Deep Reinforcement Learning

CS440/ECE448 Lecture 24

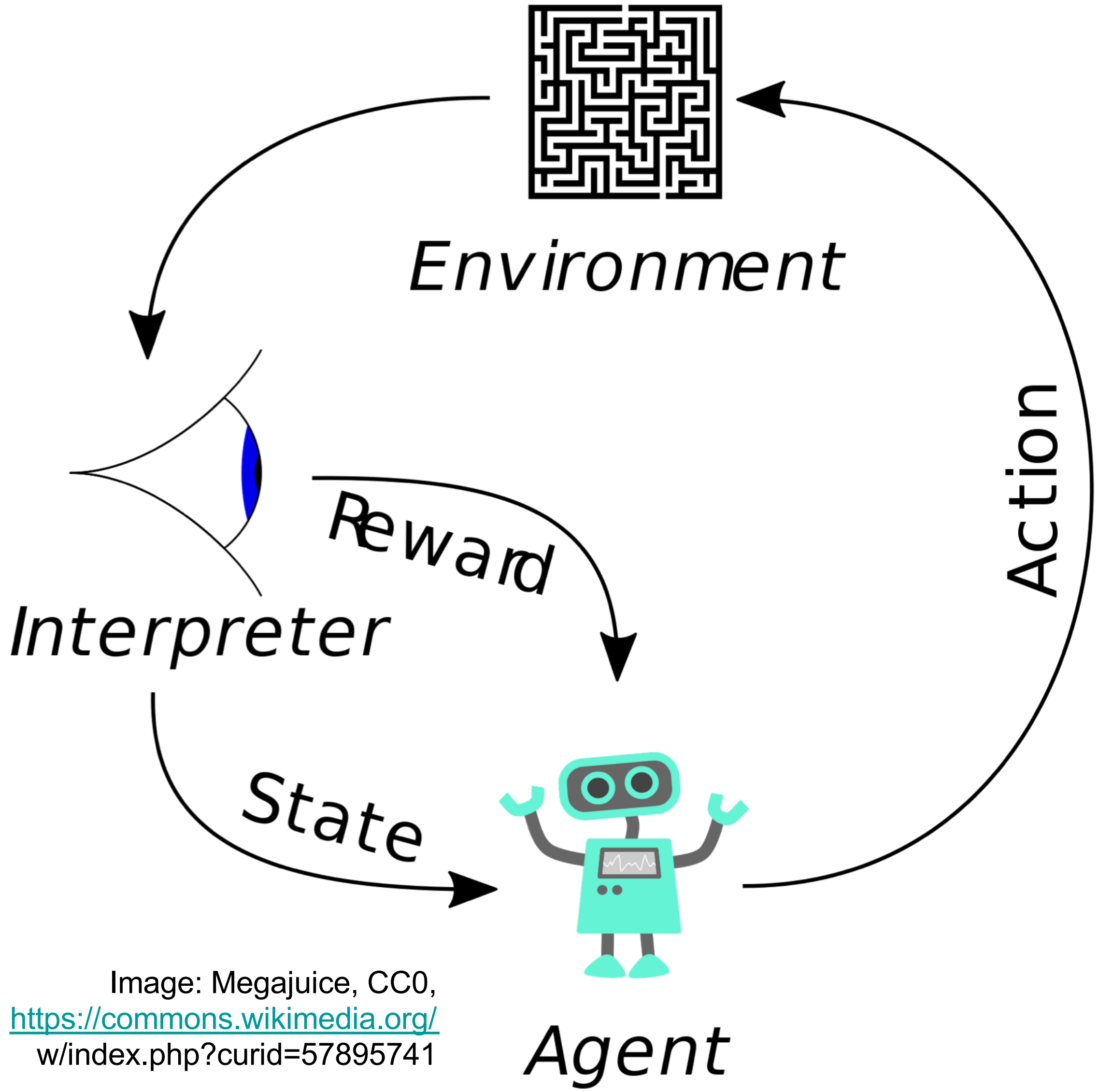


Image: Megajuice, CC0, https://commons.wikimedia.org/w/index.php?curid=57895741

Slides by Svetlana Lazebnik, 11/2017
Modified by Mark Hasegawa-Johnson, 4/2019

Last week: 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

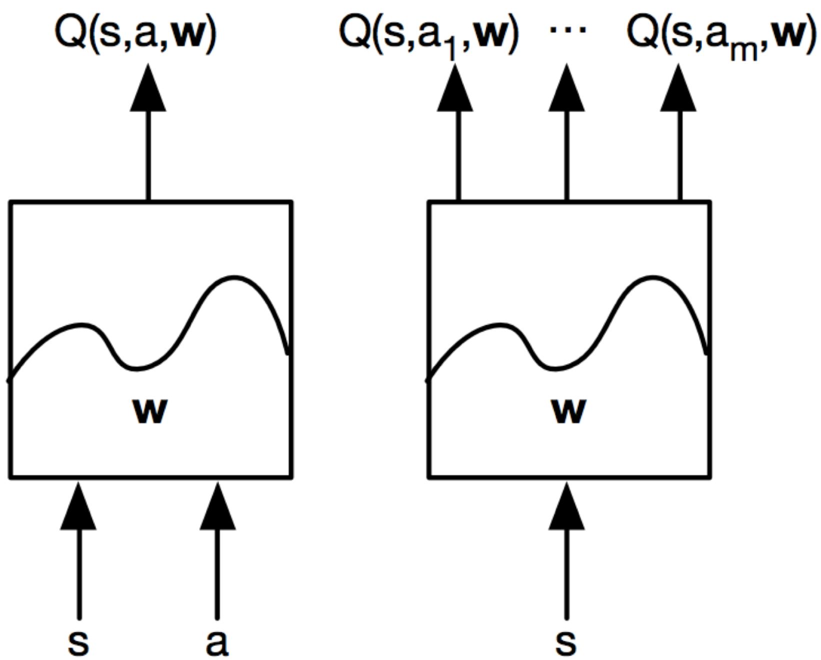
- Approximate $Q(s, a)$ by a ***parameterized function***, that is, by a function $\hat{Q}(s, a; W)$ that depends on some matrix of trainable parameters, *W*.
- Learn W by playing the game.

Outline

- On-line Q-learning
- How to make Q-learning converge to the best answer
- How to make it converge more smoothly
- Policy learning and actor-critic networks
- Imitation learning

Deep Q learning

- Train a deep neural network to output Q values:



Source: D. Silver

Deep Q learning

- SARSA update: "nudge" Q(s,a) toward value we observe it to have in the most recent action:

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

- Deep Q learning: train neural network weights, ***w***, in order to minimize a loss function that penalizes differences between Q(local) and Q(predicted):

$$L(w) = \left(\boxed{R(s) + \gamma \max_{a'} Q(s',a';w)} - \boxed{Q(s,a;w)}\right)^2$$

Q(local):
s'=state you actually reach by performing ***a*** in ***s***,
a'=action you will actually perform there.

Q(predicted):
What the network predicts for action ***a*** in state ***s***

Deep Q learning

- Regular TD update: “nudge” Q(s,a) towards the target

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

- Deep Q learning: encourage estimate to match the target by minimizing squared error:

$$L(w) = \left(\underbrace{\boxed{R(s) + \gamma \max_{a'} Q(s',a';w)}}_{\text{target}} - \underbrace{\boxed{Q(s,a;w)}}_{\text{estimate}}\right)^2$$

- Compare to supervised learning:

$$L(w) = \left(y - f(x;w)\right)^2$$

  – **<u>Key difference</u>**: the target in Q learning is not fixed – (s’,a’) is just one step ahead of (s,a)!

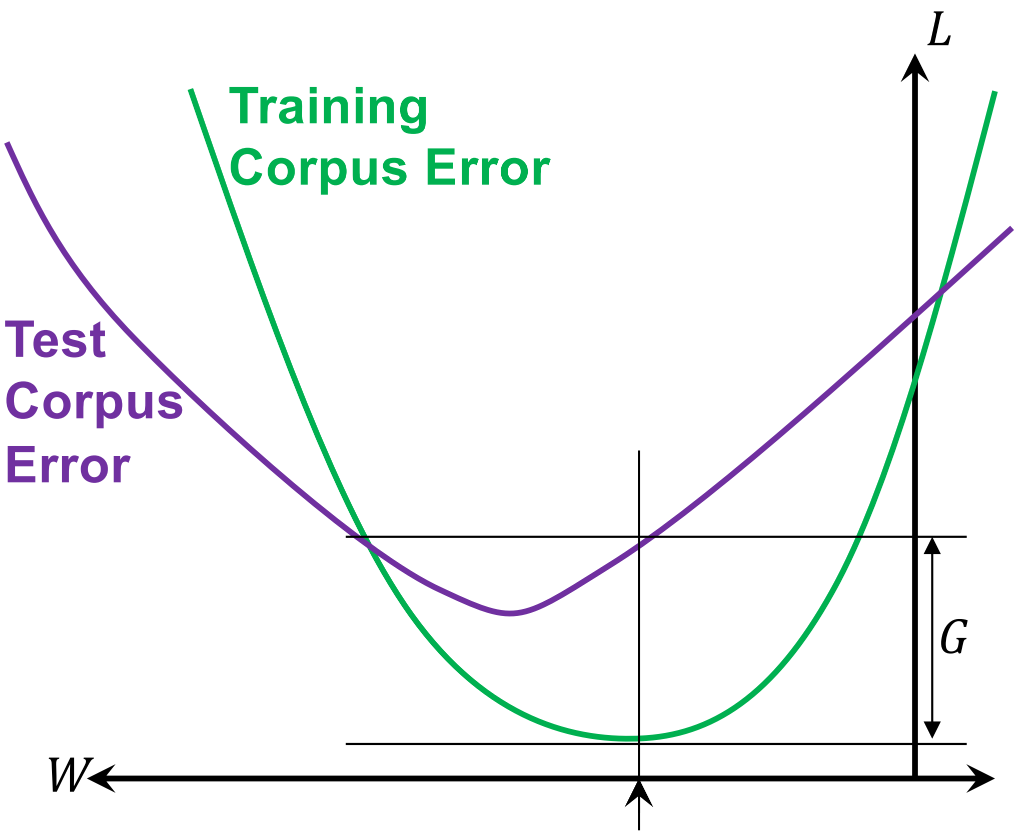
Online Q learning algorithm

- In state ***s***, perform action ***a***. Environment sends you to state ***s'***; choose the action ***a'*** that you'll perform there.
- Observe: $Q^{local}(s,a) = R(s) + \gamma \max_{a'} Q(s',a';W)$
- Update weights to reduce the error
$$L(W) = \left(Q^{local} - Q(s,a;W)\right)^2$$
- Gradient:
$$\nabla_W L = \left(Q(s,a;W) - Q^{local}\right)\nabla_W Q$$
- Weight update:
$$W \longleftarrow W - \eta \nabla_W L$$
- This is called *stochastic gradient descent* (SGD)
- "Stochastic" because the training sample (s,a,s',a') was chosen at random by our exploration function

Outline

- On-line Q-learning
- How to make Q-learning converge to the best answer
- How to make it converge more smoothly
- Policy learning and actor-critic networks
- Imitation 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:

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

- 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!

Incorporating exploration (slide from last week)

- **Idea:** explore more in the beginning, become more and more greedy over time
- Standard ("greedy") selection of optimal action:

$$a = \arg\max_{a' \in A(s)} \sum_{s'} P(s' \mid s, a') U(s')$$

- Modified strategy:

$$a = \arg\max_{a' \in A(s)} f\left(\sum_{s'} P(s' \mid s, a') U(s'), N(s, a') \right)$$

exploration function

Number of times we've taken action a' in state s

$$f(u,n) = \begin{cases} R^{+} & \text{if } n < N_e \\ u & \text{otherwise} \end{cases}$$

(optimistic reward estimate)

…but that doesn't work either:

$$f(u,n) = \begin{cases} R^{+} & \text{if } n < N_e \\ u & \text{otherwise} \end{cases}$$

- … which means that we get at least N_e samples of each action
- We can estimate Q(s,a) based on N_e samples
- But N_e is a constant, so it never $\rightarrow \infty$
- So Error never $\rightarrow 0$

Epsilon-greedy exploration

- At each time step:
  - With probability ϵ, choose an action at random
  - With probability $1 - \epsilon$, choose a=argmax Q(s,a)
  - As $\mathrm{n} \rightarrow \infty, \epsilon \rightarrow 0$, 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

- On-line Q-learning
- How to make Q-learning converge to the best answer
- How to make it converge more smoothly
- Policy learning and actor-critic networks
- Imitation 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*

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

Experience replay

- At each time step:
  - Take action a_t according to epsilon-greedy policy
  - Store experience (s_t, a_t, r_{t+1}, s_{t+1}) in *replay memory buffer*
  - Randomly sample *mini-batch* of experiences from the buffer

s_1, a_1, r_2, s_2
s_2, a_2, r_3, s_3
s_3, a_3, r_4, s_4
$\ldots$
$s_t, a_t, r_{t+1}, s_{t+1}$

$s_t, a_t, r_{t+1}, s_{t+1} \rightarrow$

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

Experience replay

At each time step:

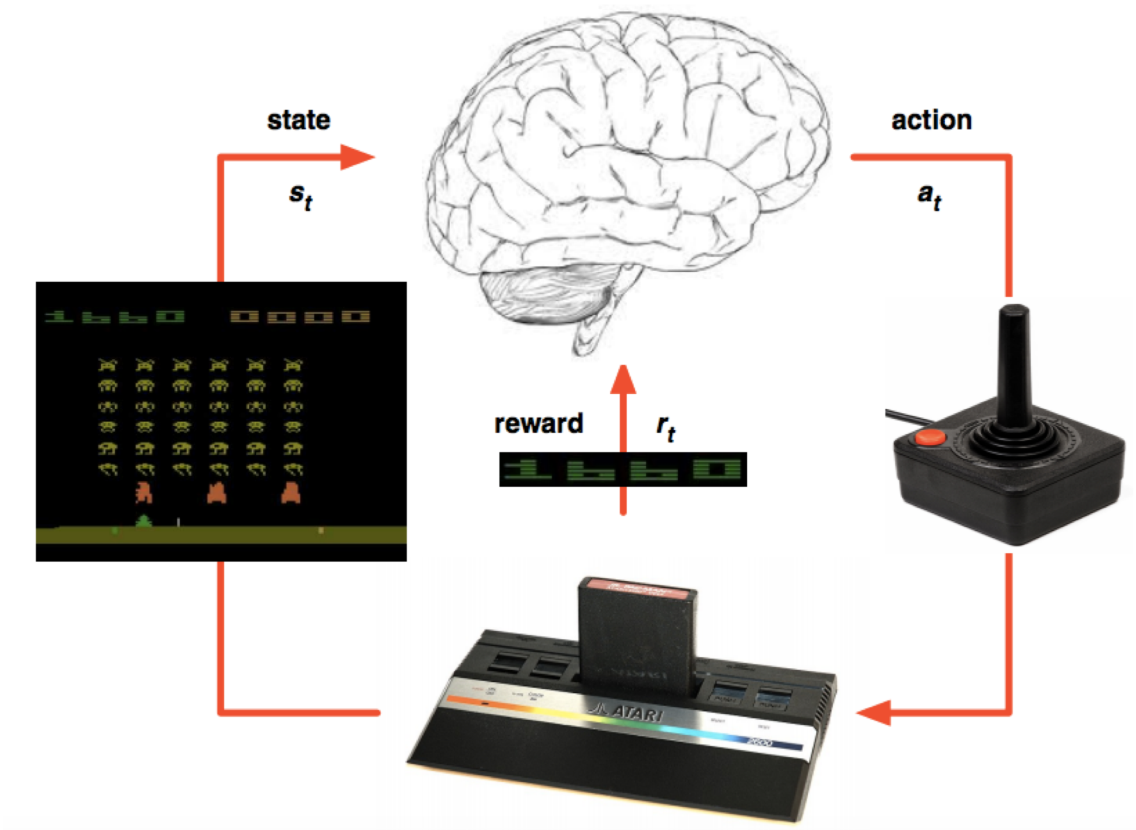
- Take action a_t according to epsilon-greedy policy
- Store experience (s_t, a_t, r_{t+1}, s_{t+1}) in *replay memory buffer*
- Randomly sample *mini-batch* of experiences from the buffer
- Perform update to reduce objective function

$$\mathbf{E}_{s,a,s'}\left[\left(R(s)+\gamma\max_{a'}\boxed{Q(s',a';w^-)}-Q(s,a;w)\right)^2\right]$$

Keep parameters of *target network* fixed during the entire mini-batch; only update between mini-batches

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

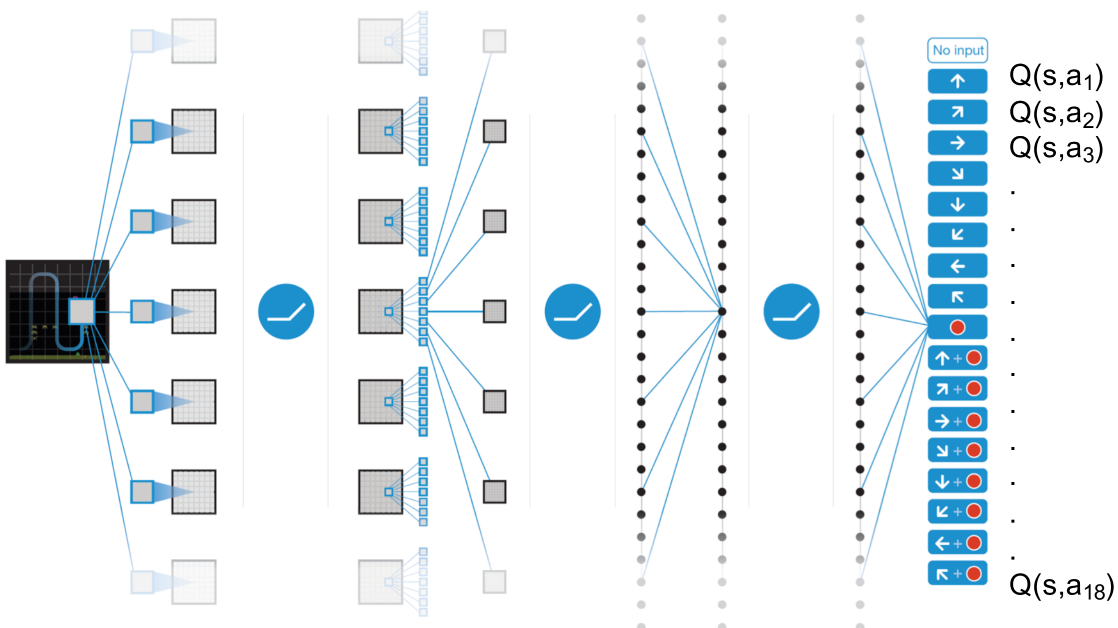
Deep Q learning in Atari



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

Deep Q learning in Atari

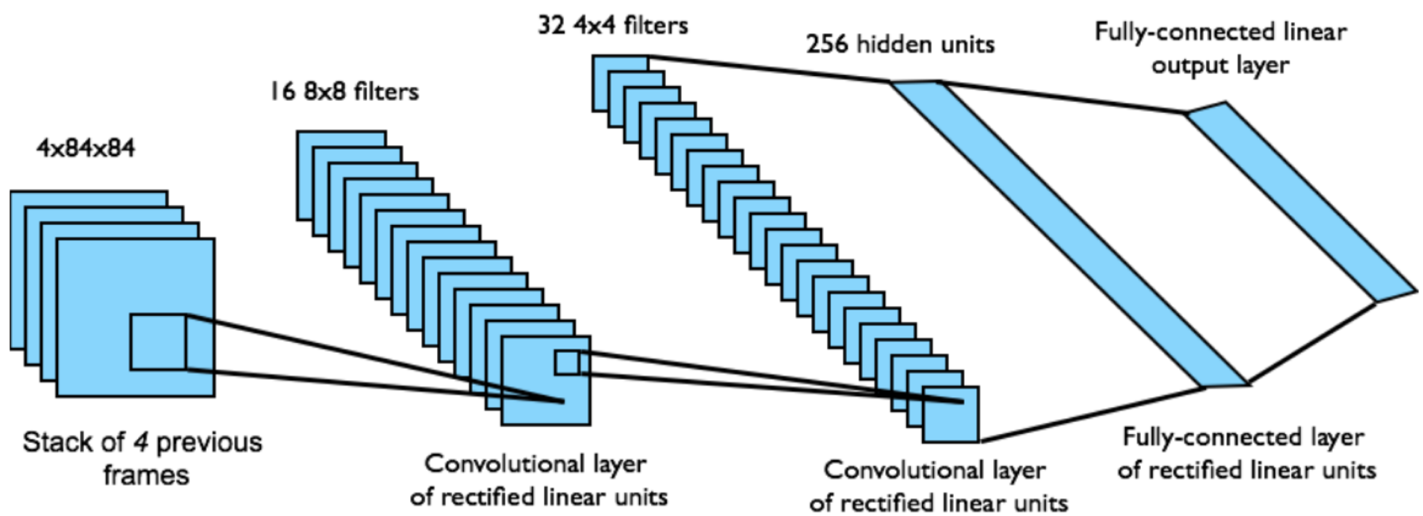
- End-to-end learning of Q(s,a) from pixels s
- Output is Q(s,a) for 18 joystick/button configurations
- Reward is change in score for that step



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

- On-line Q-learning
- How to make Q-learning converge to the best answer
- How to make it converge more smoothly
- Policy learning and actor-critic networks
- Imitation learning

Policy gradient methods

- Learning the policy directly can be much simpler than learning Q values
- We can train a neural network to output *stochastic policies*, or probabilities of taking each action in a given state
- *Softmax* policy:

$$\pi(s,a;u) = \frac{\exp\big(f(s,a;u)\big)}{\sum_{a'} \exp\big(f(s,a';u)\big)}$$

Policy gradient: the softmax function

- Notice that the softmax is normalized so that

$$\pi(s, a; u) \geq 0, \text{ and } \sum_a \pi(s, a; u) = 1$$

- So we can interpret $\pi(s, a; w)$ as some kind of probability. Something like "the probability that a is the best action to take from state s."
- In reality, there is no such probability. There is just one correct action. But the agent doesn't know what it is! So $\pi(s, a; u)$ is kind of like the agent's "degree of belief" that a is the best action (determined by parameters u).

Actor-critic algorithm

- Remember the relationship between the utility of a state, and the quality of an action:

$$U(s) = \max_a Q(s,a)$$

- If we don't know which action is best, then we could say that

$$U(s) \approx \sum_a \pi(s,a;u)Q(s,a;w)$$

- $\pi(s,a;u)$ is the "actor:" a neural net that tells the agent how to act.
- $Q(s,a;w)$ is the "critic:" a neural net that tells the agent how good or bad that action was.

Actor-critic algorithm

- Define objective function as total discounted reward:

$$J(u) = \mathbf{E}\left[R_1 + \gamma R_2 + \gamma^2 R_3 + \ldots\right]$$

- The gradient for a stochastic policy is given by

$$\nabla_u J = \mathbf{E}\left[\nabla_u \log \underbrace{\boxed{\pi(s,a;u)}}_{\text{Actor network}} \underbrace{\boxed{Q^\pi(s,a;w)}}_{\text{Critic network}}\right]$$

- Actor network update: $u \leftarrow u + \alpha \nabla_u J$
- Critic network update: use Q learning (following actor's policy)

Advantage actor-critic

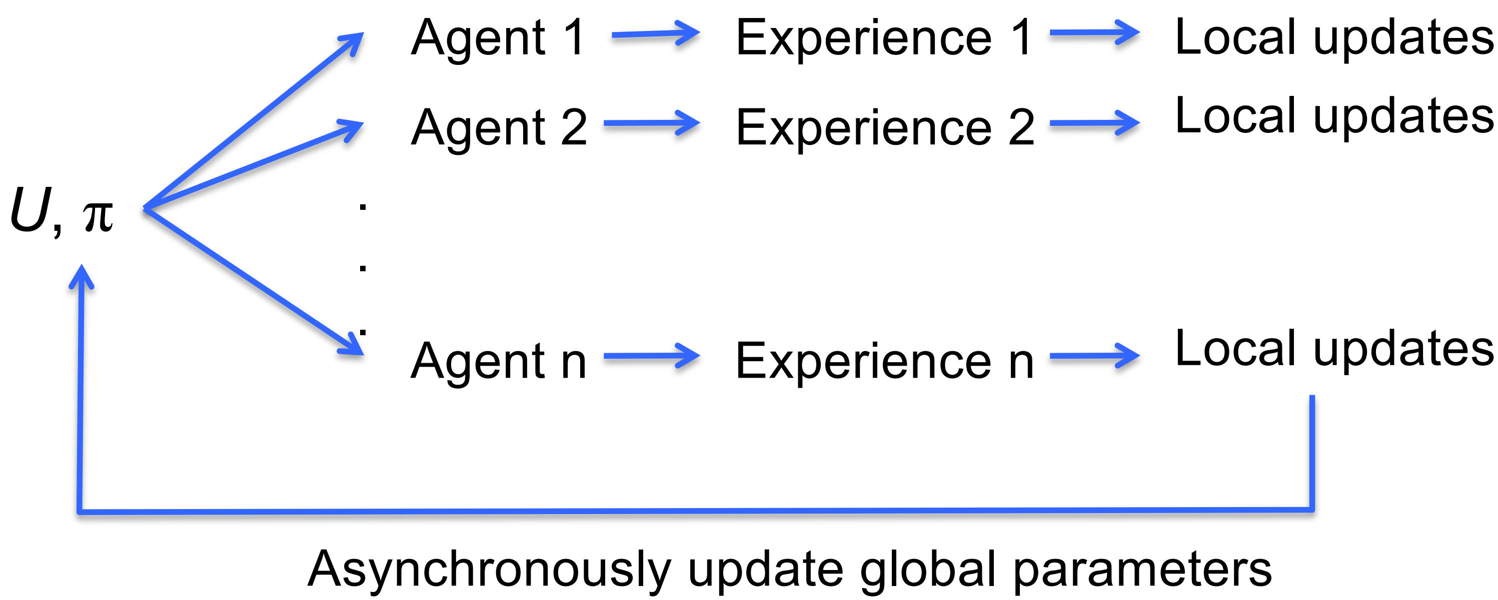
- The raw Q value is less meaningful than whether the reward is better or worse than what you expect to get
- Introduce an *advantage function* that subtracts a baseline number from all Q values

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s)$$

  - Estimate *V* using a *value network*
- Advantage actor-critic:

$$\nabla_u J = \mathbf{E}\left[\nabla_u \log \pi(s,a;u) A^{\pi}(s,a;w)\right]$$

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

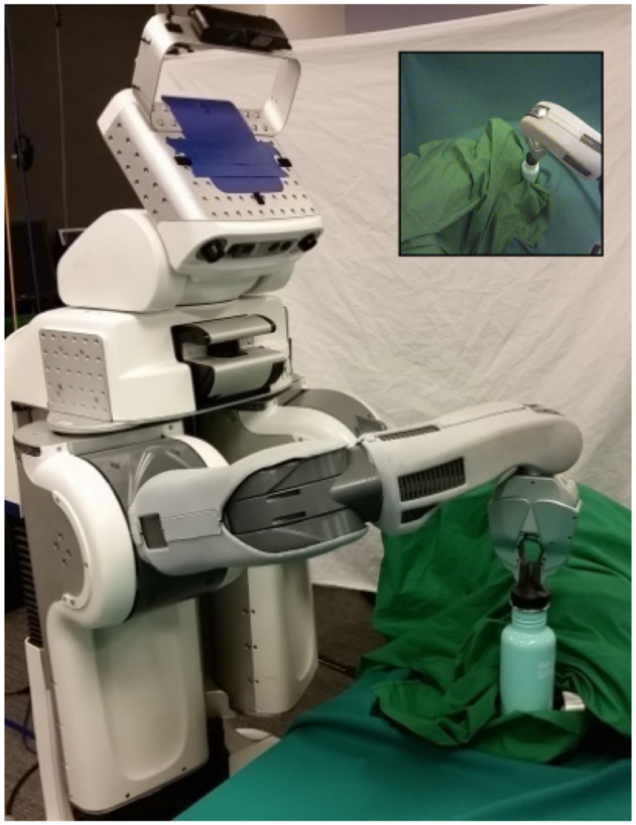
- On-line Q-learning
- How to make Q-learning converge to the best answer
- How to make it converge more smoothly
- Policy learning and actor-critic networks
- Imitation learning

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

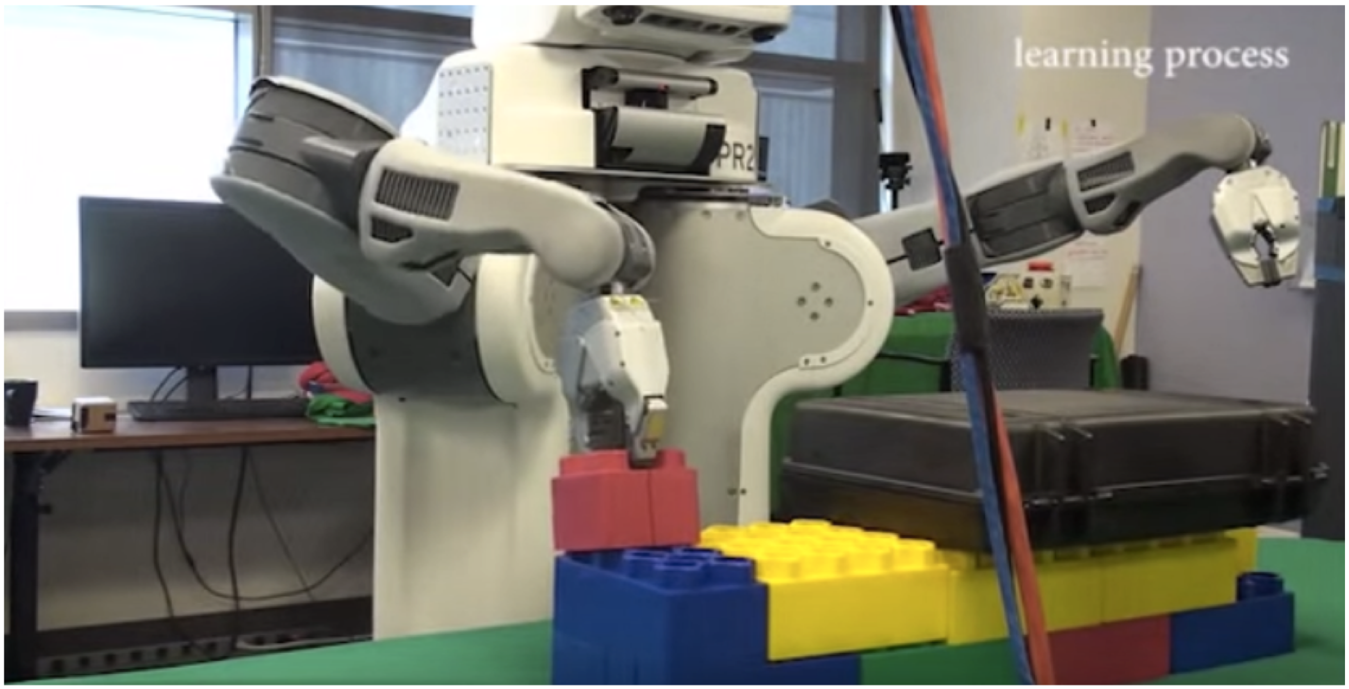


- *Underlying state* x: true object position, robot configuration
- *Observations* o: image pixels

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