ml-mipt-advanced

Lecture 09. Approximate Value-Based Methods. DQN.

Nikolay Karpachev
1.04.2024

Value Based RL

Value Based RL

- Goal: optmize total return

$$G_t = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{T-1} R_T$$

Value Based RL

- Goal: optmize total return

$$G_t = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{T-1} R_T$$

- Assign numeric value to each state and (state, action) pair

$$v_\pi(s) \triangleq \mathbb{E}_\pi [G_t \,|\, S_t = s] \qquad q_\pi(s, a) = \mathbb{E}_\pi [G_t \,|\, S_t = s, A_t = a]$$

Value Based RL

- Goal: optmize total return

$$G_t = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{T-1} R_T$$

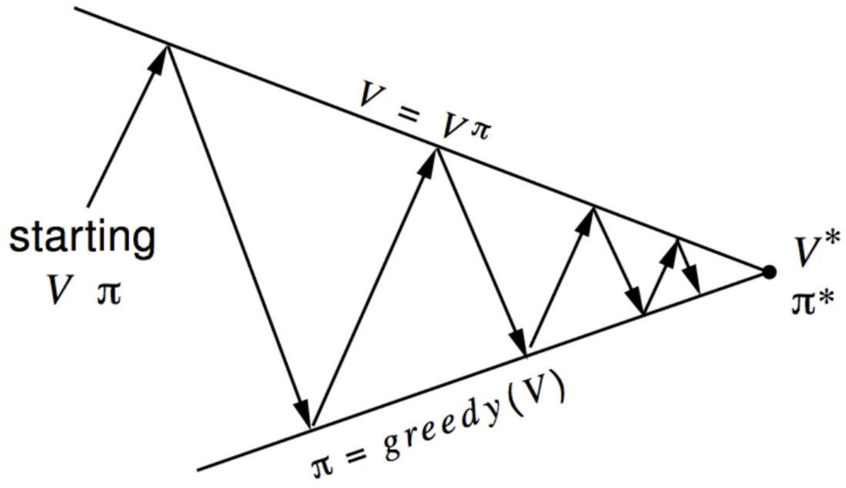
- Assign numeric value to each state and (state, action) pair

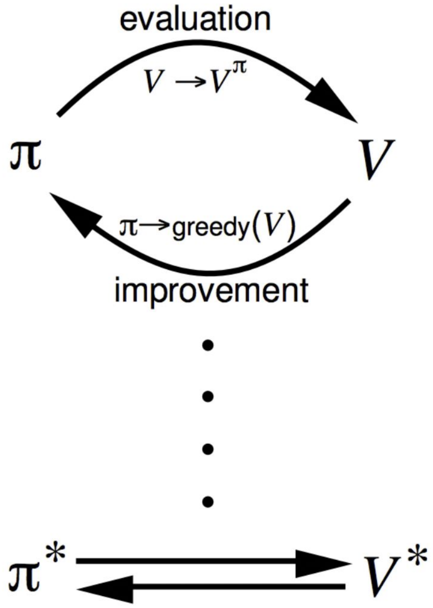
$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s] \qquad q_\pi(s,a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a]$$

- Optimize policy
  - Estimate correct value functions for current policy
  - Improve policy using estimated V

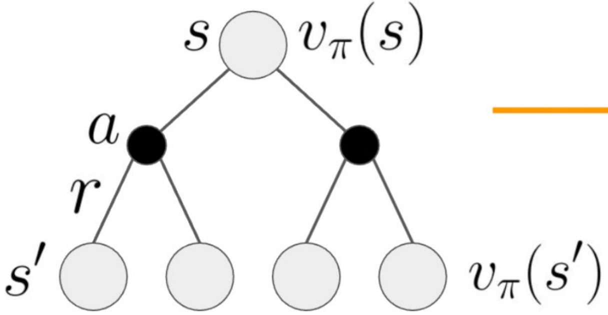
Policy iteration

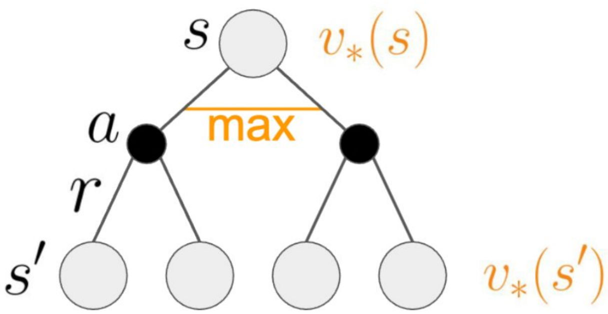
1. Policy evaluation - given policy p, estimate V_p
2. Policy improvement - improve p greedily





Policy iteration





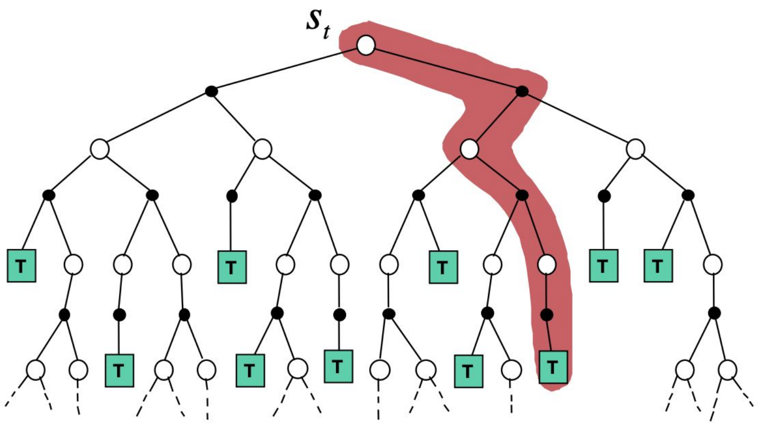
Bellman expectation equation for v(s)

Bellman optimality equation for $v_*(s)$

Model Free Policy Evaluation

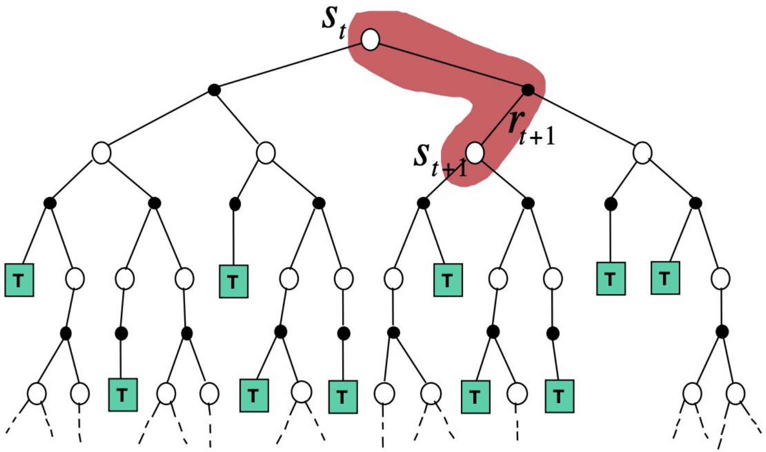
Monte-Carlo backup

$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t - V(S_t)\right)$$



Temporal Difference backup

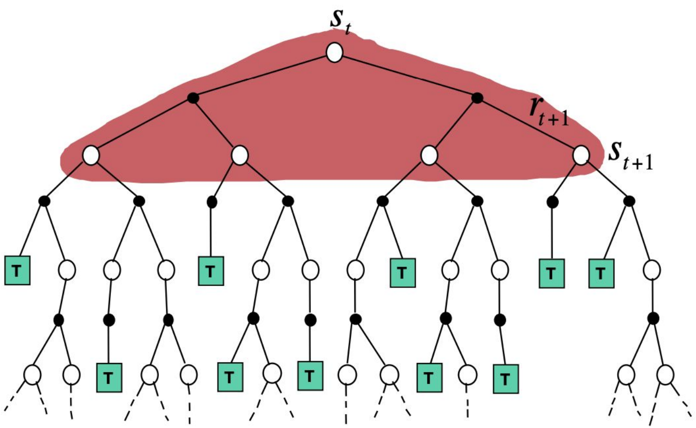
$$V(S_t) \leftarrow V(S_t) + \alpha\left(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right)$$



DP vs. Model Free TD

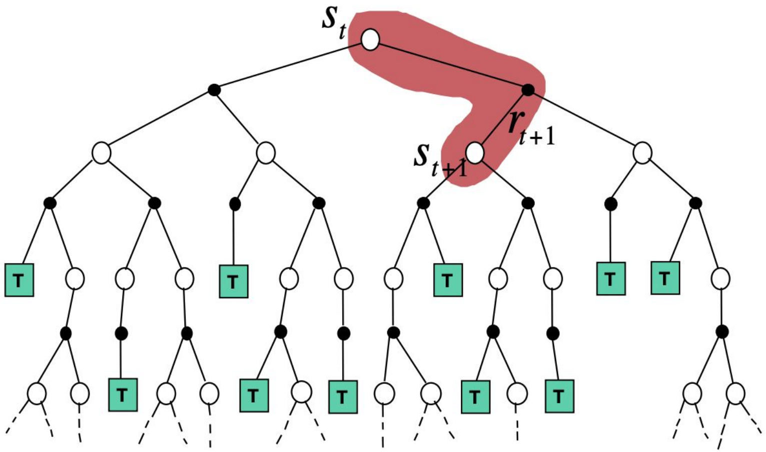
Dynamic Programming backup

$$V(S_t) \leftarrow \mathbb{E}_\pi \left[R_{t+1} + \gamma V(S_{t+1})\right]$$

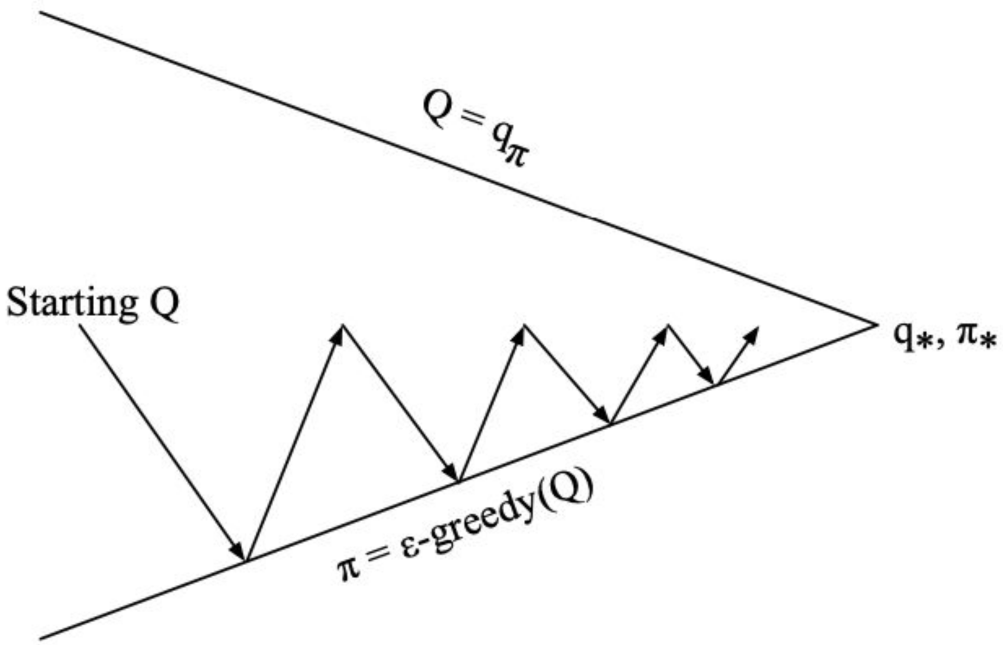


Temporal Difference backup

$$V(S_t) \leftarrow V(S_t) + \alpha \left(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right)$$



Monte-Carlo Model Free Control



Just like in value iteration

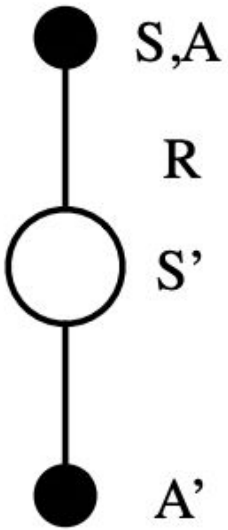
on every episode *(!)*

1. *Approx. policy evaluation*
2. *Policy improvement*

Model Free Control: SARSA

Basically, SARSA is

- One-step Temporal Difference Policy Evaluation
- Epsilon-greedy Policy Improvement



$$Q(S,A) \leftarrow Q(S,A) + \alpha \left(R + \gamma Q(S',A') - Q(S,A)\right)$$

Policy evaluation

Sample from experience

Off-policy model-free control: Q-Learning

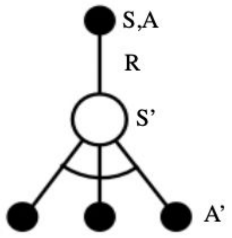
<u>Key idea:</u> optimize greedy policy

- Our policy is greedy (or eps-greedy) w.r.t. current q-values
- Hence, no A’ is required in (S, A, R, S’, A’) updates

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(R_{t+1} + \gamma Q(S_{t+1}, A') - Q(S_t, A_t) \right)$$

$$\pi(S_{t+1}) = \underset{a'}{\text{argmax}}\ Q(S_{t+1}, a')$$

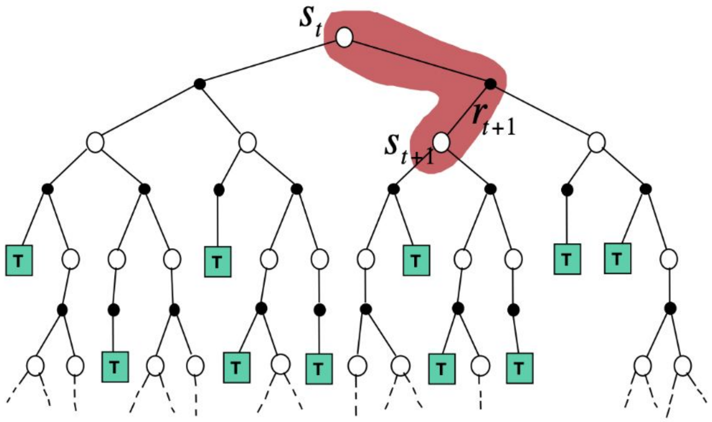
$$Q(S, A) \leftarrow Q(S, A) + \alpha \left(R + \gamma \max_{a'} Q(S', a') - Q(S, A) \right)$$



Approximate Methods

Continuous Spaces

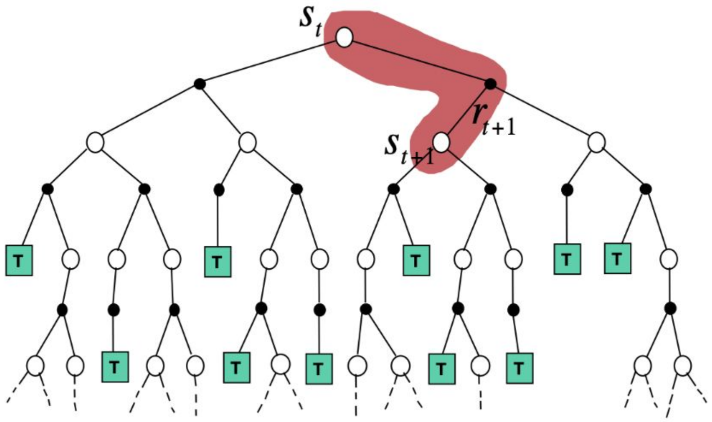
Tabular Q-Learning



Great!

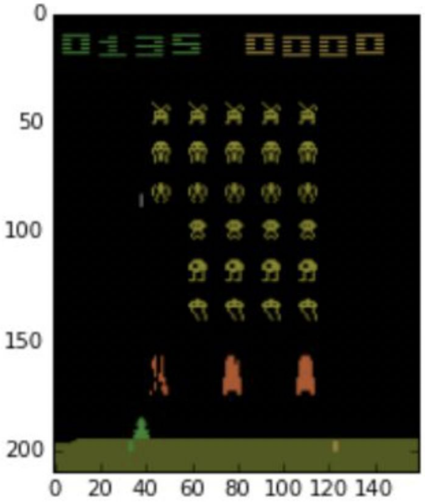
Continuous Spaces

Tabular Q-Learning



Great!

Atari games



How many states are there?

approximately

$$|S| = 2^{210 \cdot 160 \cdot 8 \cdot 3}$$

Q-Learning

Training step

$$Q(s_t, a_t) \longleftarrow Q(s_t, a_t) + \alpha(r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t))$$

Q-Learning

Training step

$$Q(s_t, a_t) \longleftarrow Q(s_t, a_t) + \alpha(r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t))$$

Q-learning as MSE minimization

$$L = (r_t + \gamma \max_{a'} \underbrace{\boxed{Q(s_{t+1}, a')}}_{\text{Const}} - Q(s_t, a_t))^2$$

$$\nabla L = 2 \cdot (r_t + \gamma \max_{a'} \boxed{Q(s_{t+1}, a')} - Q(s_t, a_t))$$

Q-Learning

Training step

$$Q(s_t, a_t) \longleftarrow Q(s_t, a_t) + \alpha(r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t))$$

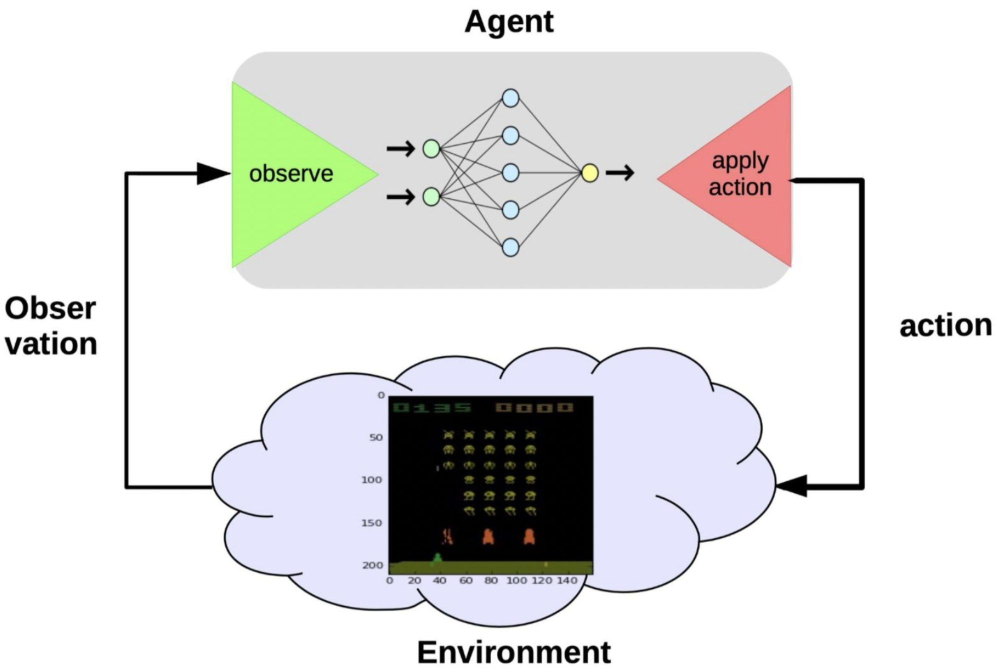
Q-learning as MSE minimization

$$L = (r_t + \gamma \max_{a'} \underbrace{\boxed{Q(s_{t+1}, a')}}_{\text{Const}} - Q(s_t, a_t))^2$$

$$\nabla L = 2 \cdot (r_t + \gamma \max_{a'} \boxed{Q(s_{t+1}, a')} - Q(s_t, a_t))$$

running mean updates are equivalent to minimizing MSE with targets: $r_t + \gamma \cdot max_{a'} Q(s_{t+1}, a')$

Approximate Q-Learning

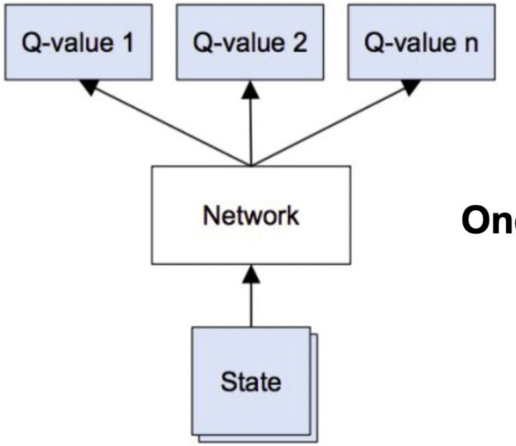


Approximate Q-Learning

- What are we predicting? And what are the features?

Approximate Q-Learning

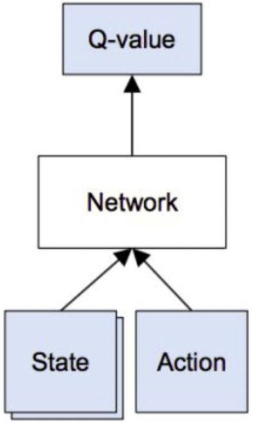
- What are we predicting? And what are the features?



One pass for all actions

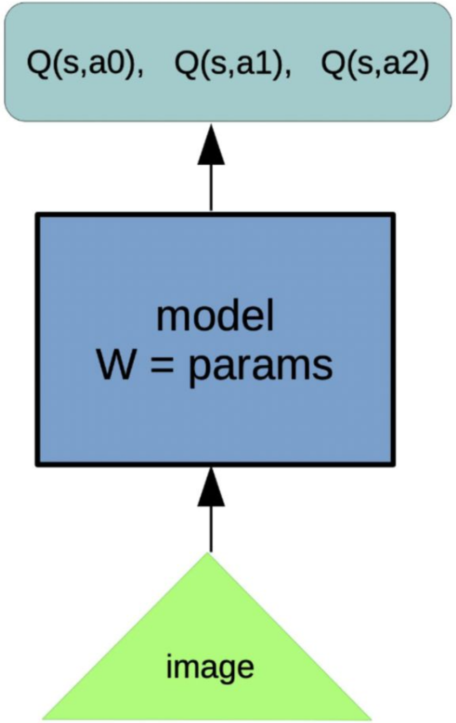
Given **s** predict all q-values
Q(s,a0), Q(s,a1), Q(s,a2)

Continuous control or large number of actions



Given **(s,a)**
Predict Q(s,a)

Approximate Q-Learning



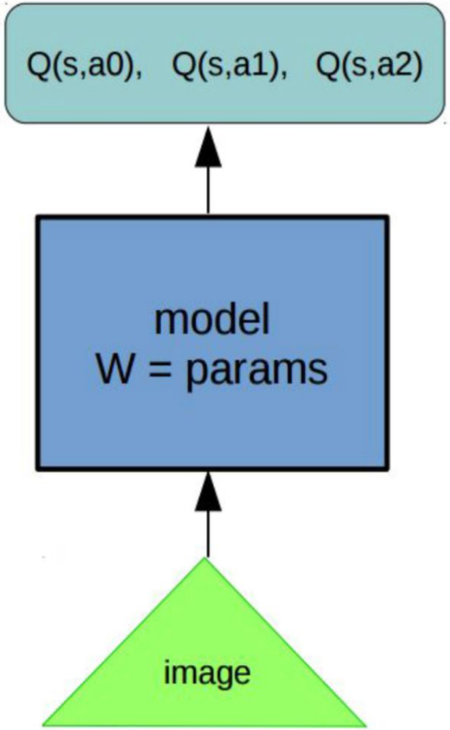
$$\hat{Q}(s_t, a_t) = r + \gamma \cdot max_{a'} \hat{Q}(s_{t+1}, a')$$

$$L = \left(Q(s_t, a_t) - \boxed{\left[r + \gamma \cdot max_{a'} Q(s_{t+1}, a')\right]}\right)^2$$

Consider const

$$w_{t+1} = w_t - \alpha \cdot \frac{\delta L}{\delta w}$$

Approximate Q-Learning



Objective:

$$L=\left(Q(s_t,a_t)-\underbrace{\hat{Q}(s_t,a_t)}_{\text{consider const}}\right)^2$$

Q-learning:

$$\hat{Q}(s_t,a_t)=r+\gamma\cdot max_{a'}Q(s_{t+1},a')$$

SARSA:

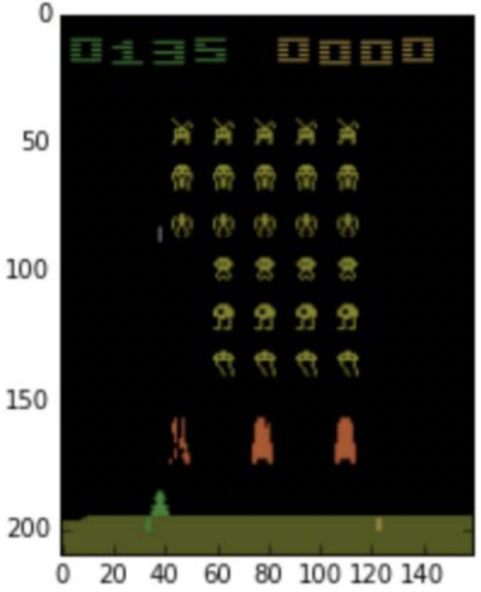
$$\hat{Q}(s_t,a_t)=r+\gamma\cdot Q(s_{t+1},a_{t+1})$$

Expected Value SARSA:

$$\hat{Q}(s_t,a_t)=r+\gamma\cdot \underset{a'\sim\pi(a|s)}{E} Q(s_{t+1},a')$$

Approximate Q-Learning

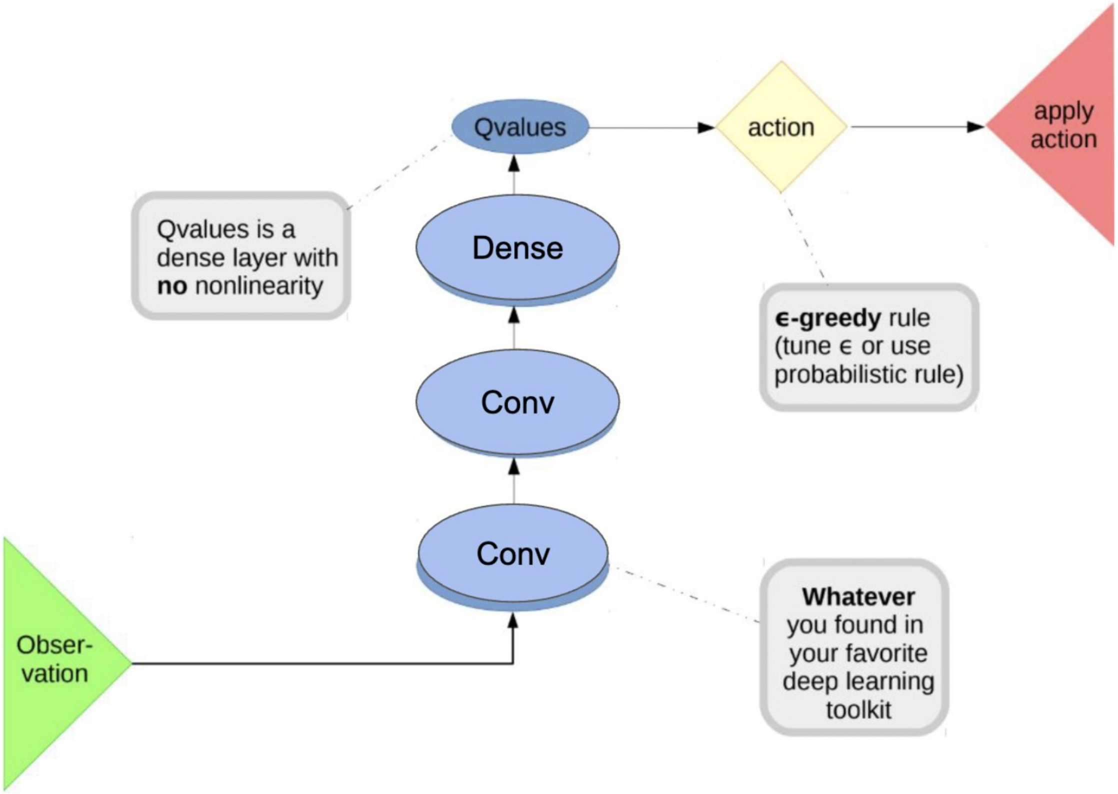
Atari SpaceInvaders



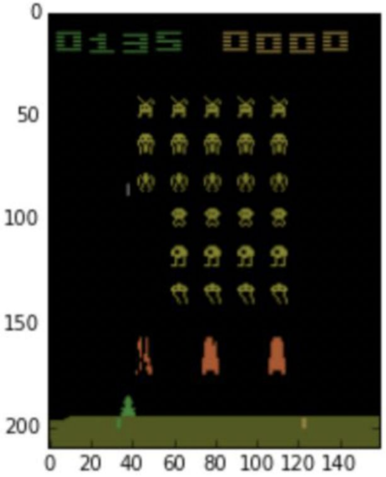
Q.: How to actually predict Q(s, a)
(features, architecture)

Basic DQN

DQN: Deep Q Network



Autocorrelation

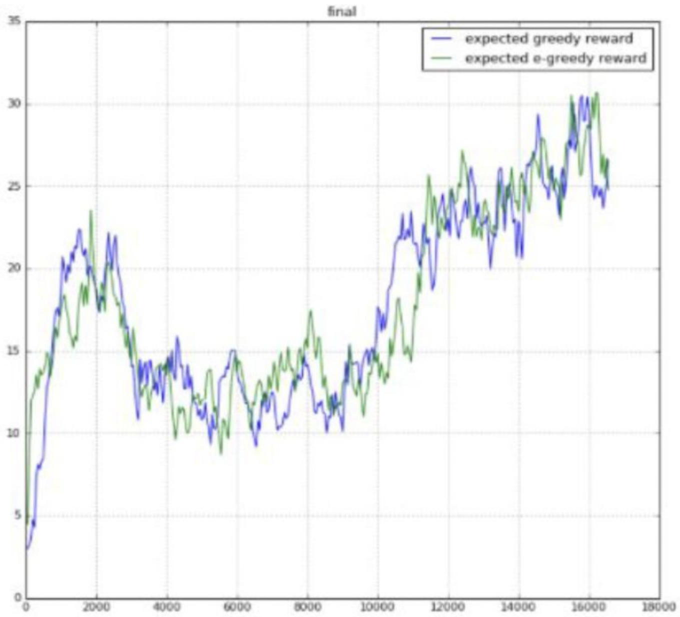


How bad it is if agent spends

next 1000 ticks under the left rock?
(while training)

Autocorrelation

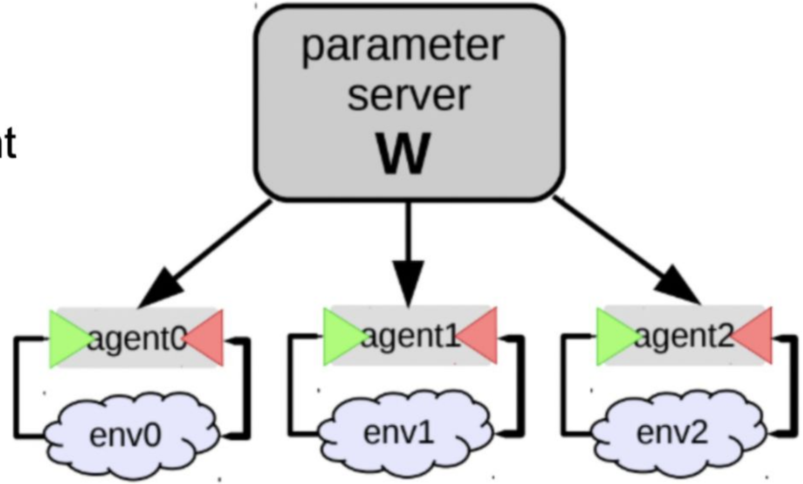
- Training samples are **not “i.i.d”**,
- Model forgets parts of environment it hasn't visited for some time
- Drops on learning curve
- **Any ideas?**



Autocorrelation: Multiagent Training

Idea: Throw in several agents with shared **W**.

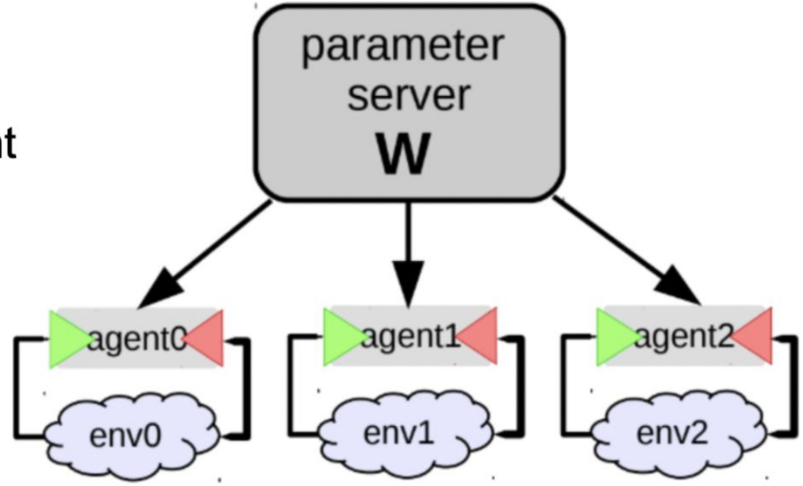
- Chances are, they will be exploring different parts of the environment
- More stable training
- Requires a lot of interaction



Autocorrelation: Multiagent Training

Idea: Throw in several agents with shared **W**.

- Chances are, they will be exploring different parts of the environment
- More stable training
- Requires a lot of interaction



Question: *your agent is a real robot car. Will there be any problems?*

Autocorrelation: Experience Replay

Idea:

Store several past interactions

<s,a,r,s'>

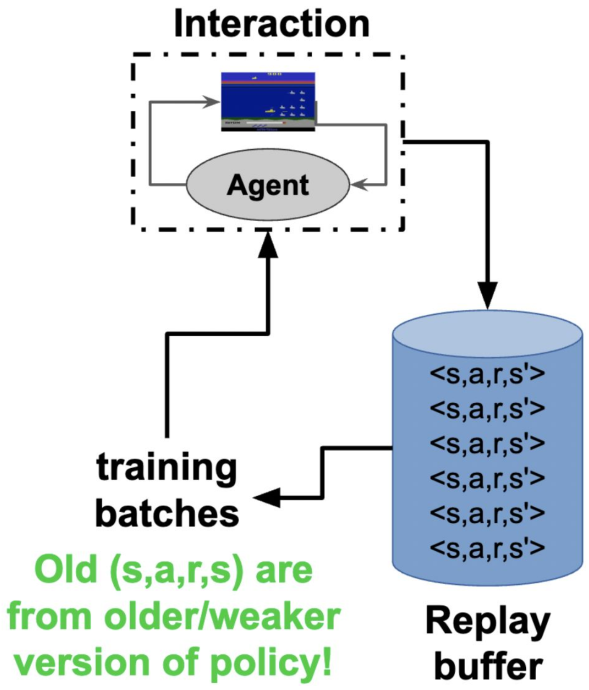
Train on random subsamples

Training curriculum:

- Play 1 step and record it
- Pick N random transitions to train

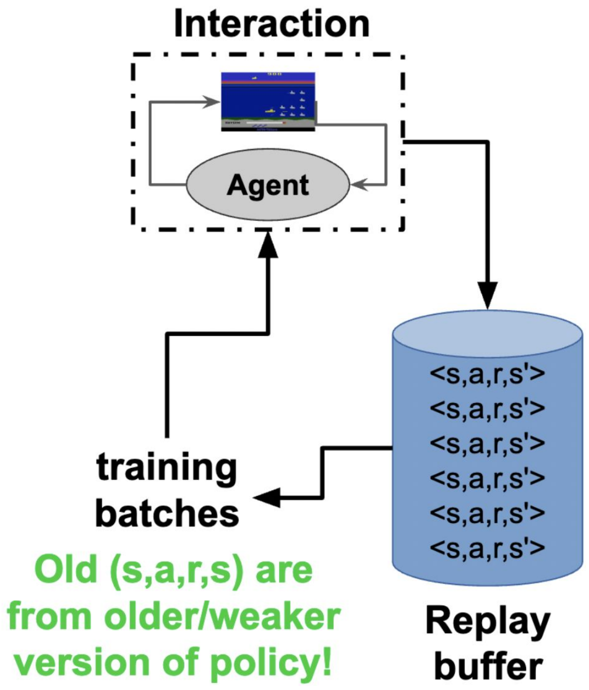
Profit:

you don't need to revisit same (s,a) many times to learn it.



Autocorrelation: Experience Replay

- Atari DQN > 10^5 interactions
- Closer to i.i.d.
  pool contains several sessions
- Older interactions were
  obtained under weaker policy



Experience Replay

- You approximate Q(s, a) with a neural network
- You use **experience replay** when training

Question: which of those algorithms will fail?

- Q-learning
- SARSA
- CEM
- Expected Value SARSA

Experience Replay

- You approximate Q(s, a) with a neural network
- You use **experience replay** when training

Agent trains off-policy on an older version of himself

Question: which of those algorithms will fail?

Off-policy methods work, **On-policy methods are super slow (fail)**

- **Q-learning**
- **SARSA**
- **CEM**
- **Expected Value SARSA**

Experience Replay

When training with on-policy methods,

- use no (or small) experience replay
- compensate with parallel game sessions

DQN: Observation Design

Left or right?

DQN: Observation Design

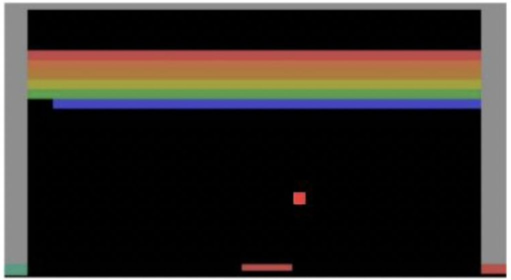
<u>N-Gram trick</u>

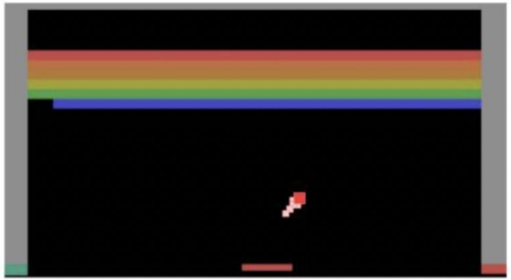
Idea:

$$s_t \neq o(s_t)$$

$$s_t \approx (o(s_{t-n}), a_{t-n}, \ldots, o(s_{t-1}), a_{t-1}, o(s_t))$$

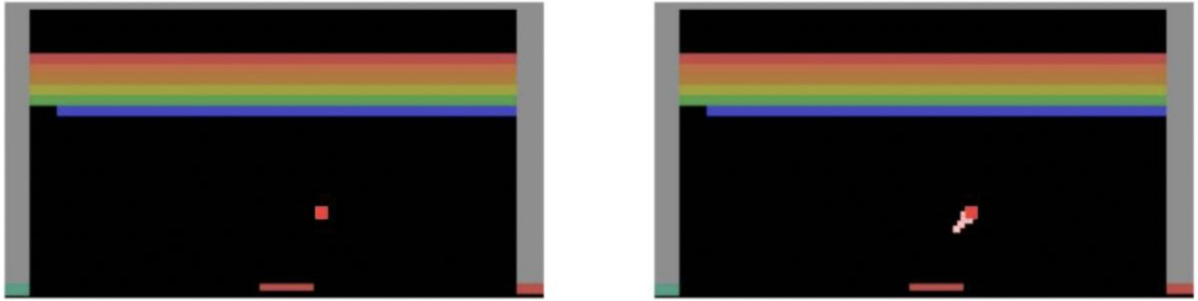
e.g. ball movement in breakout





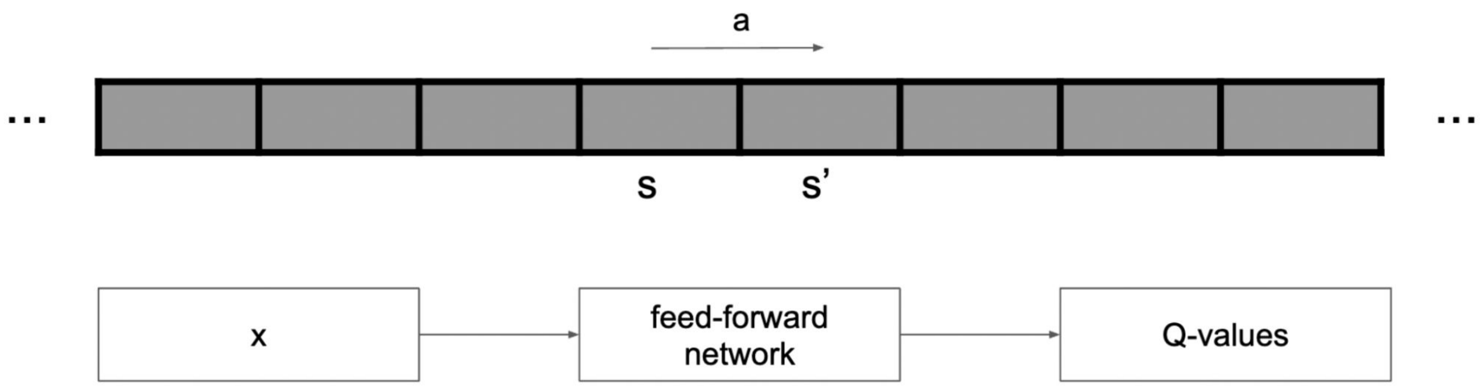
DQN: Observation Design

<u>N-Gram trick</u>



- Nth-order markov assumption
- Works for velocity/timers
- Fails for anything longer that N frames
- Impractical for large N

DQN: Target Networks



Target is based on prediction

Q(s, a) correlates with Q(s', a)

DQN: Target Networks

Idea: use network with frozen weights to compute the target

$$L(\Theta) = E_{s\sim S,a\sim A}[(Q(s,a,\Theta) - (r + \gamma \underbrace{\boxed{\max_{a'} Q(s',a',\Theta^-)}}_{\textbf{Const}}))^2]$$

where Θ^- is the frozen weights

Hard target network:

Update Θ^- every **n** steps and set its weights as Θ

DQN: Target Networks

Idea: use network with frozen weights to compute the target

$$L(\Theta) = E_{s\sim S,a\sim A}[(Q(s,a,\Theta) - (r + \gamma \underbrace{\boxed{\max_{a'} Q(s',a',\Theta^-)}}_{\textbf{Const}}))^2]$$

where Θ^- is the frozen weights

Hard target network:

Update Θ^- every **n** steps and set its weights as Θ

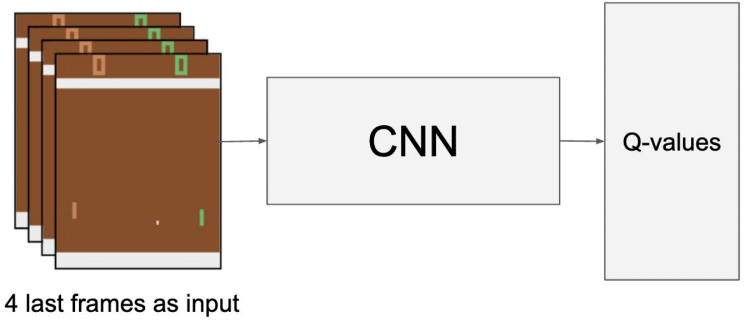
Soft target network:

Update Θ^- every step:

$$\Theta^- = (1-\alpha)\Theta^- + \alpha\Theta$$

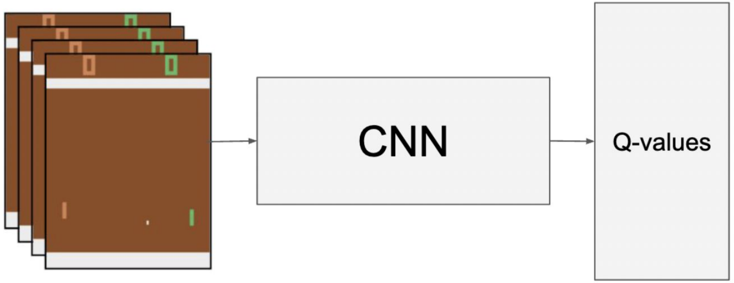
DQN: Putting it all together

[Playing Atari with Deep Reinforcement Learning (Deepmind, 2013)]



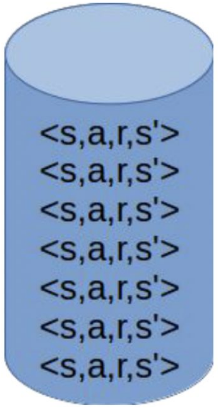
DQN: Putting it all together

[Playing Atari with Deep Reinforcement Learning (Deepmind, 2013)]



4 last frames as input

Experience replay



10^6 last transitions

Update weights using:

$$L(\Theta) = E_{s\sim S, a\sim A}[(Q(s,a,\Theta) - (r + \gamma \max_{a'} Q(s',a',\Theta^-)))^2]$$

Update Θ^- every 5000 train steps

Outro

- Value-based RL
- Env model is unknown: MC and TD from samples
- Continuous state/action space: predict Q with MSE on TD target
- DQN
  - Training hacks and state design
  - Autocorrelation
  - Target Networks

Acknowledgements

This lecture uses materials from

(1) RL Lectures by David Siver (licensed CC-BY-NC 4.0)

(2) Practical_RL lectures by Yandex Data School (Unlicense license)

Questions?