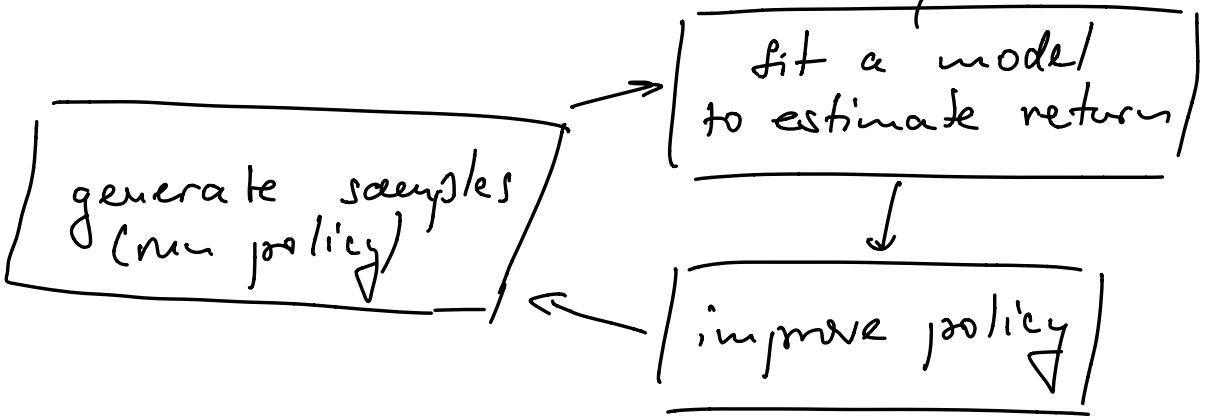
Value Function Methods

- PG methods suffer from high variance when estimating $\nabla_\theta J(\theta)$
- AC methods introduce fitted value iteration to reduce the variance in the MC estimate of $\nabla_\theta J(\theta)$

- Why do we need the actor π_θ?
- can we get rid of it?
- anatomy of RL algorithms with value function approx



e.g. use a (ε-) greedy policy:

$$\pi \leftarrow \pi' = \begin{cases} 1, & \text{if } a = \arg\max_A Q^\pi_\phi \\ 0, & \text{otherwise} \end{cases}$$

- cannot fit V_φ^π, since we can't improve the policy online

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

=> fit the Q function $\rightarrow Q_\varphi^\pi$

- Algorithm: Fitted Q-iteration

1. collect data set $\{(s_i, a_i, r_i, s_i')\}_{i=1}^N$ (repeat S times)
   using some policy
   (Q-learning is off-policy)

2. compute target:
   $y_i = r(s_i,a_i) + \gamma \max_{a' \in A(s')} Q_\varphi^\pi(s',a')$

3. do "supervised" regression
   $\varphi \leftarrow \arg\min_\varphi \frac{1}{2} \sum_i \| Q_\varphi^\pi(s_i,a_i) - y_i \|^2$

   ↑ assumed const. in φ
   → not ordinary GD

(2 K)

- Q-learning is off-policy
  → we are not learning trajectories
  but every transition $s \to s'_i$ is indep.
  taken into account

issue: it doesn't work!

- Algorithm: online Q-learning (watkins Q-learning)

1. take action a_j using some policy
   & observe (s_j, a_j, r_j, s'_j): single transition

2. $y_j = r(s_j, a_j) + \gamma \max_{a'_j} Q_\varphi(s'_j, a'_j)$

3. $\varphi \leftarrow \operatorname{argmin}_\varphi \frac{1}{2} \| Q_\varphi(s_j, a_j) - y_j \|^2$

no sum $\sum_j$!

note: $\operatorname{argmin}_\varphi \frac{1}{2} \| Q_\varphi(s_j, a_j) - y_j \|^2$

$\stackrel{?}{=} \nabla_\varphi \frac{1}{2} \| Q_\varphi(s_j, a_j) - y_j \|^2 \stackrel{!}{=} 0$

$\Rightarrow \left[\nabla_\varphi Q_\varphi(s_j, a_j)\right] \times \left(Q_\varphi(s_j, a_j) - y_j\right) = 0$

↑ y_j is treated as φ-indep!

- problems with online Q-learning:

1) regress on running target y_j

$\arg\min_\varphi \| Q_\varphi(s_j, a_j) - y_j \|^2$ → looks like a regression problem (?)

- is Q-learning gradient descent?

NO! b/c y_j is treated to be indep of φ

in Q-learning y_j is a moving target

(distribution which generates the data (i.e. policy) changes over time)

→ Q-learning with non-linear function approximation (e.g. DNN w/ nonlinear activations) does not converge.

- the absence of a proof for convergence does not mean that in a given instance Q-learning cannot converge

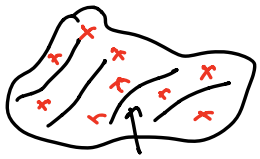
2) samples drawn along a trajectory are correlated

- single-sample minibatches

(recall MC sampling: autocorrelation time)

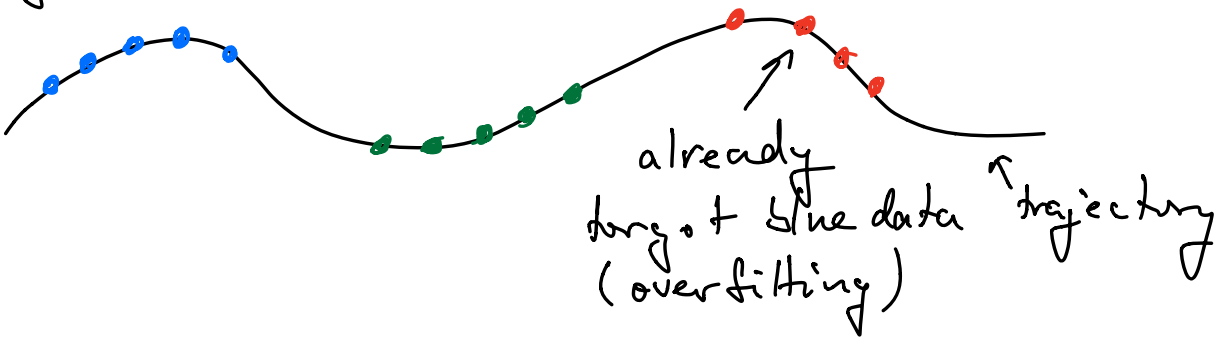
→ overfitting ?

→ cannot sample well the space of transitions :



→ need iid samples for SGD to work

e.g.

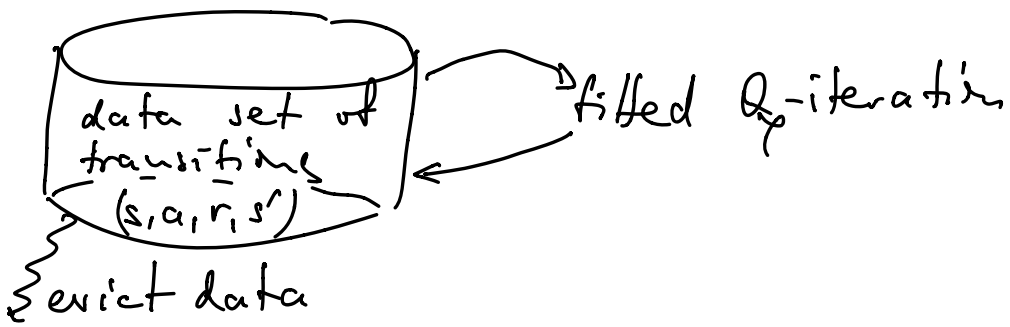


- replay buffers :

→ at step 1 : use any policy (off-policy)

- can re-use older data
- need policy w/ broad support

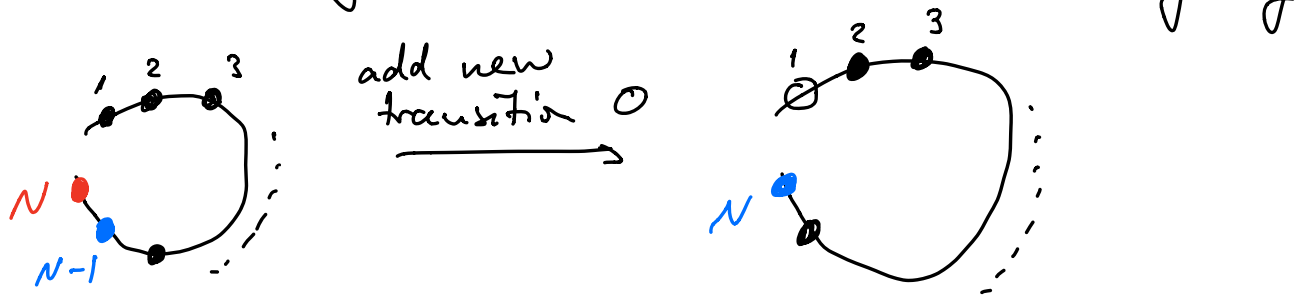
idea : Buffer



- load a small random batch from the buffer uniformly
  → remove correlations btw. transitions/data points

→ buffer needs to be initialized
  - eg, use a random policy (or if you have a guess for a good policy)

→ update buffer periodically:
  e.g. ring buffer: discard oldest data chronologically



→ new transitions sampled using, e.g. ε-greedy policy w.r.t. Q_θ

note:
- we don't immediately train on the most recent data point (maybe we never use it for training)
- transitions sampled i.i.d from buffer

→ Q-iteration w/ replay buffer:

1. collect data set $\{(s_j, a_j, r_j, s'_j)\}_j$ using some policy; add it to buffer B
2. sample a batch from B uniformly
3. $\varphi \leftarrow \varphi - \alpha \sum_j \nabla_\varphi Q_\varphi(s_j, a_j) \times$

$$\times \Big\{ Q_\varphi(s_j, a_j) - \overbrace{\big[r(s_j, a_j) + \gamma \max_{a'_j} Q_\varphi(s'_j, a'_j) \big]}^{= y_j} \Big\}$$

φ: no gradient pushed here

- target Q-network

still: moving target problem

but:

2. set $y_j = r(s_j, a_j) + \gamma \max_{a'_j} Q_\varphi(s'_j, a'_j)$
3. $\varphi \leftarrow \arg\min_\varphi \frac{1}{2} \sum_j \| Q_\varphi(s_j, a_j) - y_j \|^2$

well-posed regression problem ↓

SGD works ✓

y_j: treat as a label, i.e. indep. of φ ↑

idea: use two Q-networks

i) Q_φ : network to update in step 3

ii) $Q_{\varphi'}$: target network : hold fixed in step 3

step 3:

$\varphi \leftarrow \arg\min_{\varphi} \frac{1}{2} \sum_j \| Q_\varphi(s_j, a_j) -$ (Q_φ net, keep updating in step 3)

$- [r(s_j, a_j) + \gamma \max_{a'_j} Q_{\varphi'}(s'_j, a'_j)] \|^2$

($Q_{\varphi'}$: target network, keep fixed)

- steps 2 & 3 together : regular regression:

$Q_\varphi \longrightarrow Q_{\varphi'}$

makes Q_φ similar to the target net $Q_{\varphi'}$

- how do we choose $Q_{\varphi'}$?

  -> use the old (φ parameters) (from previous iteration step)

  physics intuition: "adiabatic learning"

  -> how often do we update $Q_{\varphi'}$?

  - new time scale

1. save target $Q_{\varphi'}$-net params $\varphi' \leftarrow \varphi$

2. collect dataset $\{(s_j, a_j, r_j, s'_j)\}$ using some policy; add it to buffer B;

xN

3. sample a batch from B uniformly

Kx

4. update Q_φ-network params

$$\varphi \leftarrow \varphi - \alpha \nabla_\varphi \frac{1}{2} \sum_j \| Q_\varphi(s_j, a_j) - [r(s_j, a_j) + \gamma \max_{a'_j} Q_{\varphi'}(s'_j, a'_j)] \|^2$$

- inner loop of steps 3 & 4 regresses on stationary targets y_j

— algorithm: DQN:

(Q-learning with a replay buffer & target Q-net)

1. take a_j using some policy, e.g. ε-greedy wrt Q_φ observe (s_j, a_j, r_j, s'_j); add it to buffer B.

2. sample minibatch $\{(s_j, a_j, r_j, s'_j)\}_j$ from B uniformly

3. compute: $y_j = r(s_j, a_j) + \gamma \max_{a'} Q_{\varphi'}(s'_j, a'_j)$

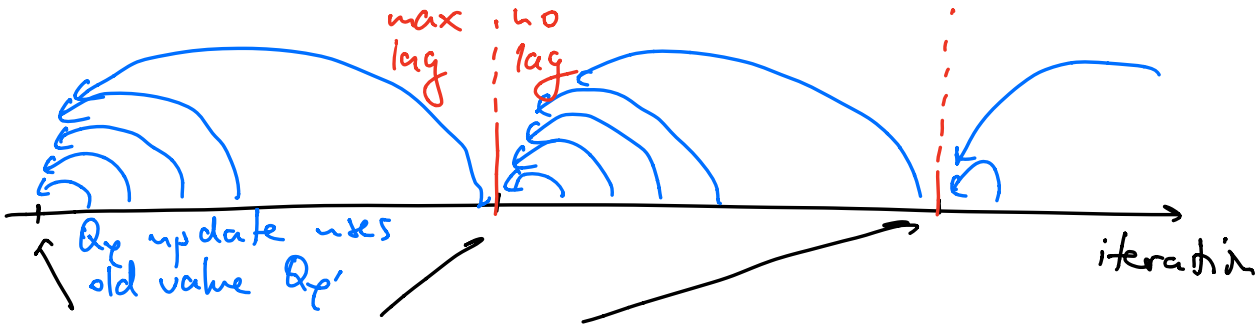
using target $Q_{\varphi'}$-network

y_j is indep. of φ

4. $\varphi \leftarrow \varphi - \alpha \sum_j \nabla_\varphi Q_\varphi(s_j, a_j)\left[Q_\varphi(s_j, a_j) - y_j\right]$

5. update target network: $\varphi' \leftarrow \varphi$ at a fixed number of steps.

— <u>alternative rule</u> to update the target $Q_{\varphi'}$-net at step 5:



iterations to update $\varphi' \leftarrow \varphi$

→ <u>issue</u>: abrupt change when $\varphi' \leftarrow \varphi$, can we smear out the boundary ⋮ ?

→ <u>Polyak averaging</u>:

$\varphi' \leftarrow \tau\varphi' + (1-\tau)\varphi \quad , \tau = 0.999$

<u>Note</u>: now we update $Q_{\varphi'}$-net at every step.

1. take a_j using some policy, e.g. ε-greedy wrt Q_φ, observe (s_j, a_j, r_j, s_j'); add it to buffer B

2. update target $Q_{\varphi'}$-net parameters: $\varphi' \leftarrow \varphi$ using Polyak averaging

3. sample minibatch from B uniformly $\{(s_j, a_j, r_j, s_j')\}_j$

4. compute: $y_j = r(s_j, a_j) + \gamma \max_{a_j'} Q_{\varphi'}(s_j', a_j')$
using target $Q_{\varphi'}$-network

5. $\varphi \leftarrow \varphi - \alpha \sum_j \nabla_\varphi Q_\varphi(s_j, a_j)\left[Q_\varphi(s_j, a_j) - y_j\right]$

— <u>general view</u>: Deep Q-learning:

process 1: data collection

process 2: target update

process 3: Q-function regression

- different processes can be repeated more/less frequently
  → get different Q-learning algo's discussed so far

-11-

eg.

1) online Q-learning: processes 1, 2, 3 all run at same speed

2) DQN: processes 1 & 3 run at same speed
process 2 is slow

3) fitted Q-iteration: process 3 is in the inner loop of process 2, which is in inner loop of process 1

— Q values may not be accurate

- do the values $Q(s,a)$ correspond to the actual expected return?

recall: Q-learning estimates are optimistic b/c of the $\max_{a'}$

→ use double Q-learning:

recall: $\max_{a'} Q(s',a') = Q(s', \text{argmax}_{a'} Q(s',a'))$

$Q_{\varphi_A}(s,a) \leftarrow r + \gamma Q_{\varphi_B}(s', \text{argmax}_{a'} Q_{\varphi_A}(s',a'))$

$Q_{\varphi_B}(s,a) \leftarrow r + \gamma Q_{\varphi_A}(s', \text{argmax}_{a'} Q_{\varphi_B}(s',a'))$

avoid evaluating a Q function at an action a' which the argmax of the same Q function

- use Q_φ & target net $Q_{\varphi'}$:

standard Q-learning: $y = r + \gamma Q_{\varphi'}(s', \underset{a'}{\text{argmax}}\, Q_{\varphi'}(s', a'))$

double Q-learning: $y = r + \gamma Q_{\varphi'}(s', \underset{a'}{\text{argmax}}\, Q_{\varphi}(s', a'))$

→ double Q-learning mitigates the overestimation of Q-values!

– Practical implementation: "best practice"

→ network architecture for $Q_\varphi(s, a)$

i) $s \longrightarrow$ $a \longrightarrow$ $DNN_\varphi \longrightarrow Q(s, a)$

ii) $s \longrightarrow DNN_\varphi \Rightarrow$ $Q(s, a_1)$, $Q(s, a_2)$, ⋮, $Q(s, a_{|A|})$ for $a_1, a_2, \ldots, a_{|A|}$

→ Q-learning takes time to stabilize:

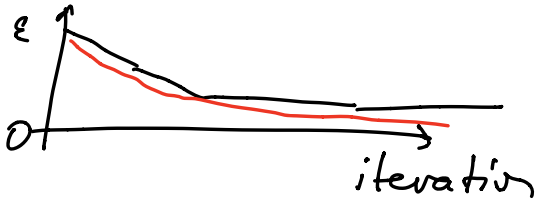
- test your code on a simple task first
  e.g. cartpole with a simple state space (not images, etc.)
  mountain car, etc.

- once you are convinced your code works, go to actual problem & do a hyperparameter search!

→ use large replay buffers

→ start with high exploration, then use schedule

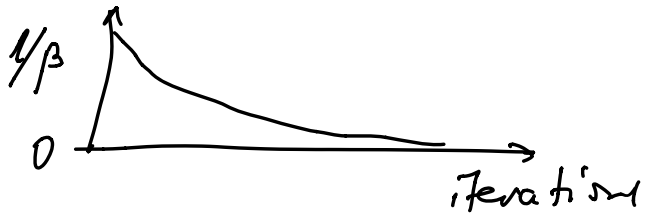
i) ε-greedy with



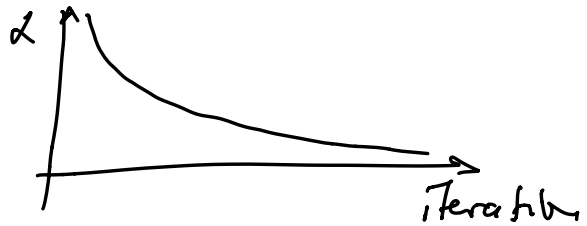
ii) Boltzmann exploration

choose an action according to $p(a) \propto e^{\beta Q(s,a)}$

$1/\beta$ exploration temperature



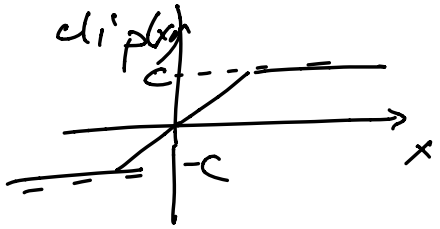
→ use a schedule for learning rate / step size



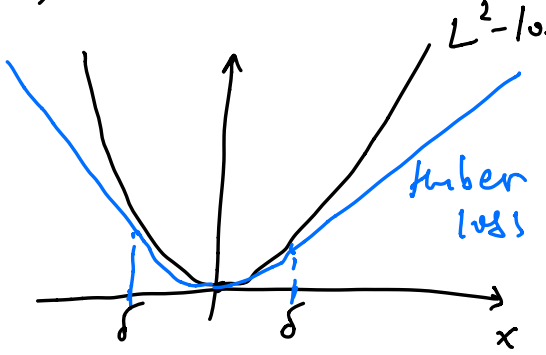
ADAM may help (but it may also mess things up)

→ Bellman error gradients can be large:

i) clip gradients:



ii) use Huber loss function, instead of L^2-norm



$$L(x) = \begin{cases} x^2/2, & |x| \le \delta \\ \delta|x| - \delta^2/2, & |x| > \delta \end{cases}$$

→ use double Q-learning

→ DQN takes time to train; be patient.

→ run multiple random seeds & average the results.

- single-seed simulations not necessarily representative!