Tutorial: Deep Reinforcement Learning

David Silver, Google DeepMind

Outline

Reinforcement Learning: AI = RL

- RL is a general-purpose framework for artificial intelligence
- We seek a single agent which can solve any human-level task
- The essence of an intelligent agent
- Powerful RL requires powerful representations

Outline

Deep Representations

- A deep representation is a composition of many functions

$$x \xrightarrow[w_1]{} h_1 \xrightarrow[w_2]{} \dots \xrightarrow[w_n]{} h_n \xrightarrow[w_{n+1}]{} y$$

- Its gradient can be backpropagated by the chain rule

$$\begin{tikzcd}
\frac{\partial h_1}{\partial x} & \frac{\partial h_2}{\partial h_1} \arrow[l] \arrow[d] & \dots \arrow[l] & \frac{\partial y}{\partial h_n} \arrow[l] \arrow[d] & \frac{\partial}{\partial y} \arrow[l] \arrow[d] \\
 & \frac{\partial h_1}{\partial w_1} & \dots & \frac{\partial h_n}{\partial w_n} & \frac{\partial y}{\partial w_{n+1}}
\end{tikzcd}$$

Deep Neural Network

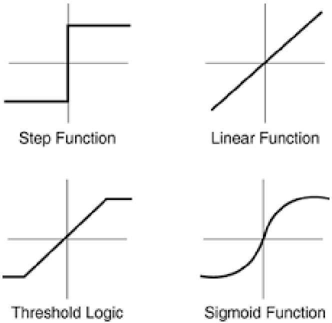
A deep neural network is typically composed of:

- Linear transformations

$$h_{k+1} = Wh_k$$

- Non-linear activation functions

$$h_{k+2} = f(h_{k+1})$$

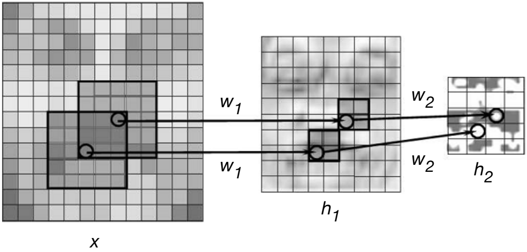


Weight Sharing

Recurrent neural network shares weights between time-steps

$$\begin{tikzcd}
 & y_1 & y_2 & \ldots & y_n \\
h_0 \arrow[r, "w"'] & h_1 \arrow[u] \arrow[r, "w"'] & h_2 \arrow[u] \arrow[r, "w"'] & \ldots \arrow[r, "w"'] & h_n \arrow[u] \\
 & x_1 \arrow[u] & x_2 \arrow[u] & \ldots & x_n \arrow[u]
\end{tikzcd}$$

Convolutional neural network shares weights between local regions



Loss Function

- A loss function $l(y)$ measures goodness of output y, e.g.
  - Mean-squared error $l(y) = ||y^* - y||^2$
  - Log likelihood $l(y) = \log \mathbb{P}[y^*|x]$
- The loss is appended to the forward computation

$$x \xrightarrow[w_1]{} h_1 \xrightarrow[w_2]{} \dots \xrightarrow[w_n]{} h_n \xrightarrow[w_{n+1}]{} y \longrightarrow l(y)$$

- Gradient of loss is appended to the backward computation

$$\begin{tikzcd}
\frac{\partial h_1}{\partial x} & \frac{\partial h_2}{\partial h_1} \arrow[l] \arrow[d] & \dots \arrow[l] & \frac{\partial y}{\partial h_n} \arrow[l] \arrow[d] & \frac{\partial l(y)}{\partial y} \arrow[l] \arrow[d] \\
 & \frac{\partial h_1}{\partial w_1} & \dots & \frac{\partial h_n}{\partial w_n} & \frac{\partial y}{\partial w_{n+1}}
\end{tikzcd}$$

Stochastic Gradient Descent

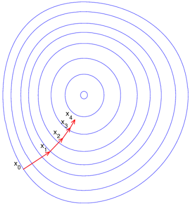
- Minimise expected loss $\mathcal{L}(w) = \mathbb{E}_x\left[l(y)\right]$
- Follow the gradient of $\mathcal{L}(w)$

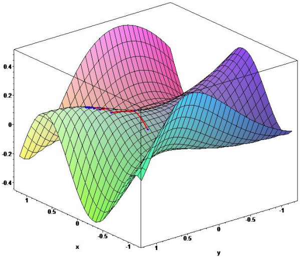
$$\frac{\partial \mathcal{L}(w)}{\partial w} = \mathbb{E}_x\left[\frac{\partial l(y)}{\partial w}\right] = \mathbb{E}_x \begin{pmatrix} \frac{\partial l(y)}{\partial w^{(1)}} \\ \vdots \\ \frac{\partial l(y)}{\partial w^{(k)}} \end{pmatrix}$$

- Adjust w in direction of -ve gradient

$$\Delta w = -\frac{\alpha}{2}\alpha\frac{\partial l(y)}{\partial w}$$

where α is a step-size parameter





Deep Supervised Learning

- Deep neural networks have achieved remarkable success
- Simple ingredients solve supervised learning problems
  - Use deep network as a function approximator
  - Define loss function
  - Optimise parameters end-to-end by SGD
- Scales well with memory/data/computation
- Solves the representation learning problem
- State-of-the-art for images, audio, language, ...

Deep Supervised Learning

- Deep neural networks have achieved remarkable success
- Simple ingredients solve supervised learning problems
  - Use deep network as a function approximator
  - Define loss function
  - Optimise parameters end-to-end by SGD
- Scales well with memory/data/computation
- Solves the representation learning problem
- State-of-the-art for images, audio, language, ...
- Can we follow the same recipe for RL?

Outline

Policies and Value Functions

- Policy π is a behaviour function selecting actions given states

$$a = \pi(s)$$

- Value function $Q^\pi(s, a)$ is expected total reward from state s and action a under policy π

$$Q^\pi(s, a) = \mathbb{E}\left[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... \mid s, a\right]$$

"How good is action a in state s?"

Approaches To Reinforcement Learning

Policy-based RL

- Search directly for the optimal policy π^*
- This is the policy achieving maximum future reward

Value-based RL

- Estimate the optimal value function $Q^*(s, a)$
- This is the maximum value achievable under any policy

Approaches To Reinforcement Learning

Policy-based RL

- Search directly for the optimal policy π^*
- This is the policy achieving maximum future reward

Value-based RL

- Estimate the optimal value function $Q^*(s, a)$
- This is the maximum value achievable under any policy

Model-based RL

- Build a transition model of the environment
- Plan (e.g. by lookahead) using model

Deep Reinforcement Learning

- Can we apply deep learning to RL?
- Use deep network to represent value function / policy / model
- Optimise value function / policy /model end-to-end
- Using stochastic gradient descent

Outline

Deep Learning

Reinforcement Learning

Deep Value Functions

Deep Policies

Deep Models

Bellman Equation

- Bellman expectation equation unrolls value function Q^π

$$
\begin{aligned}
Q^\pi(s,a) &= \mathbb{E}\left[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots \mid s, a\right] \\
&= \mathbb{E}_{s',a'}\left[r + \gamma Q^\pi(s',a') \mid s, a\right]
\end{aligned}
$$

Bellman Equation

- Bellman expectation equation unrolls value function Q^π

$$\begin{aligned} Q^\pi(s,a) &= \mathbb{E}\left[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots \mid s,a\right] \\ &= \mathbb{E}_{s',a'}\left[r + \gamma Q^\pi(s',a') \mid s,a\right] \end{aligned}$$

- Bellman optimality equation unrolls optimal value function Q^*

$$Q^*(s,a) = \mathbb{E}_{s'}\left[r + \gamma \max_{a'} Q^*(s',a') \mid s,a\right]$$

Bellman Equation

- Bellman expectation equation unrolls value function Q^{π}

$$Q^{\pi}(s,a) = \mathbb{E}\left[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots \mid s,a\right]$$
$$= \mathbb{E}_{s',a'}\left[r + \gamma Q^{\pi}(s',a') \mid s,a\right]$$

- Bellman optimality equation unrolls optimal value function Q^{*}

$$Q^{*}(s,a) = \mathbb{E}_{s'}\left[r + \gamma \max_{a'} Q^{*}(s',a') \mid s,a\right]$$

- Policy iteration algorithms solve Bellman expectation equation

$$Q_{i+1}(s,a) = \mathbb{E}_{s'}\left[r + \gamma\ Q_{i}(s',a') \mid s,a\right]$$

- Value iteration algorithms solve Bellman optimality equation

$$Q_{i+1}(s,a) = \mathbb{E}_{s',a'}\left[r + \gamma \max_{a'} Q_{i}(s',a') \mid s,a\right]$$

Policy Iteration with Non-Linear Sarsa

- Represent value function by Q-network with weights w

$$Q(s, a, w) \approx Q^{\pi}(s, a)$$

Policy Iteration with Non-Linear Sarsa

- Represent value function by Q-network with weights w

$$Q(s, a, w) \approx Q^{\pi}(s, a)$$

- Define objective function by mean-squared error in Q-values

$$\mathcal{L}(w) = \mathbb{E}\left[\left(\underbrace{r + \gamma Q(s', a', w)}_{\text{target}} - Q(s, a, w)\right)^2\right]$$

Policy Iteration with Non-Linear Sarsa

- Represent value function by Q-network with weights w

$$Q(s, a, w) \approx Q^{\pi}(s, a)$$

- Define objective function by mean-squared error in Q-values

$$\mathcal{L}(w) = \mathbb{E}\left[\left(\underbrace{r + \gamma Q(s', a', w)}_{\text{target}} - Q(s, a, w)\right)^2\right]$$

- Leading to the following Sarsa gradient

$$\frac{\partial \mathcal{L}(w)}{\partial w} = \mathbb{E}\left[\left(r + \gamma Q(s', a', w) - Q(s, a, w)\right) \frac{\partial Q(s, a, w)}{\partial w}\right]$$

- Optimise objective end-to-end by SGD, using $\frac{\partial L(w)}{\partial w}$

Value Iteration with Non-Linear Q-Learning

- Represent value function by deep Q-network with weights w

$$Q(s, a, w) \approx Q^{\pi}(s, a)$$

- Define objective function by mean-squared error in Q-values

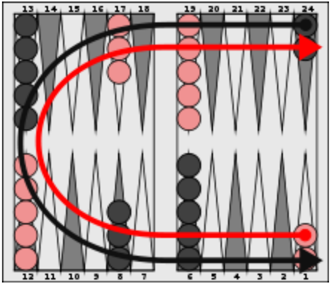
$$\mathcal{L}(w) = \mathbb{E}\left[\left(\underbrace{r + \gamma \max_{a'} Q(s', a', w)}_{\text{target}} - Q(s, a, w)\right)^2\right]$$

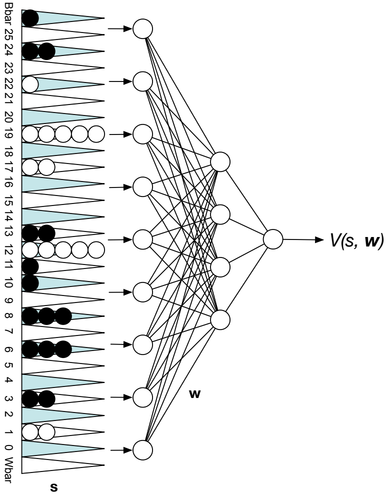
- Leading to the following Q-learning gradient

$$\frac{\partial \mathcal{L}(w)}{\partial w} = \mathbb{E}\left[\left(r + \gamma \max_{a'} Q(s', a', w) - Q(s, a, w)\right)\frac{\partial Q(s, a, w)}{\partial w}\right]$$

- Optimise objective end-to-end by SGD, using $\frac{\partial L(w)}{\partial w}$

Example: TD Gammon





Self-Play Non-Linear Sarsa

- Initialised with random weights
- Trained by games of self-play
- Using non-linear Sarsa with afterstate value function

$$Q(s, a, w) = \mathbb{E}\left[V(s', w)\right]$$

- Greedy policy improvement (no exploration)
- Algorithm converged in practice (not true for other games)

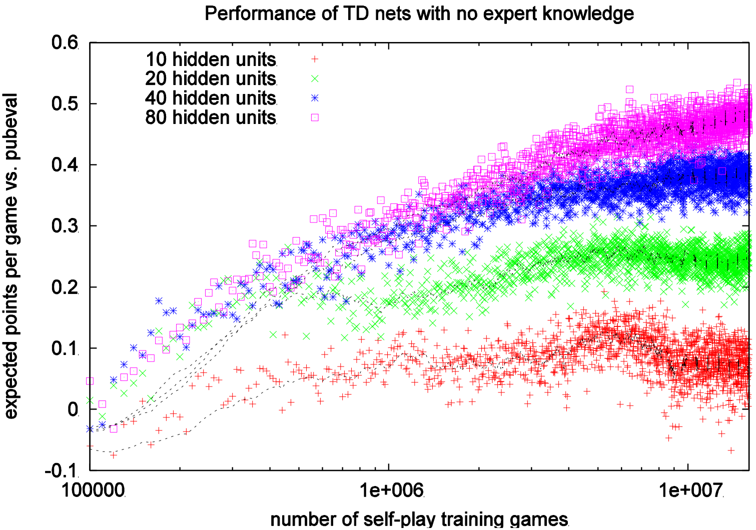
Self-Play Non-Linear Sarsa

- Initialised with random weights
- Trained by games of self-play
- Using non-linear Sarsa with afterstate value function

$$Q(s, a, w) = \mathbb{E}\left[V(s', w)\right]$$

- Greedy policy improvement (no exploration)
- Algorithm converged in practice (not true for other games)
- TD Gammon defeated world champion Luigi Villa 7-1 (Tesauro, 1992)

New TD-Gammon Results



Stability Issues with Deep RL

Naive Q-learning oscillates or diverges with neural nets

1. Data is sequential
   - Successive samples are correlated, non-iid
2. Policy changes rapidly with slight changes to Q-values
   - Policy may oscillate
   - Distribution of data can swing from one extreme to another
3. Scale of rewards and Q-values is unknown
   - Naive Q-learning gradients can be large
     unstable when backpropagated

Deep Q-Networks

DQN provides a stable solution to deep value-based RL

1. Use experience replay
   - Break correlations in data, bring us back to iid setting
   - Learn from all past policies
   - Using off-policy Q-learning
2. Freeze target Q-network
   - Avoid oscillations
   - Break correlations between Q-network and target
3. Clip rewards or normalize network adaptively to sensible range
   - Robust gradients

Stable Deep RL (1): Experience Replay

To remove correlations, build data-set from agent's own experience

- Take action a_t according to ϵ-greedy policy
- Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay memory $\mathcal{D}$
- Sample random mini-batch of transitions (s, a, r, s') from $\mathcal{D}$
- Optimise MSE between Q-network and Q-learning targets, e.g.

$$\mathcal{L}(w) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}} \left[\left(r + \gamma \max_{a'} Q(s', a', w) - Q(s, a, w) \right)^2 \right]$$

Stable Deep RL (2): Fixed Target Q-Network

To avoid oscillations, fix parameters used in Q-learning target

- Compute Q-learning targets w.r.t. old, fixed parameters w^-

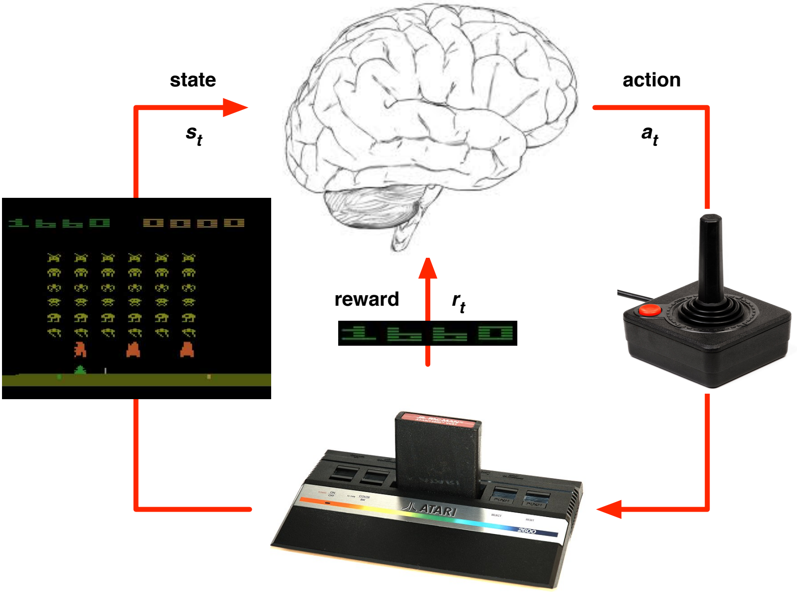
$$r + \gamma \max_{a'} Q(s', a', w^-)$$

- Optimise MSE between Q-network and Q-learning targets

$$\mathcal{L}(w) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}} \left[\left(r + \gamma \max_{a'} Q(s', a', w^-) - Q(s, a, w) \right)^2 \right]$$

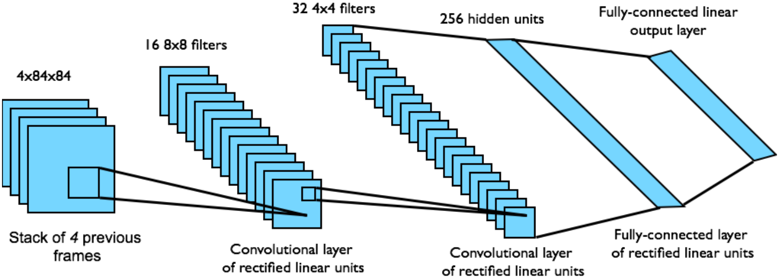
- Periodically update fixed parameters $w^- \leftarrow w$

Reinforcement Learning in Atari



DQN in Atari

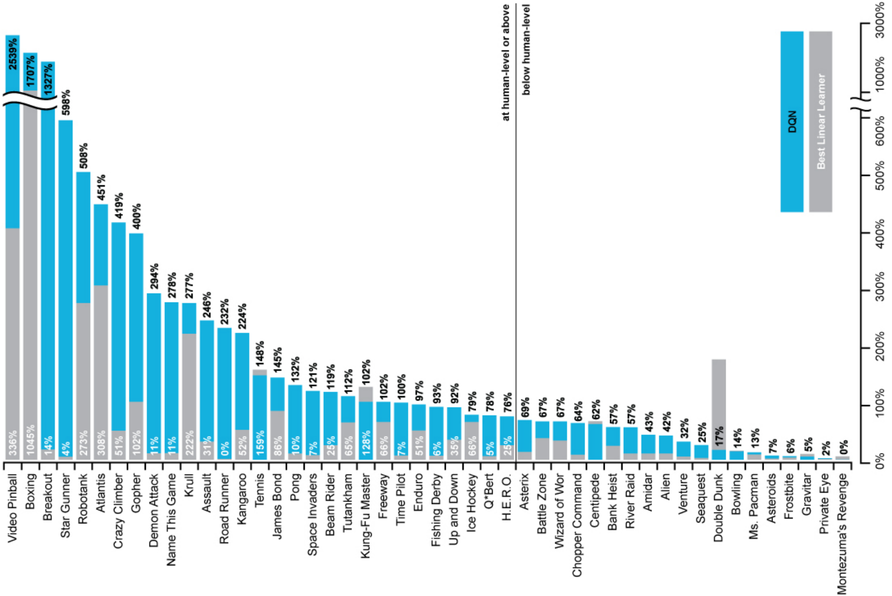
- End-to-end learning of values $Q(s, a)$ from pixels s
- Input state s is stack of raw pixels from last 4 frames
- Output is $Q(s, a)$ for 18 joystick/button positions
- Reward is change in score for that step



Network architecture and hyperparameters fixed across all games

[Mnih et al.]

DQN Results in Atari



DQN Demo

How much does DQN help?

DQN

	Q-learning	Q-learning + Target Q	Q-learning + Replay	Q-learning + Replay + Target Q
Breakout	3	10	241	**317**
Enduro	29	142	831	**1006**
River Raid	1453	2868	4103	**7447**
Seaquest	276	1003	823	**2894**
Space Invaders	302	373	826	**1089**

Stable Deep RL (3): Reward/Value Range

- DQN clips the rewards to $[-1, +1]$
- This prevents Q-values from becoming too large
- Ensures gradients are well-conditioned

Stable Deep RL (3): Reward/Value Range

- DQN clips the rewards to $[-1, +1]$
- This prevents Q-values from becoming too large
- Ensures gradients are well-conditioned
- Can't tell difference between small and large rewards
- Better approach: normalise network output
- e.g. via batch normalisation

Outline

Policy Gradient for Continuous Actions

- Represent policy by deep network $a = \pi(s, u)$ with weights u
- Define objective function as total discounted reward

$$J(u) = \mathbb{E}\left[r_1 + \gamma r_2 + \gamma^2 r_3 + ...\right]$$

- Optimise objective end-to-end by SGD
- i.e. Adjust policy parameters u to achieve more reward

Deterministic Policy Gradient

The gradient of the policy is given by

$$\begin{aligned}
\frac{\partial J(u)}{\partial u} &= \mathbb{E}_s\left[\frac{\partial Q^{\pi}(s,a)}{\partial u}\right] \\
&= \mathbb{E}_s\left[\frac{\partial Q^{\pi}(s,a)}{\partial a}\frac{\partial \pi(s,u)}{\partial u}\right]
\end{aligned}$$

Policy gradient is the direction that most improves Q

Deterministic Actor-Critic

Use two networks

- Actor is a policy $\pi(s, u)$ with parameters u

$$s \xrightarrow[u_1]{} \dots \xrightarrow[u_n]{} a$$

- Critic is value function $Q(s, a, w)$ with parameters w

$$s, a \xrightarrow[w_1]{} \dots \xrightarrow[w_n]{} Q$$

- Critic provides loss function for actor

$$s \xrightarrow[u_1]{} \dots \xrightarrow[u_n]{} a \xrightarrow[w_1]{} \dots \xrightarrow[w_n]{} Q$$

- Gradient backpropagates from critic into actor

$$\frac{\partial a}{\partial u} \longleftarrow \dots \longleftarrow \frac{\partial Q}{\partial a} \longleftarrow \dots \longleftarrow$$

Deterministic Actor-Critic: Learning Rule

- Critic estimates value of current policy by Q-learning

$$\frac{\partial \mathcal{L}(w)}{\partial w} = \mathbb{E}\left[\left(r + \gamma Q(s', \pi(s'), w) - Q(s, a, w)\right)\frac{\partial Q(s, a, w)}{\partial w}\right]$$

- Actor updates policy in direction that improves Q

$$\frac{\partial J(u)}{\partial u} = \mathbb{E}_s\left[\frac{\partial Q(s, a, w)}{\partial a}\frac{\partial \pi(s, u)}{\partial u}\right]$$

Deterministic Deep Policy Gradient (DDPG)

- Naive actor-critic oscillates or diverges with neural nets
- DDPG provides a stable solution

Deterministic Deep Policy Gradient (DDPG)

- Naive actor-critic oscillates or diverges with neural nets
- DDPG provides a stable solution

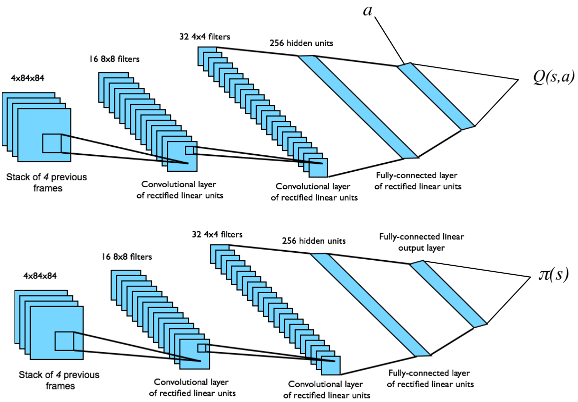
1. Use experience replay for both actor and critic
2. Freeze target network to avoid oscillations

$$\frac{\partial \mathcal{L}(w)}{\partial w} = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}} \left[\left(r + \gamma Q(s', \pi(s', u^-), w^-) - Q(s,a,w) \right) \frac{\partial Q(s,a,w)}{\partial w} \right]$$

$$\frac{\partial J(u)}{\partial u} = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}} \left[\frac{\partial Q(s,a,w)}{\partial a} \frac{\partial \pi(s,u)}{\partial u} \right]$$

DDPG for Continuous Control

- End-to-end learning of control policy from raw pixels s
- Input state s is stack of raw pixels from last 4 frames
- Two separate convnets are used for Q and π
- Physics are simulated in MuJoCo



[Lillicrap et al.]

DDPG Demo

Outline

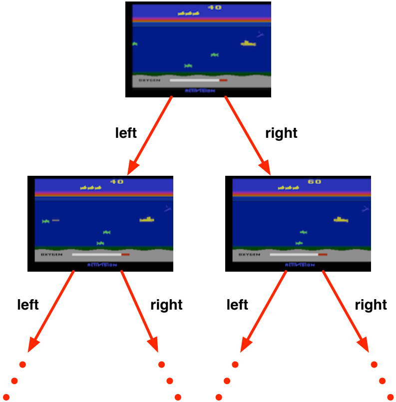
Model-Based RL

Learn a transition model of the environment

$$p(r, s' \mid s, a)$$

Plan using the transition model

- e.g. Lookahead using transition model to find optimal actions



Learning Models of the Environment

- Challenging to plan due to compounding errors
  - Errors in the transition model compound over the trajectory
  - Planning trajectories differ from executed trajectories
  - At end of long, unusual trajectory, rewards are totally wrong

Deep Reinforcement Learning in Go

What if we have a perfect model? e.g. game rules are known

AlphaGo paper:
www.nature.com/articles/nature16961

AlphaGo resources:
deepmind.com/alphago/

Conclusion

- General, stable and scalable RL is now possible
- Using deep networks to represent value, policy, model
- Successful in several tasks
- Using a variety of deep RL paradigms