Deep Reinforcement Learning

CS440/ECE448 Lecture 32

Mark Hasegawa-Johnson, 4/2020,
including slides by Svetlana Lazebnik, 11/2017

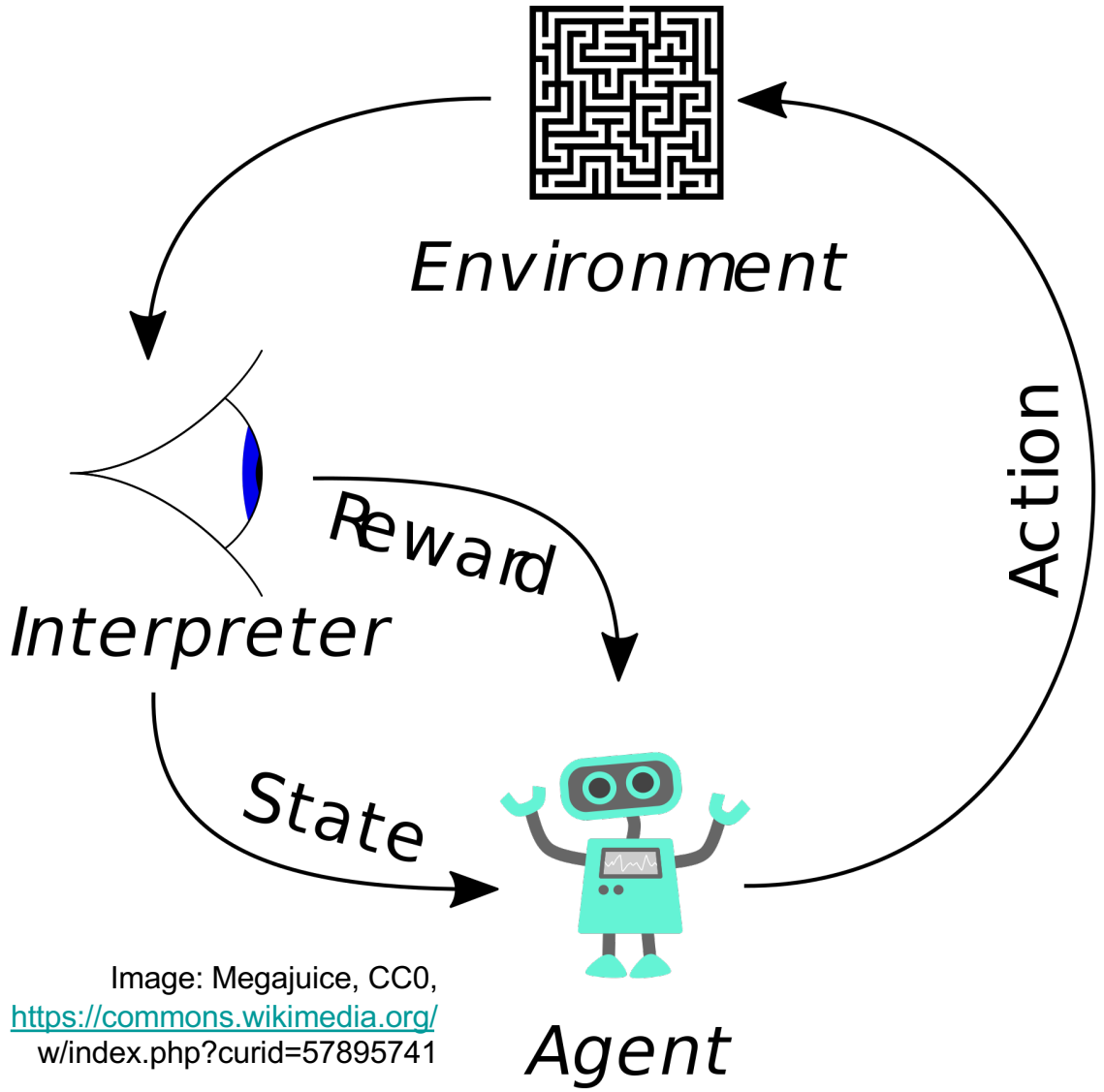


Image: Megajuice, CC0, https://commons.wikimedia.org/w/index.php?curid=57895741

Last lecture: Q-learning for discrete s, a

- So far, we've assumed a *lookup table* representation for utility function U(s) or action-utility function Q(s,a)
- This does not work if the state space is really large or continuous

This time: Function approximation

- Approximate $Q(s, a)$ by a ***parameterized function***, that is, by a function $\hat{Q}(s, a; W)$ that depends on some matrix of trainable parameters, *W*.
- Learn W by playing the game.

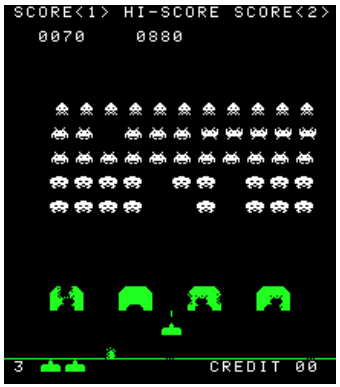
Outline

- Deep Q-learning: learn an MMSE estimate of Q(s,a)
  - Off-policy vs. On-policy
  - Batch learning (experience replay)
- Policy learning: learn the policy with the maximum value
  - Estimating the value: actor-critic network
  - Estimating the relative value: advantage actor-critic
- Imitation learning: MMSE imitation of a human expert
  - A good way to initialize Q-learning or policy learning

Deep Q learning

Instead of discrete s, suppose $\vec{s}$ is a vector of real numbers, e.g., the image from the robot's eye camera:

$$\vec{s} = [s_1, \ldots, s_D] =$$

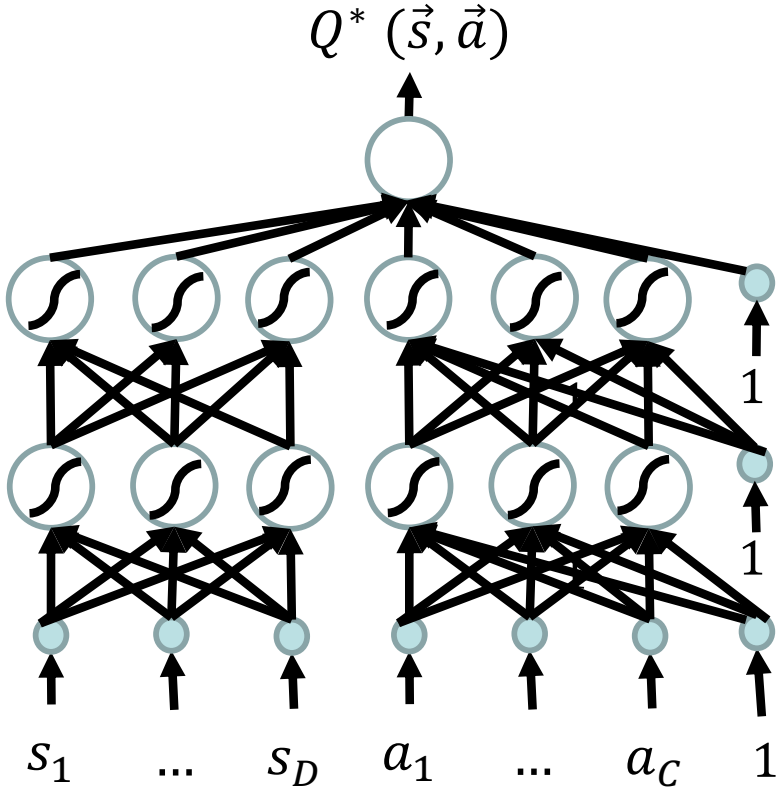


Copyright Taito.

Instead of discrete a, suppose $\vec{a}$ is a vector, e.g., cannon angle and velocity,

$$\vec{a} = [a_1, \ldots, a_C]$$

Deep Q-learning uses a neural network to compute an estimate $Q^*(\vec{s}, \vec{a})$ which is as close as possible to $Q(\vec{s}, \vec{a})$.



MMSE Deep Q learning

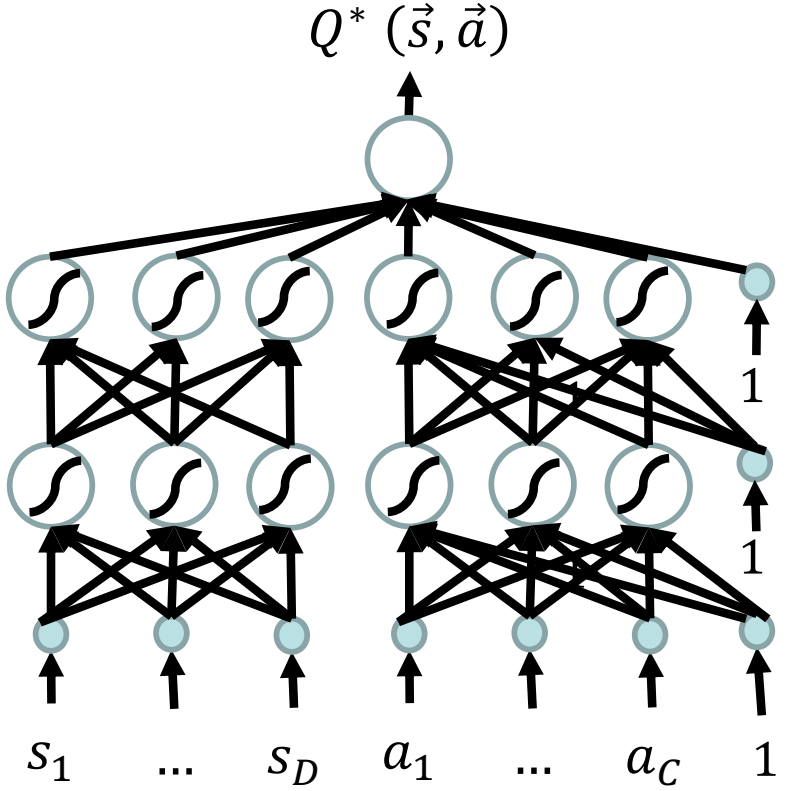
Suppose we train the neural network weights in order to minimize the mean-squared error (MMSE):

$$\mathcal{L} = \frac{1}{2} E[(Q^*(\vec{s}, \vec{a}) - Q(\vec{s}, \vec{a}))^2]$$

(where I'm using $E[\cdot]$ as a lazy way to write "average over all training runs of the game).

Then, for each weight w, we update as

$$w \leftarrow w - \eta \frac{d\mathcal{L}}{dw}$$

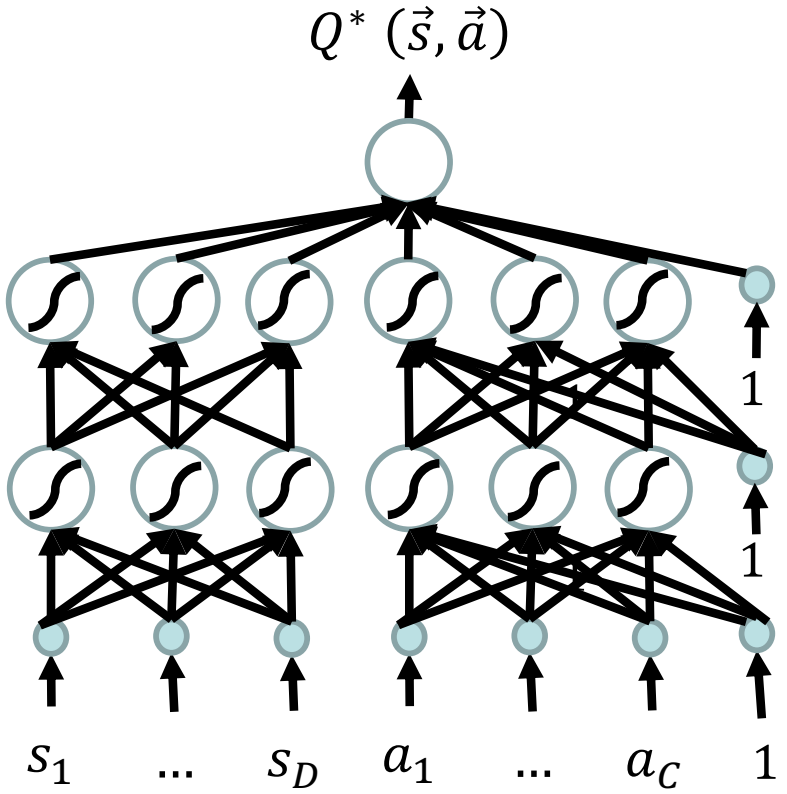


What makes deep Q learning harder than normal neural network training

- We don't know the true value of $Q(\vec{s}, \vec{a})$ for ***any*** of the training runs!

- $Q(\vec{s}, \vec{a})$ is defined to be the expected value of performing action $\vec{a}$. We never know its true expected value: all we know is whether we won or lost that particular game.

- So we can't compute $\mathcal{L}$, and we can't compute $\frac{d\mathcal{L}}{dw}$, and we can't update w!



The solution: Q_{local}

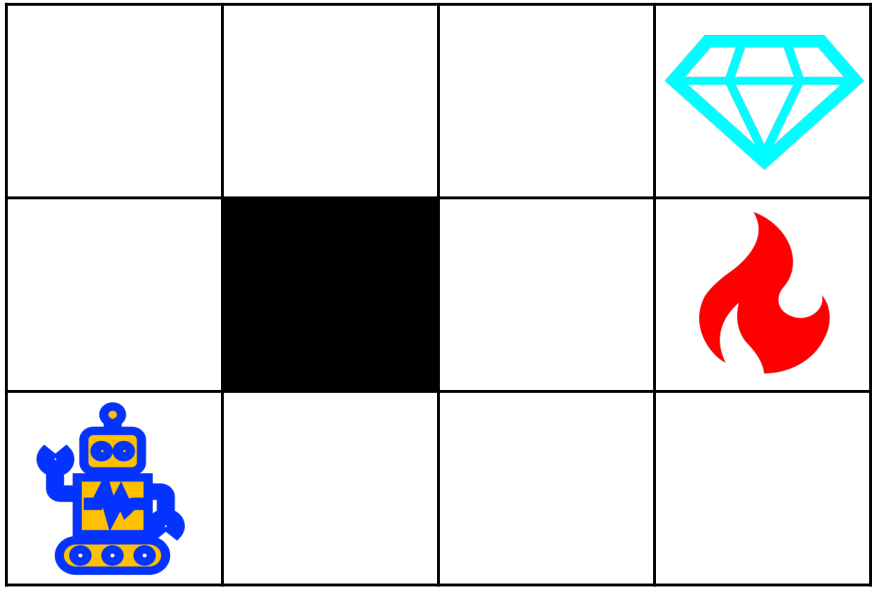
Remember that Q learning was defined as

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha\big(Q_{local}(s,a) - Q_t(s,a)\big)$$

where $Q_{local}(s,a)$ is defined, e.g., in TD as

$$Q_{local}(s,a) = R_t(s) + \gamma \max_{a'} Q_t(s',a')$$

…for s' equal to the next state we reach after action a on **this particular game**.



The solution: Q_{local}

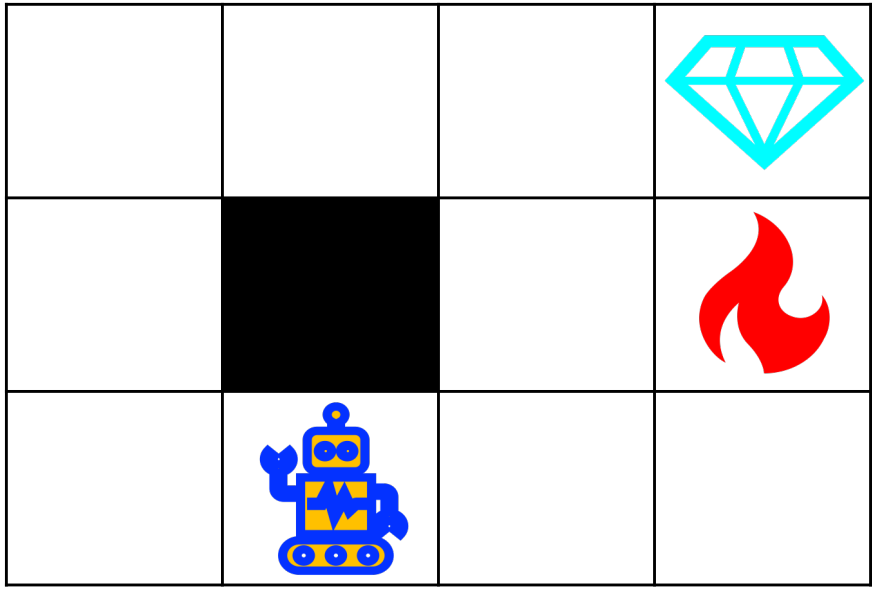
Let's define deep Q learning using the same Q_{local}:

$$\mathcal{L} = \frac{1}{2} E[(Q^*(\vec{s}, \vec{a}) - Q_{local}(\vec{s}, \vec{a}))^2]$$

where $Q_{local}(\vec{s}, \vec{a})$ is:

$$Q_{local}(\vec{s}, \vec{a}) = R_t(\vec{s}) + \gamma \max_{\vec{a}'} Q^*(\vec{s}', \vec{a}')$$

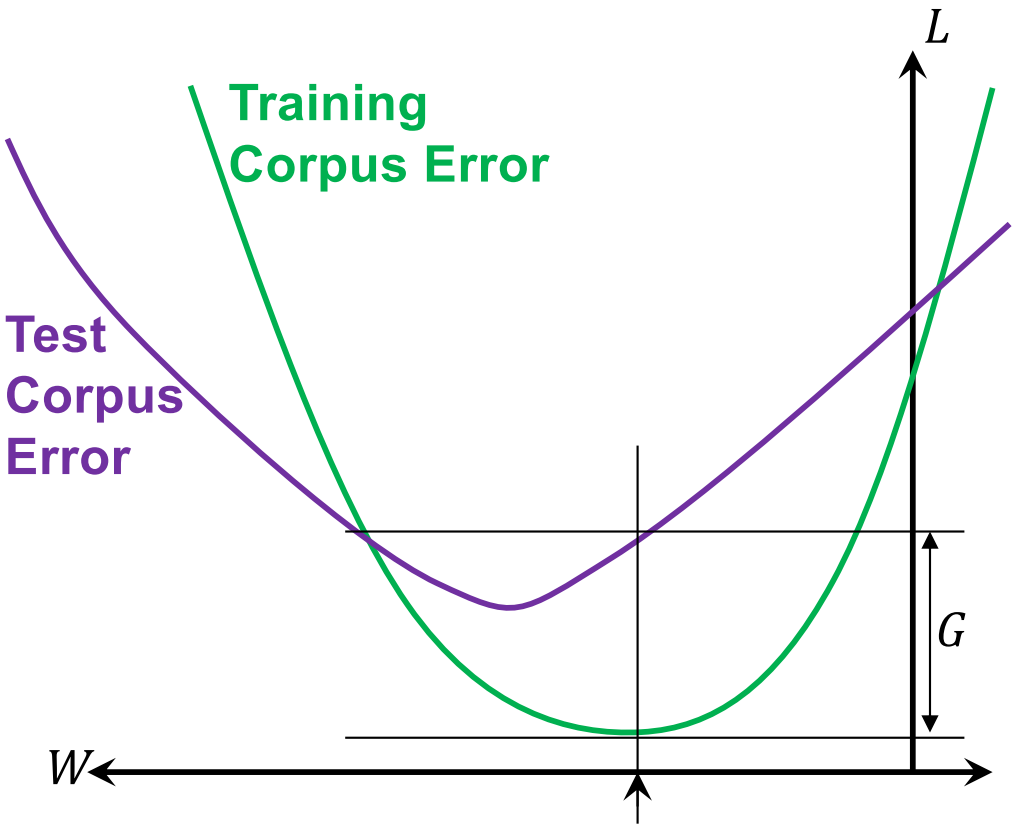
Now we have an L that depends only on things we know ($Q^*(\vec{s}, \vec{a})$, $R_t(\vec{s})$, and $Q^*(\vec{s}', \vec{a}')$), so it can be calculated, differentiated, and used to update the neural network.



Outline

- Deep Q-learning: learn an MMSE estimate of Q(s,a)
  - Off-policy vs. On-policy
  - Batch learning (experience replay)
- Policy learning: learn the policy with the maximum value
  - Estimating the value: actor-critic network
  - Estimating the relative value: advantage actor-critic
- Imitation learning: MMSE imitation of a human expert
  - A good way to initialize Q-learning or policy learning

Convergence of neural networks



Training the neural net finds this set of weights by minimizing the training corpus error.

- A general neural net (e.g., a classifier) is trained to minimize the training corpus error.
- Test corpus error might be very different!
- Barron showed: generalization error is $G <$(#hidden nodes/#training tokens)
- As #training tokens$\rightarrow \infty$, $G \rightarrow 0$

Does Q-learning Converge?

- No!
- Because:

$$\vec{a} = \text{argmax}\, Q(\vec{s}, \vec{a})$$

- If we always choose the action that is best, according to our current estimate of the Q-function, then we can never learn anything about any of the other actions!

Epsilon-greedy exploration

- At each time step:
  - With probability ϵ, choose an action at random
  - With probability $1-\epsilon$, choose $\vec{a} = \text{argmax}\, Q(\vec{s}, \vec{a})$
  - (Decaying epsilon version): As $\text{n} \to \infty$,$\epsilon \to 0$, for example, $\epsilon = 1/n$
- Result:
  - As you play the game infinite times, each action is sampled an infinite number of samples, so Q converges, but also,
  - As you play the game infinite times, you start to exploit your knowledge more and more frequently, so that you converge to the best possible policy.
  - … actually, it doesn't always work in practice. To guarantee success, you need a few more tweaks, e.g., Re-Trace algorithm, Munos et al., 2016.

Outline

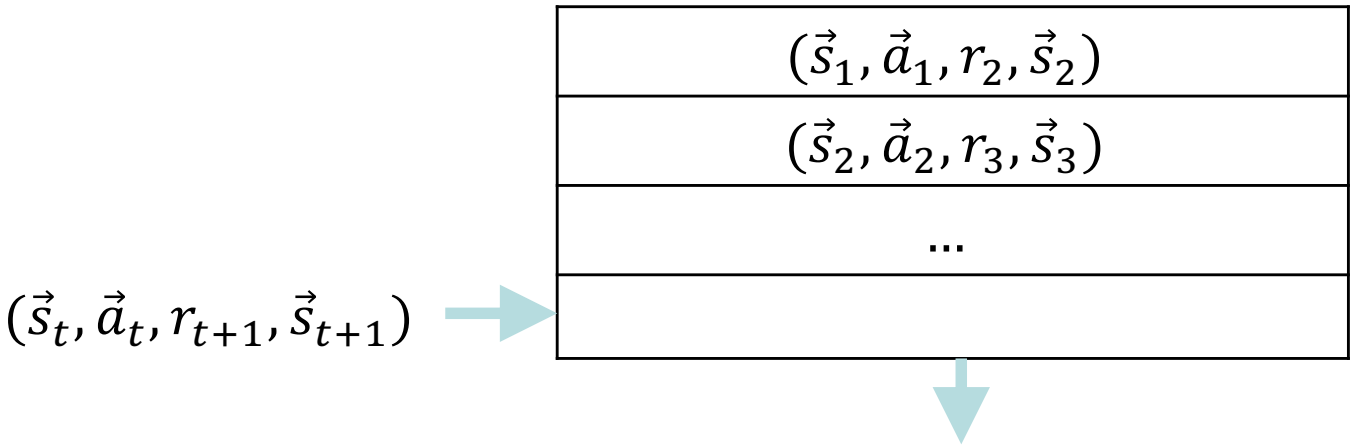
- Deep Q-learning: learn an MMSE estimate of Q(s,a)
  - Off-policy vs. On-policy
  - Batch learning (experience replay)
- Policy learning: learn the policy with the maximum value
  - Estimating the value: actor-critic network
  - Estimating the relative value: advantage actor-critic
- Imitation learning: MMSE imitation of a human expert
  - A good way to initialize Q-learning or policy learning

Dealing with training instability

- Challenges
  - Target values are not fixed
  - Successive experiences are correlated and dependent on the policy
  - Policy may change rapidly with slight changes to parameters, leading to drastic change in data distribution
- Solutions
  - Freeze target Q network
  - Use *experience replay*

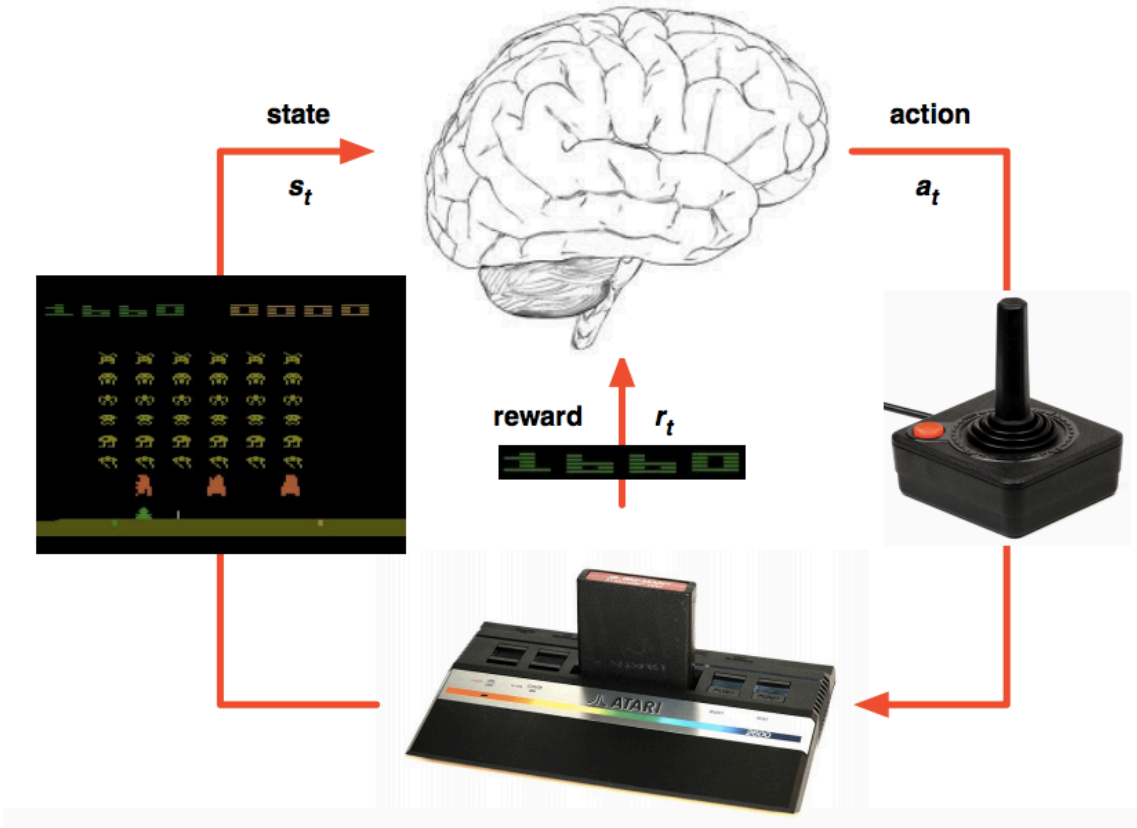
Experience replay

- At each time step:
  - Take action $\vec{a}_t$ according to epsilon-greedy policy
  - Store experience $(\vec{s}_t, \vec{a}_t, r_{t+1}, \vec{s}_{t+1})$ in *replay memory buffer*



- Learning:
  - Randomly sample a minibatch, $\mathcal{D}$, from the replay buffer.

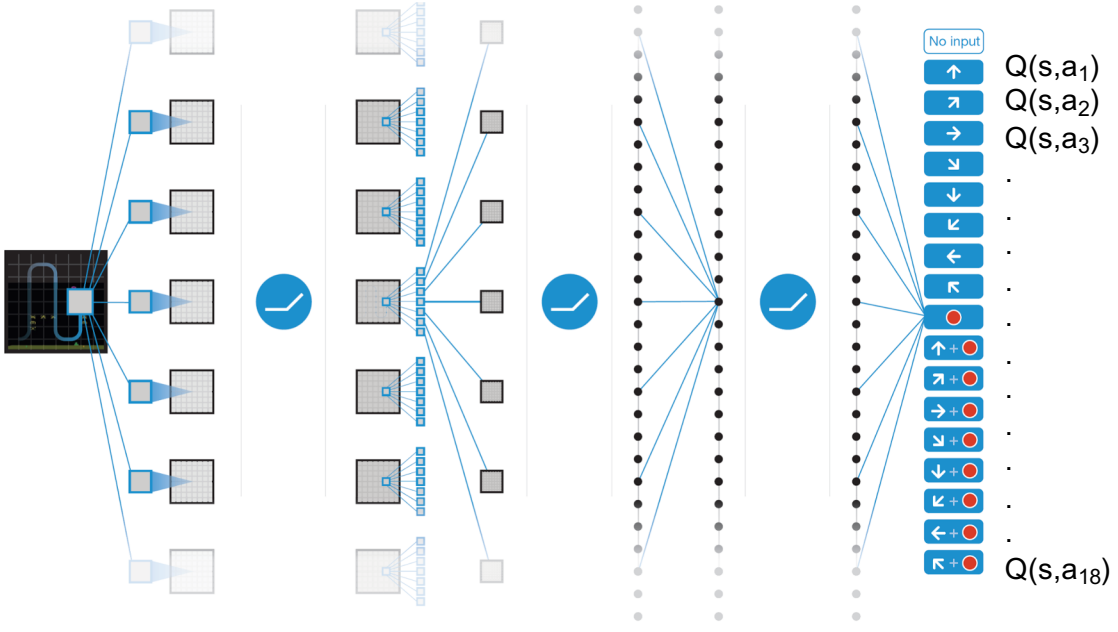
Deep Q learning in Atari



Mnih et al. Human-level control through deep reinforcement learning, *Nature* 2015

Deep Q learning in Atari

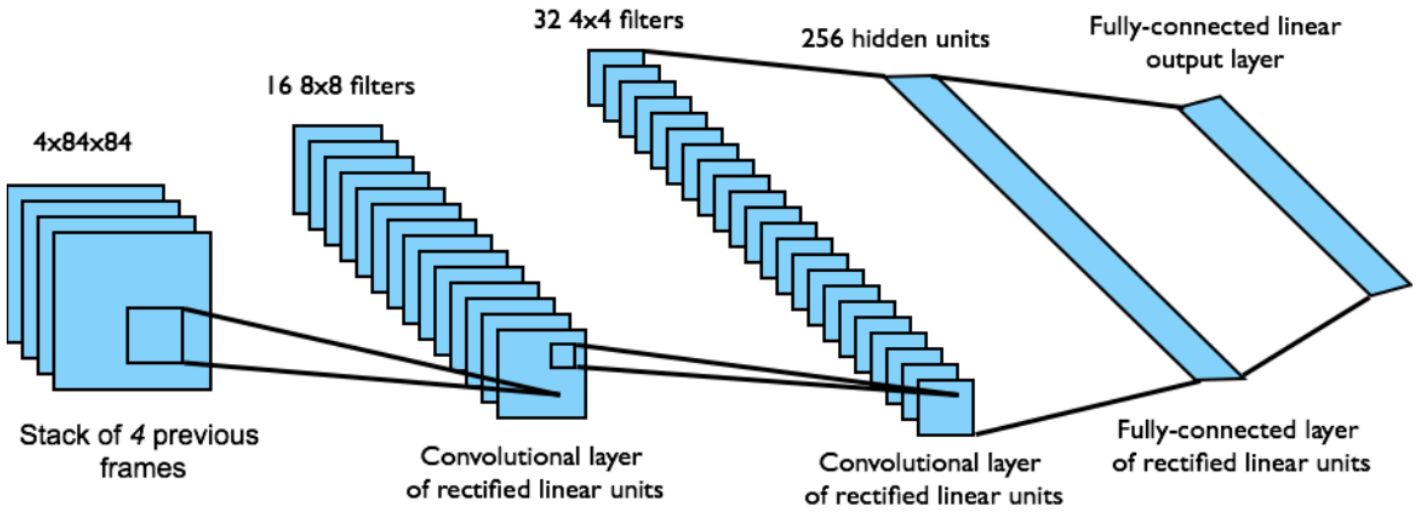
- End-to-end learning of Q(s,a) from pixels s
- Output is Q(s,a) for 18 joystick/button configurations
- Reward is change in score for that step



Mnih et al. Human-level control through deep reinforcement learning, *Nature* 2015

Deep Q learning in Atari

- Input state s is stack of raw pixels from last 4 frames
- Network architecture and hyperparameters fixed for all games



Mnih et al. Human-level control through deep reinforcement learning, *Nature* 2015

Outline

- Deep Q-learning: learn an MMSE estimate of Q(s,a)
  - Off-policy vs. On-policy
  - Batch learning (experience replay)
- Policy learning: learn the policy with the maximum value
  - Estimating the value: actor-critic network
  - Estimating the relative value: advantage actor-critic
- Imitation learning: MMSE imitation of a human expert
  - A good way to initialize Q-learning or policy learning

Policy learning methods

- Suppose that $\vec{s}$ is continuous, but $\vec{a}$ is discrete (e.g., a one-hot vector).
- Then learning the policy directly can be much faster than learning Q values.
- We can train a neural network for a stochastic policy---a policy that chooses an action at random, using the probability distribution:

$$\pi^*(\vec{s}, \vec{a}) = \frac{e^{f(\vec{s},\vec{a})}}{\sum_{\vec{a}\prime} e^{f(\vec{s},\vec{a}\prime)}}$$

How do we train $\pi^*(\vec{s}, \vec{a})$?

- MMSE loss doesn't work very well, b/c the true optimum policy is a one-hot vector (choose the best action w/probability=1.0).
- Cross-entropy ($-\log \pi^*$ of the best action) is possible, if we know what the best action is. Usually, we don't.
- Let's propose a new learning criterion: learn the π^* that maximizes your expected total reward.

How do we train $\pi^*(\vec{s}, \vec{a})$?

- Expected total reward = Bellman's utility, $U(\vec{s})$.
- If we always choose the best action, then

$$U(\vec{s}) = \max_{\vec{a}} Q(\vec{s}, \vec{a})$$

- With a stochastic policy, the utility of state $\vec{s}$ is suboptimal, given by:

$$U^{\pi}(\vec{s}) = \sum_{\vec{a}} \pi^*(\vec{s}, \vec{a}) Q(\vec{s}, \vec{a})$$

- If we knew $Q(\vec{s}, \vec{a})$, then we'd learn $\pi^*(\vec{s}, \vec{a})$ to maximize $U^{\pi}(\vec{s})$. Unfortunately, we don't…

Outline

- Deep Q-learning: learn an MMSE estimate of Q(s,a)
  - Off-policy vs. On-policy
  - Batch learning (experience replay)
- Policy learning: learn the policy with the maximum value
  - Estimating the value: actor-critic network
  - Estimating the relative value: advantage actor-critic
- Imitation learning: MMSE imitation of a human expert
  - A good way to initialize Q-learning or policy learning

Actors from the Comédie Française, by Antoine Watteau, 1720. Public Domain, https://commons.wikimedia.org/w/index.php?curid=15418670

Actor-critic algorithm

So let's train two neural nets!

- $Q^*(\vec{s}, \vec{a})$ is the critic, and is trained according to the deep Q-learning algorithm (MMSE).
- $\pi^*(\vec{s}, \vec{a})$ is the actor, and is trained to satisfy the critic:

The Critic, by Lajos Tihanyi. Oil on canvas, 1916. Public Domain, https://commons.wikimedia.org/w/index.php?curid=17837438

$$U^*(\vec{s}) = \sum_{\vec{a}} \pi^*(\vec{s}, \vec{a}) Q^*(\vec{s}, \vec{a})$$

Actor-Critic Algorithm

- Benefits of the actor-critic algorithm
  - It usually converges faster and more reliably than deep Q-learning, because the softmax probability $\pi^*(\vec{s}, \vec{a})$ is usually easier to learn than the real-valued function $Q^*(\vec{s}, \vec{a})$
- Disadvantages of the actor-critic algorithm
  - …but sometimes, it doesn't. If $Q^*(\vec{s}, \vec{a})$ is estimated badly enough, then it will give wrong information to $\pi^*(\vec{s}, \vec{a})$, and so $\pi^*(\vec{s}, \vec{a})$ will learn a bad policy.

Outline

- Deep Q-learning: learn an MMSE estimate of Q(s,a)
  - Off-policy vs. On-policy
  - Batch learning (experience replay)
- Policy learning: learn the policy with the maximum value
  - Estimating the value: actor-critic network
  - Estimating the relative value: advantage actor-critic
- Imitation learning: MMSE imitation of a human expert
  - A good way to initialize Q-learning or policy learning

Advantage actor-critic

- $Q^*(\vec{s}, \vec{a})$ is hard to learn, in part, because it has such a huge dynamic range. Some states are really good, some are really bad. We can reduce the dynamic range by just learning the ***relative advantage*** of one action over all of the others:

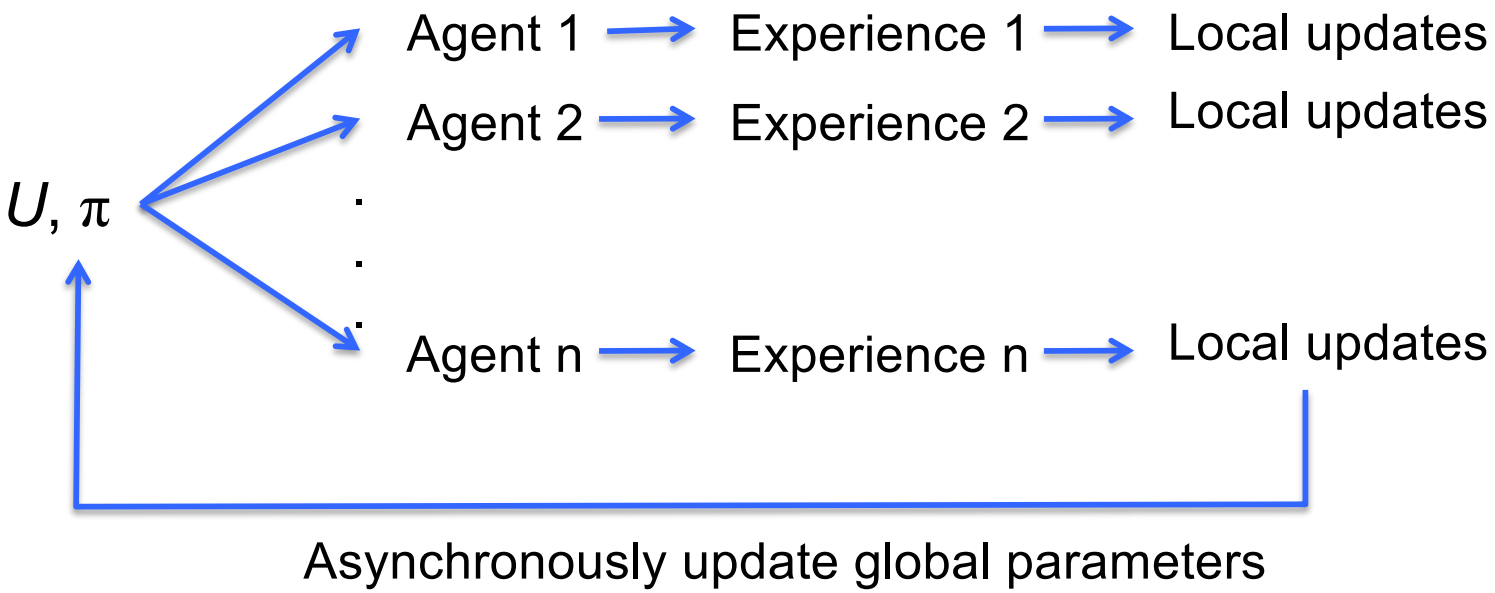
$$A^*(\vec{s}, \vec{a}) = Q^*(\vec{s}, \vec{a}) - U^*(\vec{s}\,)$$

- Now we can train the policy network in order to maximize the ***relative*** utility, which converges faster, and is more accurate:

$$\sum_{\vec{a}} \pi^*(\vec{s}, \vec{a}) A^*(\vec{s}, \vec{a})$$

- But there's computational cost. The only way to learn $A^*(\vec{s}, \vec{a})$ is deep Q-learning (MMSE), which uses Q_{local} to update the weights of $A^*(\vec{s}, \vec{a}) + U^*(\vec{s})$. So we need to train three neural nets: $\pi^*(\vec{s}, \vec{a})$, $A^*(\vec{s}, \vec{a})$, and $U^*(\vec{s}\,)$.

Asynchronous advantage actor-critic (A3C)



Mnih et al. Asynchronous Methods for Deep Reinforcement Learning. ICML 2016

Asynchronous advantage actor-critic (A3C)

TORCS car racing simulation video

Mnih et al. Asynchronous Methods for Deep Reinforcement Learning. ICML 2016

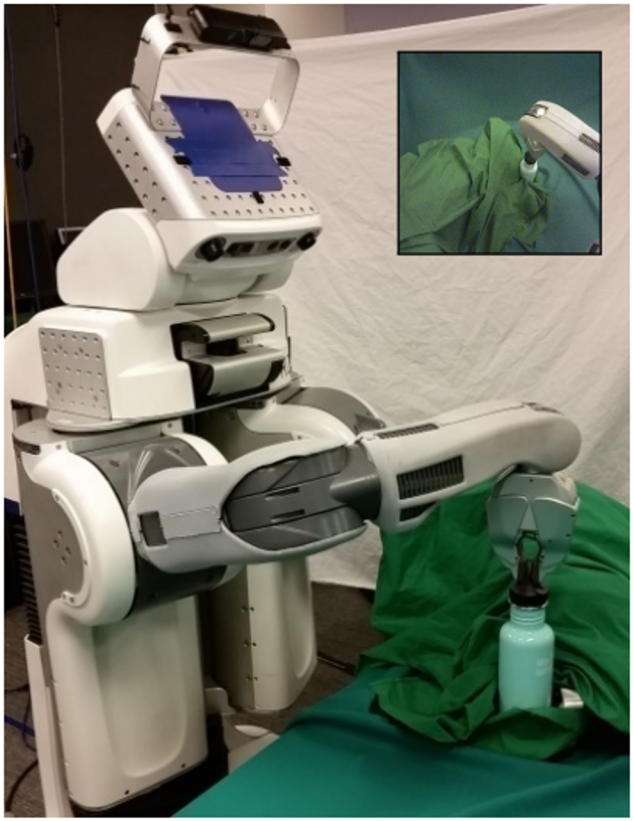
Outline

- Deep Q-learning: learn an MMSE estimate of Q(s,a)
  - Off-policy vs. On-policy
  - Batch learning (experience replay)
- Policy learning: learn the policy with the maximum value
  - Estimating the value: actor-critic network
  - Estimating the relative value: advantage actor-critic
- Imitation learning: MMSE imitation of a human expert
  - A good way to initialize Q-learning or policy learning

Imitation learning

- In some applications, you cannot bootstrap yourself from random policies
  - High-dimensional state and action spaces where most random trajectories fail miserably
  - Expensive to evaluate policies in the physical world, especially in cases of failure
- **Solution:** learn to imitate sample trajectories or demonstrations
  - This is also helpful when there is no natural reward formulation

Learning visuomotor policies

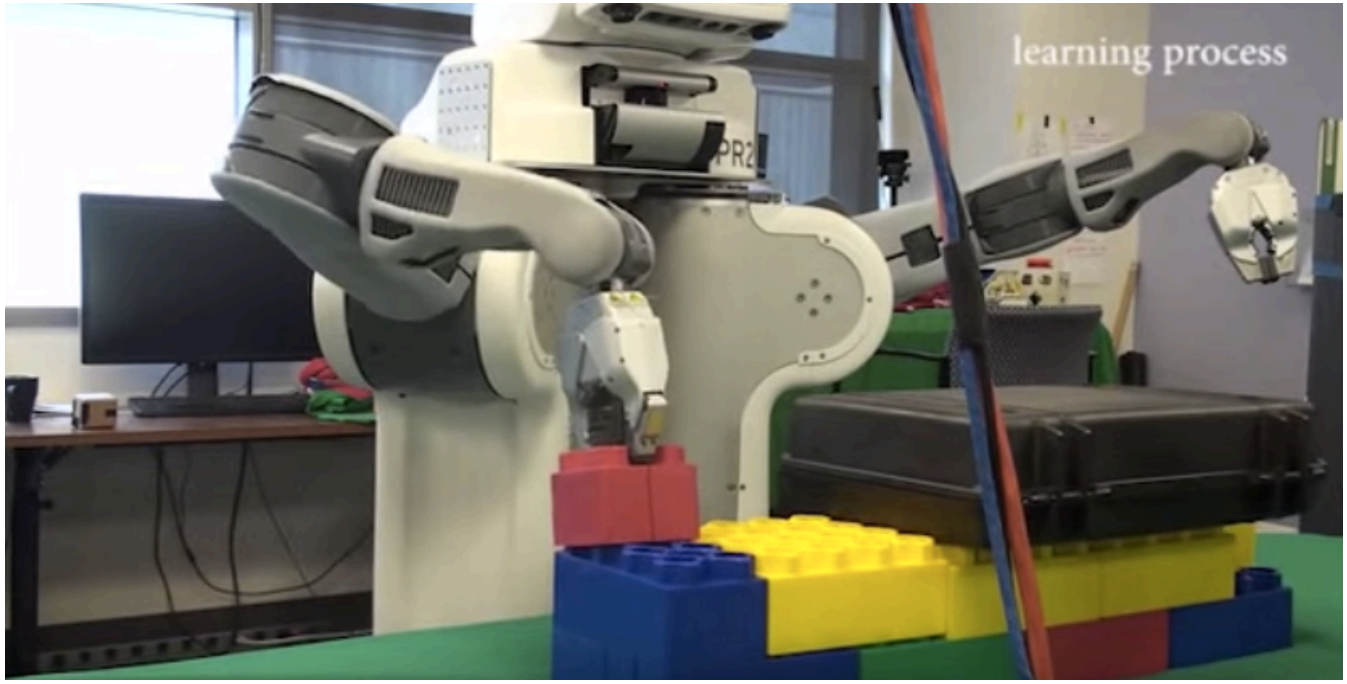


- *Underlying state* x: true object position, robot configuration
- *Observations* o: image pixels

- Two-part approach:
  - Learn *guiding policy* $\pi(a|x)$ using trajectory-centric RL and control techniques
  - Learn *visuomotor policy* $\pi(a|o)$ by imitating $\pi(a|x)$

S. Levine et al. End-to-end training of deep visuomotor policies. JMLR 2016

Learning visuomotor policies



Overview video, training video

S. Levine et al. End-to-end training of deep visuomotor policies. JMLR 2016

Conclusions

1. What is deep Q-learning?
2. How to make Q-learning converge to the best answer?
3. How to make it converge more smoothly?
4. What are policy learning and actor-critic networks?
5. What is imitation learning?

1. Estimate Q(s,a) using a neural net.
2. Epsilon-greedy usually works.
3. Experience replay.
4. Actor network: $\Pr(a)$. Critic network: $Q(s,a)$, to train the actor.
5. Learn to imitate an expert player.