_{\text{old}}(s',a')$ can be very poor, this will inject noise into the update.

Key idea: use more observed rewards before resorting to estimated Q_{old}!

Suppose one does in state s action a, and then in state s' action a', and then continues according to the best Q in state s'' ?

$$\begin{aligned} Q^{\pi}(s,a) &= \sum_{s'} P(s'|s,a)R(s,a,s') + \gamma \sum_{s'} P(s'|s,a)Q^{\pi}(s',a') \\ &= \sum_{s'} P(s'|s,a)R(s,a,s') + \gamma \sum_{s'} P(s''|s',a')R(s',a',s'') + \gamma^2 \sum_{s'} P(s''|s',a')R(s',a',s'') \max_{a''} Q^{\pi}(s'',a'') \end{aligned}$$

One can do that for three steps:

$$\begin{aligned} Q^{\pi}(s,a) &= \sum_{s'} P(s'|s,a)R(s,a,s') + \gamma \sum_{s'} P(s'|s,a)Q^{\pi}(s',a') \\ &= \sum_{s'} P(s'|s,a)R(s,a,s') + \gamma \sum_{s'} P(s''|s',a')R(s',a',s'') + \gamma^2 \sum_{s'} P(s''|s',a') \max_{a''} Q^{\pi}(s'',a'') \\ &= \sum_{s'} P(s'|s,a)R(s,a,s') + \gamma \sum_{s'} P(s''|s',a')R(s',a',s'') + \gamma^2 \sum_{s''} P(s'''|s'',a'')R(s'',a'',s''') + \gamma^3 \sum_{s''} P(s'''|s'',a'') \max_{a'''} Q^{\pi}(s''',a''') \end{aligned}$$

The structure is

Q = reward(step 1) $+\gamma^1$ reward(step 2) $+\gamma^2$ reward(step 3) $+\gamma^3 \sum_{s''} P(s'''|s'',a'') \max_{a'''} Q^{\pi}(s''',a''')$

Suppose one observes a chain: $s_0, a_0, r_0, s_1, a_1, r_1, s_2, a_2, r_2, s_3$

This could be approximated by

$$\begin{aligned} Q_{\text{new}}(s_0,a_0) &= (1-\alpha)Q_{\text{old}}(s_0,a_0) + \alpha \left(r_0 + \gamma^1 \max_{a_1} Q_{\text{old}}(s_1,a_1) \right) \\ &= (1-\alpha)Q_{\text{old}}(s_0,a_0) + \alpha \left(r_0 + \gamma^1 r_1 + \gamma^2 \max_{a_2} Q_{\text{old}}(s_2,a_2) \right) \\ &= (1-\alpha)Q_{\text{old}}(s_0,a_0) + \alpha \left(r_0 + \gamma^1 r_1 + \gamma^2 r_2 + \gamma^3 \max_{a_3} Q_{\text{old}}(s_3,a_3) \right) \end{aligned}$$

This is the basis for so called n-step algorithms such as n-step SARSA.

Advantage: use more real experiences and less to be learned estimates Q.
For an algorithm: see algorithm on page 120

6 n-step bootstrap methods for off-policy learning

This is off graded knowledge. Useful when using replay memory or ϵ-greedy policies for sampling.

see page 121 and algorithm on page 122. This is for the case when the policy used for sampling (e.g. from a Q from a past iteration, as when using reply memory!) differs from the policy π currently being optimized ... which is usually $\pi(s) = \text{argmax}_a Q(s, a)$ for the current step.