Deep reinforcement learning
Function approximation

• 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

• Alternative idea: approximate the utilities or $Q$ values using parametric functions and automatically learn the parameters:

\[
V(s) \approx \hat{V}(s; w) \\
Q(s, a) \approx \hat{Q}(s, a; w)
\]
Deep Q learning

- Train a deep neural network to output Q values:

\[ Q(s, a, w) \quad Q(s, a_1, w) \quad \cdots \quad Q(s, a_m, w) \]
Deep Q learning

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

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

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

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

  target  estimate
Deep Q learning

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

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

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

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

• Compare to supervised learning:

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

– Key difference: the target in Q learning is also moving!
Online Q learning algorithm

• Observe experience \((s, a, s')\)
• Compute target  \(y = R(s) + \gamma \max_a Q(s', a'; w)\)
• Update weights to reduce the error

\[
L = (y - Q(s, a; w))^2
\]

• Gradient:  \(\nabla_w L = (Q(s, a; w) - y) \nabla_w Q(s, a; w)\)

• Weight update:  \(w \leftarrow w - \alpha \nabla_w L\)

• This is called stochastic gradient descent (SGD)
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

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

\[
E_{s,a,s'} \left[ \left( R(s) + \gamma \max_{a'} Q(s', a'; w^-) - Q(s, a; w) \right)^2 \right]
\]

Keep parameters of *target network* fixed, update every once in a while

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
Deep Q learning in Atari

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

https://www.youtube.com/watch?v=TmPfTpjtdgg
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*:

\[
\pi(s, a; u) = \frac{\exp(f(s, a; u))}{\sum_{a'} \exp(f(s, a'; u))}
\]
Actor-critic algorithm

- Define objective function as total discounted reward:
  \[ J(u) = \mathbb{E}\left[R_1 + \gamma R_2 + \gamma^2 R_3 + \ldots\right] \]

- The gradient for a stochastic policy is given by
  \[ \nabla_u J = \mathbb{E}\left[\nabla_u \log\pi(s, a; u)Q^\pi(s, a; w)\right] \]

- 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

\[ A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s) \]

  \[ A^\pi(s,a) \approx R(s) + \gamma V^\pi(s') - V^\pi(s) \]

  – Estimate V using a value network

• Advantage actor-critic:

\[ \nabla_u J = \mathbb{E}\left[ \nabla_u \log \pi(s,a;u) A^\pi(s,a;w) \right] \]
Asynchronous advantage actor-critic (A3C)

Agent 1 $\rightarrow$ Experience 1 $\rightarrow$ Local updates
Agent 2 $\rightarrow$ Experience 2 $\rightarrow$ Local updates
\[ \cdots \]
Agent n $\rightarrow$ Experience n $\rightarrow$ Local updates

Asynchronously update global parameters

## Asynchronous advantage actor-critic (A3C)

<table>
<thead>
<tr>
<th>Method</th>
<th>Training Time</th>
<th>Mean</th>
<th>Median</th>
</tr>
</thead>
<tbody>
<tr>
<td>DQN</td>
<td>8 days on GPU</td>
<td>121.9%</td>
<td>47.5%</td>
</tr>
<tr>
<td>Gorila</td>
<td>4 days, 100 machines</td>
<td>215.2%</td>
<td>71.3%</td>
</tr>
<tr>
<td>D-DQN</td>
<td>8 days on GPU</td>
<td>332.9%</td>
<td>110.9%</td>
</tr>
<tr>
<td>Dueling D-DQN</td>
<td>8 days on GPU</td>
<td>343.8%</td>
<td>117.1%</td>
</tr>
<tr>
<td>Prioritized DQN</td>
<td>8 days on GPU</td>
<td>463.6%</td>
<td>127.6%</td>
</tr>
<tr>
<td>A3C, FF</td>
<td>1 day on CPU</td>
<td>344.1%</td>
<td>68.2%</td>
</tr>
<tr>
<td>A3C, FF</td>
<td>4 days on CPU</td>
<td>496.8%</td>
<td>116.6%</td>
</tr>
<tr>
<td>A3C, LSTM</td>
<td>4 days on CPU</td>
<td>623.0%</td>
<td>112.6%</td>
</tr>
</tbody>
</table>

Mean and median human-normalized scores over 57 Atari games

Asynchronous advantage actor-critic (A3C)

TORCS car racing simulation video

Asynchronous advantage actor-critic (A3C)

Motor control tasks video

Playing Go

• Go is a known (and deterministic) environment
• Therefore, learning to play Go involves solving a known MDP
• Key challenges: huge state and action space, long sequences, sparse rewards
Review: AlphaGo

- **Policy network:** initialized by supervised training on large amount of human games
- **Value network:** trained to predict outcome of game based on self-play
- Networks are used to guide Monte Carlo tree search (MCTS)

D. Silver et al., Mastering the Game of Go with Deep Neural Networks and Tree Search, Nature 529, January 2016
AlphaGo Zero

- A fancier architecture (deep residual networks)
- No hand-crafted features at all
- A single network to predict both value and policy
- Train network entirely by self-play, starting with random moves
- Uses MCTS inside the reinforcement learning loop, not outside

D. Silver et al., Mastering the Game of Go without Human Knowledge, Nature 550, October 2017
https://deepmind.com/blog/alphago-zero-learning-scratch/
AlphaGo Zero

- Given a position, neural network outputs both move probabilities $P$ and value $V$ (probability of winning)
- In each position, MCTS is conducted to return refined move probabilities $\pi$ and game winner $Z$
- Neural network parameters are updated to make $P$ and $V$ better match $\pi$ and $Z$
- Reminiscent of policy iteration: self-play with MCTS is policy evaluation, updating the network towards MCTS output is policy improvement

D. Silver et al., Mastering the Game of Go without Human Knowledge, Nature 550, October 2017
https://deepmind.com/blog/alphago-zero-learning-scratch/
AlphaGo Zero

D. Silver et al., Mastering the Game of Go without Human Knowledge, Nature 550, October 2017
https://deepmind.com/blog/alphago-zero-learning-scratch/
AlphaGo Zero

It’s also more efficient than older engines!

D. Silver et al., Mastering the Game of Go without Human Knowledge, Nature 550, October 2017
https://deepmind.com/blog/alphago-zero-learning-scratch/
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

Neural network architecture

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
Summary

• Deep Q learning
• Policy gradient methods
  – Actor-critic
  – Advantage actor-critic
  – A3C
• Policy iteration for AlphaGo
• Imitation learning for visuomotor policies