Advanced Machine Learning
Lecture 20

Deep Reinforcement Learning II

02.02.2017

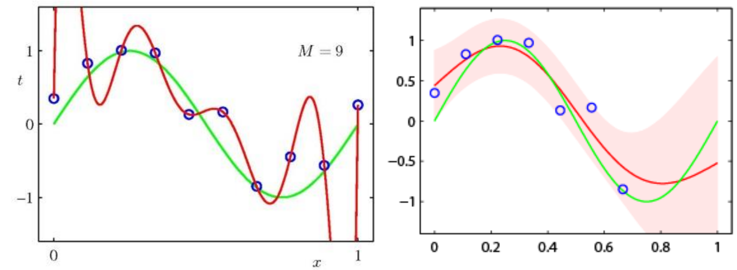
Bastian Leibe
RWTH Aachen
http://www.vision.rwth-aachen.de/

leibe@vision.rwth-aachen.de

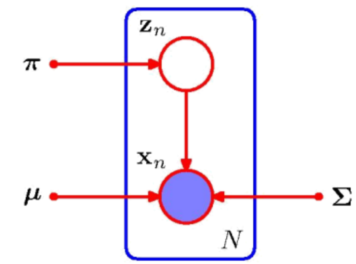
This Lecture: *Advanced Machine Learning*

- Regression Approaches
  - Linear Regression
  - Regularization (Ridge, Lasso)
  - Kernels (Kernel Ridge Regression)
  - Gaussian Processes

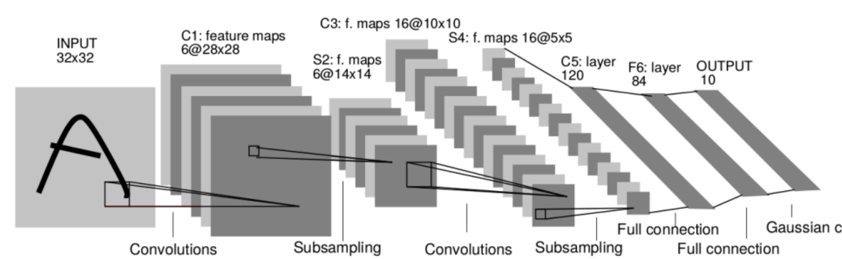
$$f : \mathcal{X} \to \mathbb{R}$$



- Approximate Inference
  - Sampling Approaches
  - MCMC



- Deep Learning
  - Linear Discriminants
  - Neural Networks
  - Backpropagation & Optimization
  - CNNs, ResNets, RNNs, Deep RL, etc.

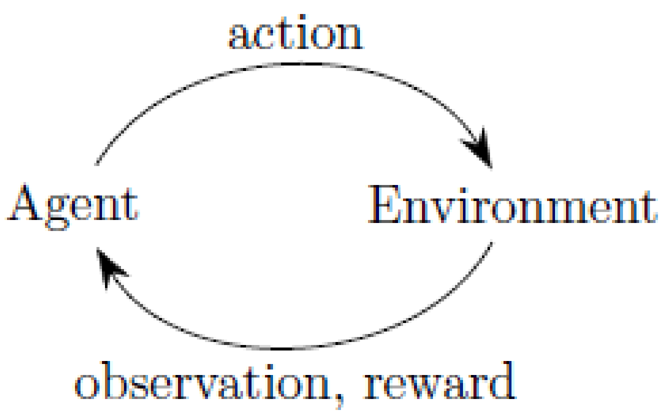


Topics of This Lecture

- Recap: Reinforcement Learning
  - Key Concepts
  - Temporal Difference Learning
- Deep Reinforcement Learning
  - Value based Deep RL
  - Policy based Deep RL
  - Model based Deep RL
- Applications

Recap: Reinforcement Learning

- **Motivation**
  - General purpose framework for decision making.
  - Basis: Agent with the capability to interact with its environment
  - Each action influences the agent's future state.
  - Success is measured by a scalar reward signal.
  - Goal: select actions to maximize future rewards.



  - Formalized as a partially observable Markov decision process (POMDP)

Slide adapted from: David Silver, Sergey Levine

Recap: Reward vs. Return

- **Objective of learning**
  - We seek to maximize the **expected return** G_t as some function of the reward sequence $R_{t+1}, R_{t+2}, R_{t+3}, \ldots$
  - Standard choice: **expected discounted return**

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

  where $0 \leq \gamma \leq 1$ is called the **discount rate**.

- **Difficulty**
  - We don't know which past actions caused the reward.

  $\Rightarrow$ Temporal credit assignment problem

Recap: Policy

- **Definition**
  - A policy determines the agent's behavior
  - Map from state to action $\pi: \mathcal{S} \rightarrow \mathcal{A}$

- **Two types of policies**
  - Deterministic policy: $a = \pi(s)$
  - Stochastic policy: $\pi(a|s) = \Pr\{A_t = a | S_t = s\}$

- **Note**
  - $\pi(a|s)$ denotes the probability of taking action a when in state s.

Recap: Value Function

- **Idea**
  - Value function is a prediction of future reward
  - Used to evaluate the goodness/badness of states
  - And thus to select between actions

- **Definition**
  - The value of a state s under a policy π, denoted $v_\pi(s)$, is the expected return when starting in s and following π thereafter.

$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \, | S_t = s]$$

  - The value of taking action a in state s under a policy π, denoted $q_\pi(s,a)$, is the expected return starting from s, taking action a, and following π thereafter.

$$q_\pi(s,a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \, | S_t = s, A_t = a]$$

Recap: Optimal Value Functions

- **Bellman optimality equations**
  - For the optimal state-value function v_*:

$$v_*(s) = \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s,a)$$

$$= \max_{a \in \mathcal{A}(s)} \sum_{s',r} p(s',r|s,a)[r + \gamma v_*(s')]$$

  - v_* is the unique solution to this system of nonlinear equations.
  - For the optimal action-value function q_*:

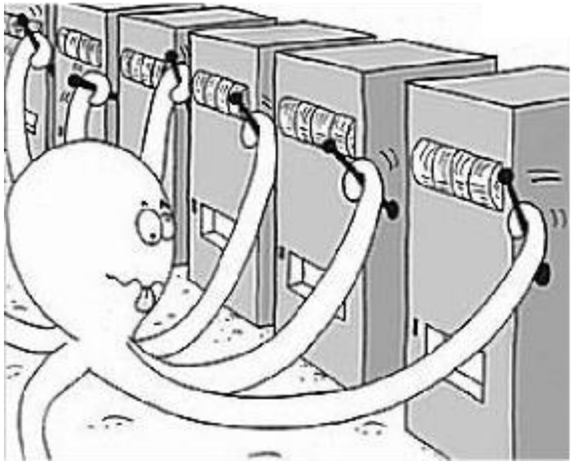
$$q_*(s,a) = \sum_{s',r} p(s',r|s,a) \left[r + \gamma \max_{a'} q_*(s',a')\right]$$

  - q_* is the unique solution to this system of nonlinear equations.

$\Rightarrow$ If the dynamics of the environment $p(s',r|s,a)$ are known, then in principle one can solve those equation systems.

Recap: Exploration-Exploitation Trade-off

- Example: N-armed bandit problem
  - Suppose we have the choice between N actions $a_1, \ldots, a_N$.
  - If we knew their value functions $q_*(s, a_i)$, it would be trivial to choose the best.
  - However, we only have estimates based on our previous actions and their returns.



- We can now
  - Exploit our current knowledge
    - And choose the greedy action that has the highest value based on our current estimate.
  - Explore to gain additional knowledge
    - And choose a non-greedy action to improve our estimate of that action's value.

Image source: research.microsoft.com

Recap: TD-Learning

- **Policy evaluation (the prediction problem)**
  - For a given policy π, compute the state-value function v_π.
- **One option: Monte-Carlo methods**
  - Play through a sequence of actions until a reward is reached, then backpropagate it to the states on the path.

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$$

Target: the actual return after time t

- **Temporal Difference Learning - TD(λ)**
  - Directly perform an update using the estimate $V(S_{t+\lambda+1})$.

$$V(S_t) \leftarrow V(S_t) + \alpha[\underbrace{R_{t+1} + \gamma V(S_{t+1})} - V(S_t)]$$

Target: an estimate of the return (here: TD(0))

Recap: SARSA – On-Policy TD Control

- **Idea**
  - Turn the TD idea into a control method by always updating the policy to be greedy w.r.t. the current estimate

- **Procedure**
  - Estimate $q_\pi(s,a)$ for the current policy π and for all states s and actions a.
  - TD(0) update equation

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1},A_{t+1}) - Q(S_t,A_t)]$$

  - This rule is applied after every transition from a nonterminal state S_t.
  - It uses every element of the quintuple $(S_t,A_t,R_{t+1},S_{t+1},A_{t+1})$.

$\Rightarrow$ *Hence, the name SARSA.*

Recap: Q-Learning – Off-Policy TD Control

- **Idea**
  - Directly approximate the optimal action-value function q_*, independent of the policy being followed.

- **Procedure**
  - TD(0) update equation

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right]$$

  - Dramatically simplifies the analysis of the algorithm.
  - All that is required for correct convergence is that all pairs continue to be updated.

Image source: Sutton & Barto

Approaches Towards RL

- **Value-based RL**
  - Estimate the optimal value function $q_*(s, a)$
  - This is the maximum value achievable under any policy
- **Policy-based RL**
  - Search directly for the optimal policy π_*
  - This is the policy achieving maximum future reward
- **Model-based RL**
  - Build a model of the environment
  - Plan (e.g. by lookahead) using model

Topics of This Lecture

- Recap: Reinforcement Learning
  - Key Concepts
  - Temporal Difference Learning
- Deep Reinforcement Learning
  - Value based Deep RL
  - Policy based Deep RL
  - Model based Deep RL
- Applications

Deep Reinforcement Learning

- RL using deep neural networks to approximate functions
  - Value functions
    - Measure goodness of states or state-action pairs
  - Policies
    - Select next action
  - Dynamics Models
    - Predict next states and rewards

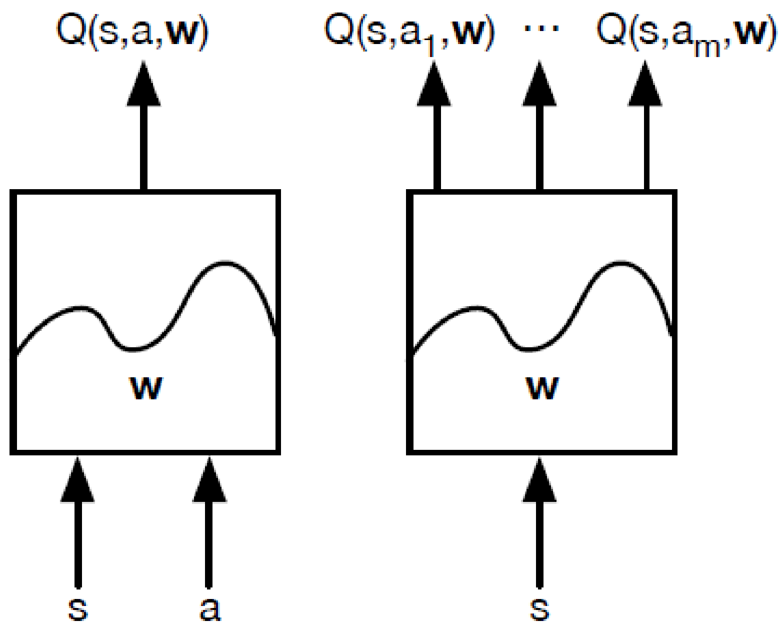
Deep Reinforcement Learning

- Use deep neural networks to represent
  - Value function
  - Policy
  - Model
- Optimize loss function by stochastic gradient descent

Q-Networks

- Represent value function by **Q-Network** with weights $\mathbf{w}$

$$Q(s, a, \mathbf{w}) = Q_*(s, a)$$



Slide credit: David Silver

Deep Q-Learning

- **Idea**
  - **Optimal Q-values should obey Bellman equation**

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

  - **Treat the right-hand side $r + \gamma \max_{a'} Q(s',a',\mathbf{w})$ as a target**
  - **Minimize MSE loss by stochastic gradient descent**

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

  - **This converges to Q_* using a lookup table representation.**
  - **Unfortunately, it diverges using neural networks due to**
    - Correlations between samples
    - Non-stationary targets

Deep Q-Networks (DQN): Experience Replay

- **Adaptations**
  - To remove correlations, build a dataset from agent's own experience

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

$s_t, a_t, r_{t+1}, s_{t+1} \rightarrow$ (table) $\rightarrow s, a, r, s'$

  - Perform minibatch updates to samples of experience drawn at random from the pool of stored samples
    - $(s, a, r, s') \sim U(D)$ where $D = \{(s_t, a_t, r_{t+1}, s_{t+1})\}$ is the dataset
  - Advantages
    - Each experience sample is used in many updates (more efficient)
    - Avoids correlation effects when learning from consecutive samples
    - Avoids feeback loops from on-policy learning

Deep Q-Networks (DQN): Experience Replay

- **Adaptations**
  - **To remove correlations, build a dataset from agent's own experience**

$$
\begin{array}{ccc}
 & \begin{array}{|c|}\hline s_1, a_1, r_2, s_2 \\ \hline s_2, a_2, r_3, s_3 \\ \hline s_3, a_3, r_4, s_4 \\ \hline \ldots \\ \hline s_t, a_t, r_{t+1}, s_{t+1} \\ \hline \end{array} & \begin{array}{c} \\ \rightarrow \quad {\color{red} s, a, r, s'} \\ \\ \\ \\ \end{array} \\
\begin{array}{c} \\ \\ \\ \\ {\color{red} s_t, a_t, r_{t+1}, s_{t+1}} \quad \rightarrow \end{array} & &
\end{array}
$$

  - **Sample from the dataset and apply an update**

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

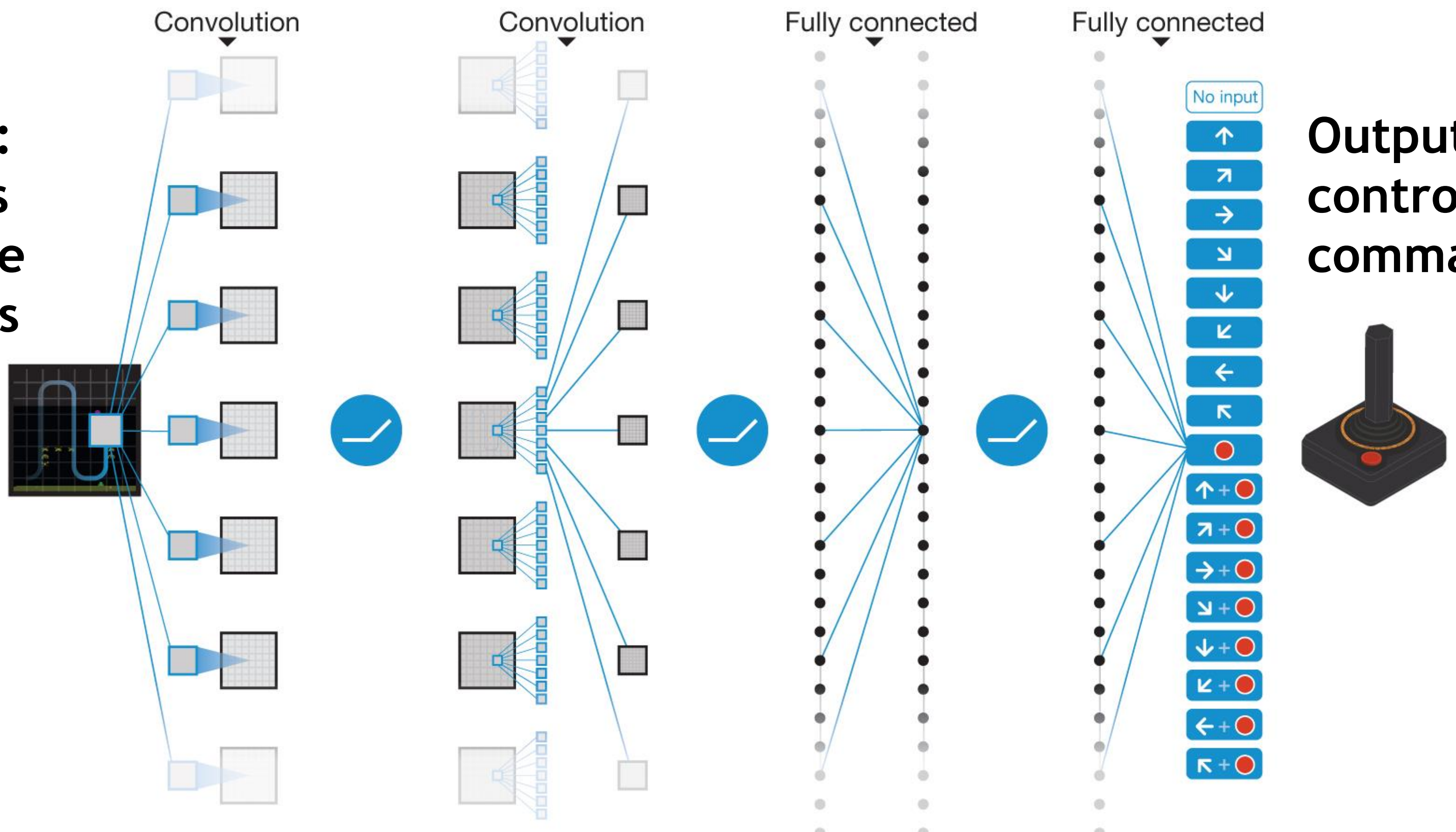
  - **To deal with non-stationary parameters $\mathbf{w}^-$, are held fixed.**
    - **Only update the target network parameters every C steps.**
    - **I.e., clone the network Q to generate a target network $\hat{Q}$.**

    $\Rightarrow$ **Again, this reduces oscillations to make learning more stable.**

Application: Deep RL in Atari

- Goal: Learning to play Atari games

Input: pixels +game scores

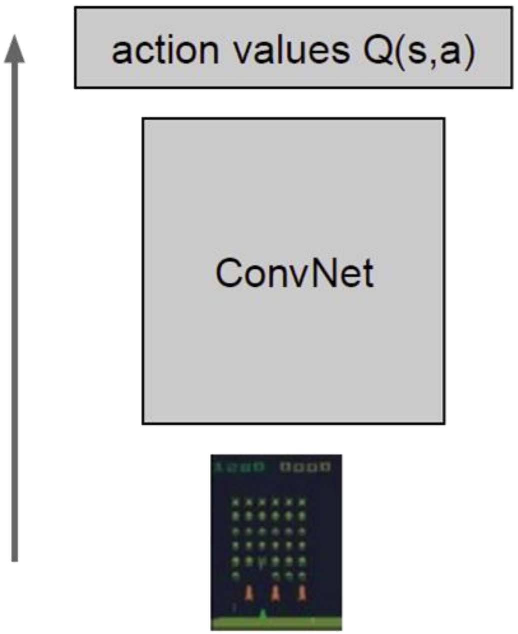


Output: control commands

V. Mnih et al., [Human-level control through deep reinforcement learning](), Nature Vol. 518, pp. 529-533, 2015

Image source: Vlodimir Minh et al.

Idea Behind the Model

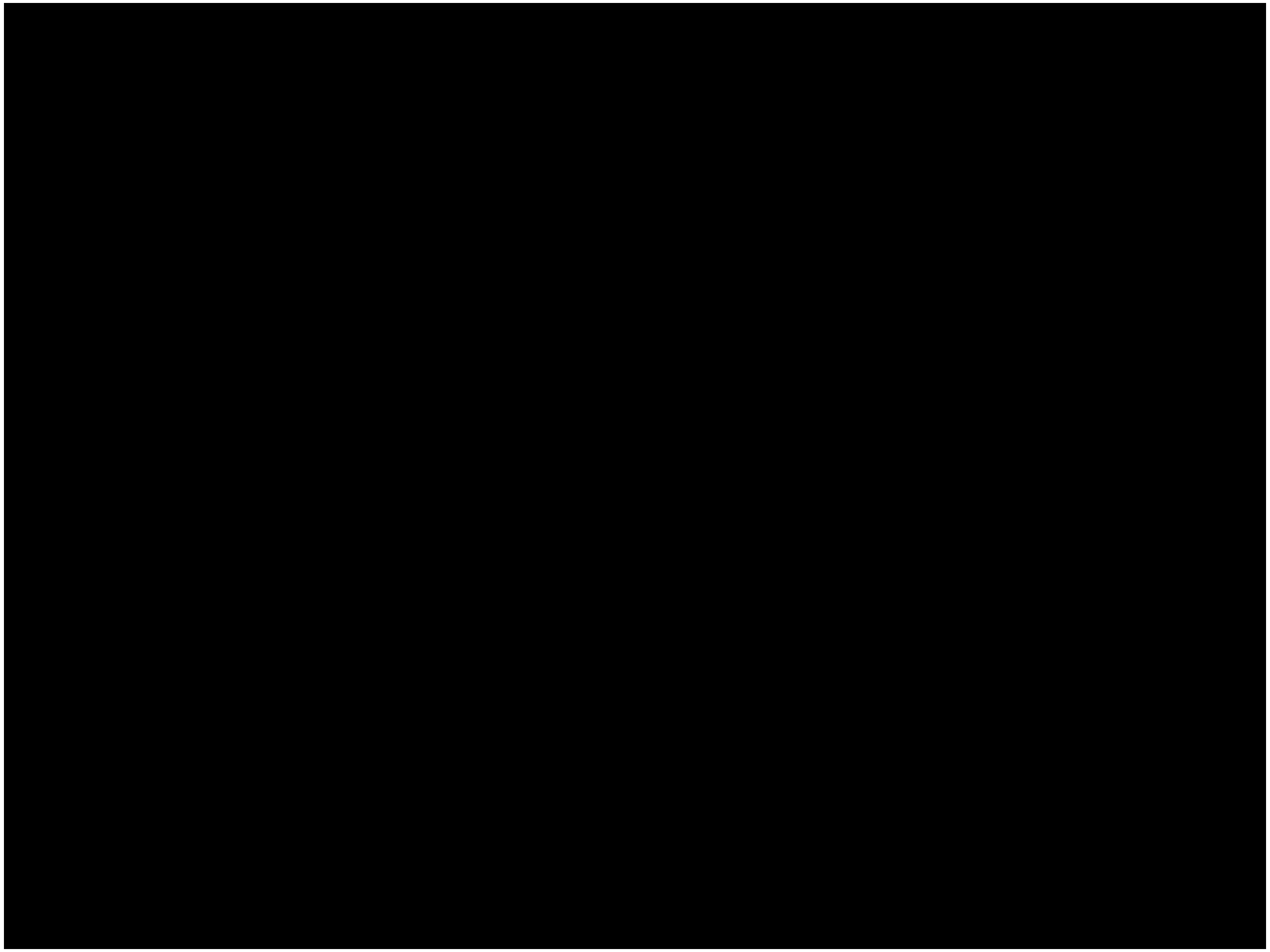


- Interpretation
  - Assume finite number of actions
  - Each number here is a real-valued quantity that represents the Q function in Reinforcement Learning
- Collect experience dataset:
  - Set of tuples {(s,a,s',r), … }
  - (State, Action taken, New state, Reward received)
- L2 Regression Loss

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim \mathrm{U}(D)}\left[\left(\overset{\text{target value}}{\boxed{r + \gamma \max_{a'} Q(s',a';\theta_i^-)}} - \overset{\text{predicted value}}{\boxed{Q(s,a;\theta_i)}}\right)^2\right]$$

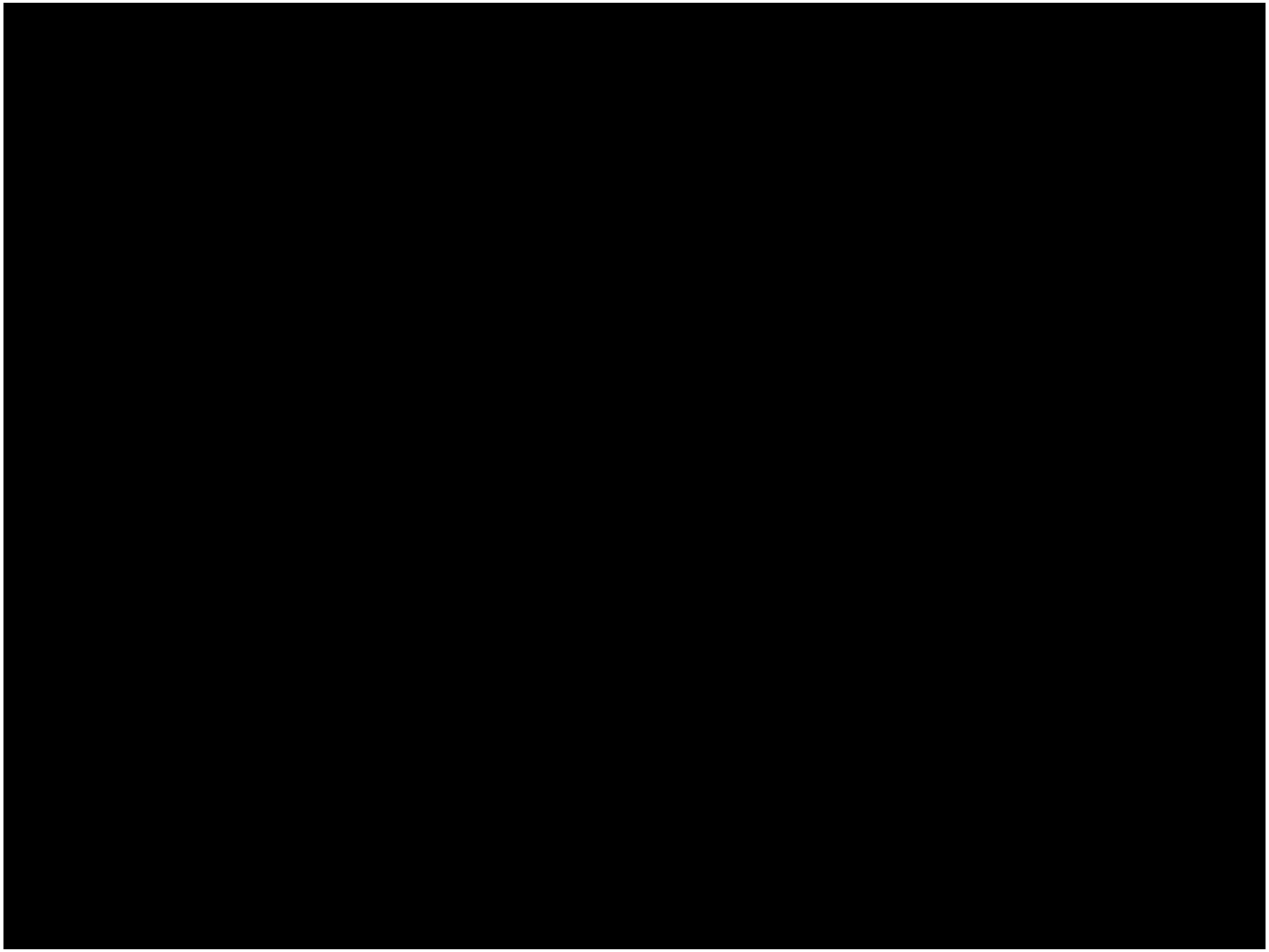
Current reward + estimate of future reward, discounted by γ

Results: Breakout



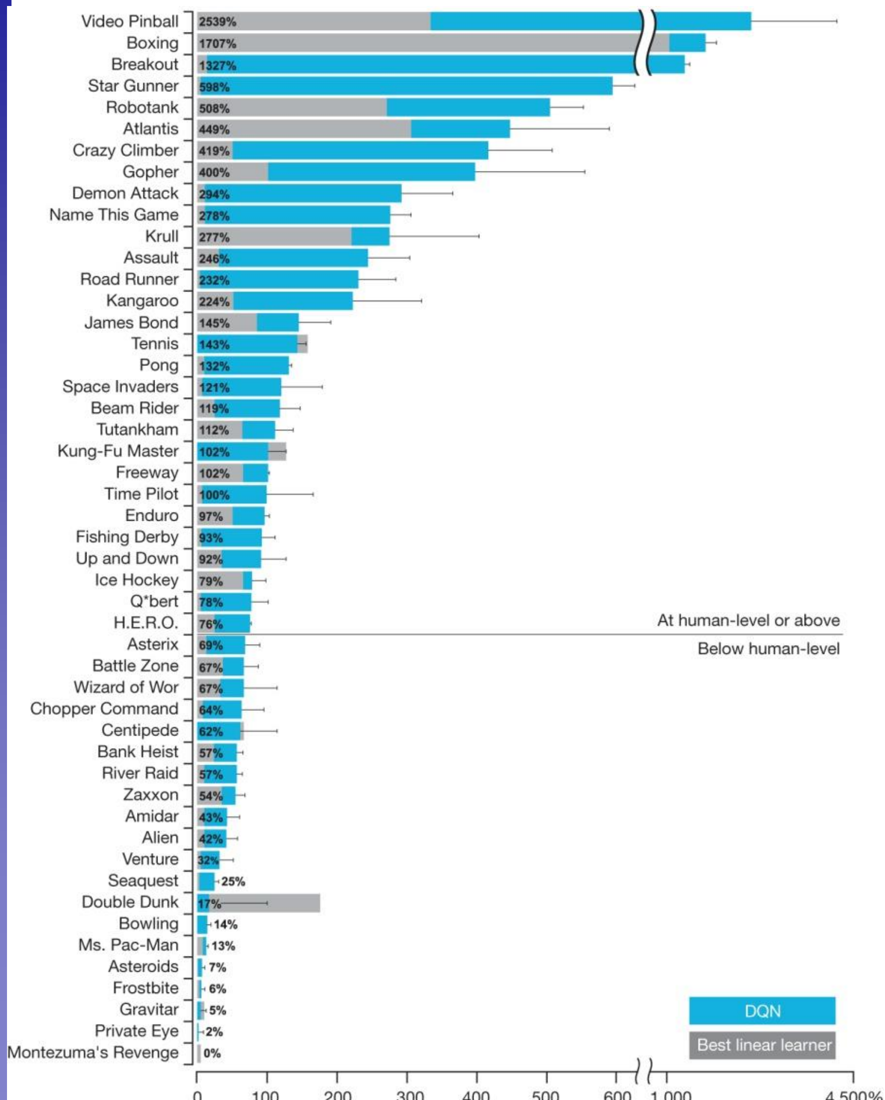
Video source: Vlodimir Minh et al.

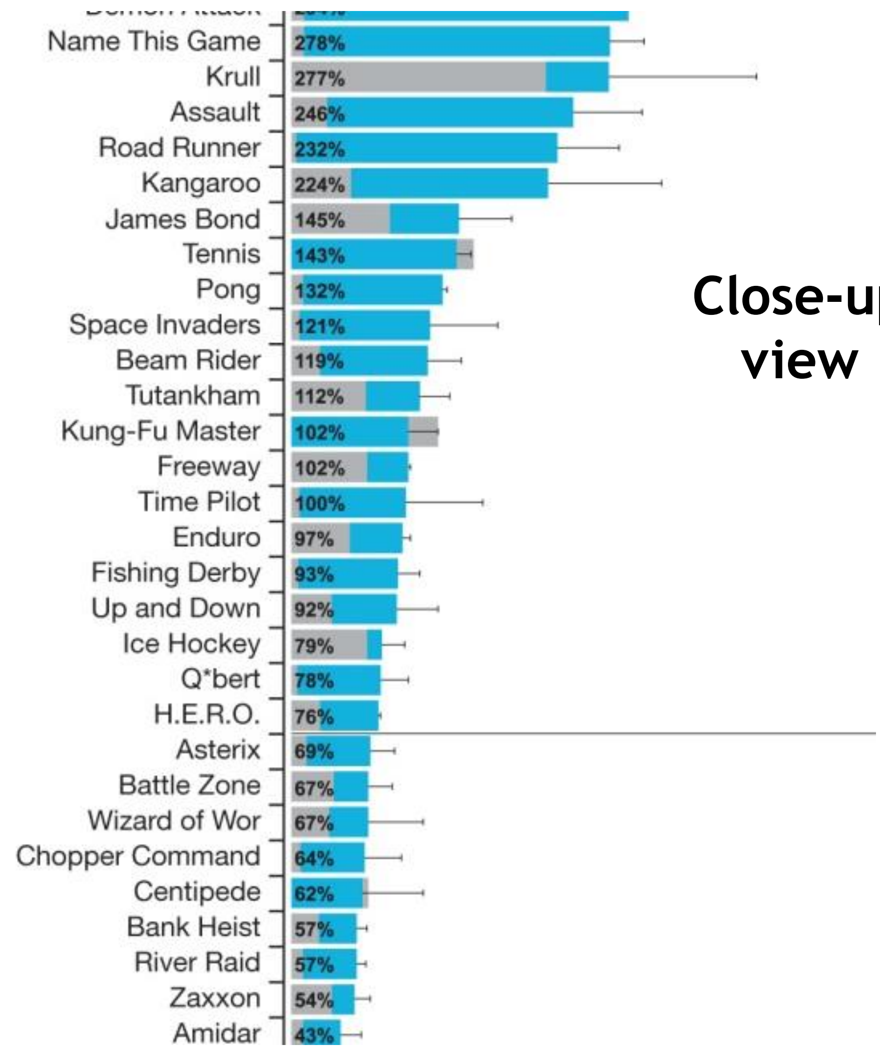
Results: Space Invaders



Video source: Vlodimir Minh et al.

Comparison with Human Performance

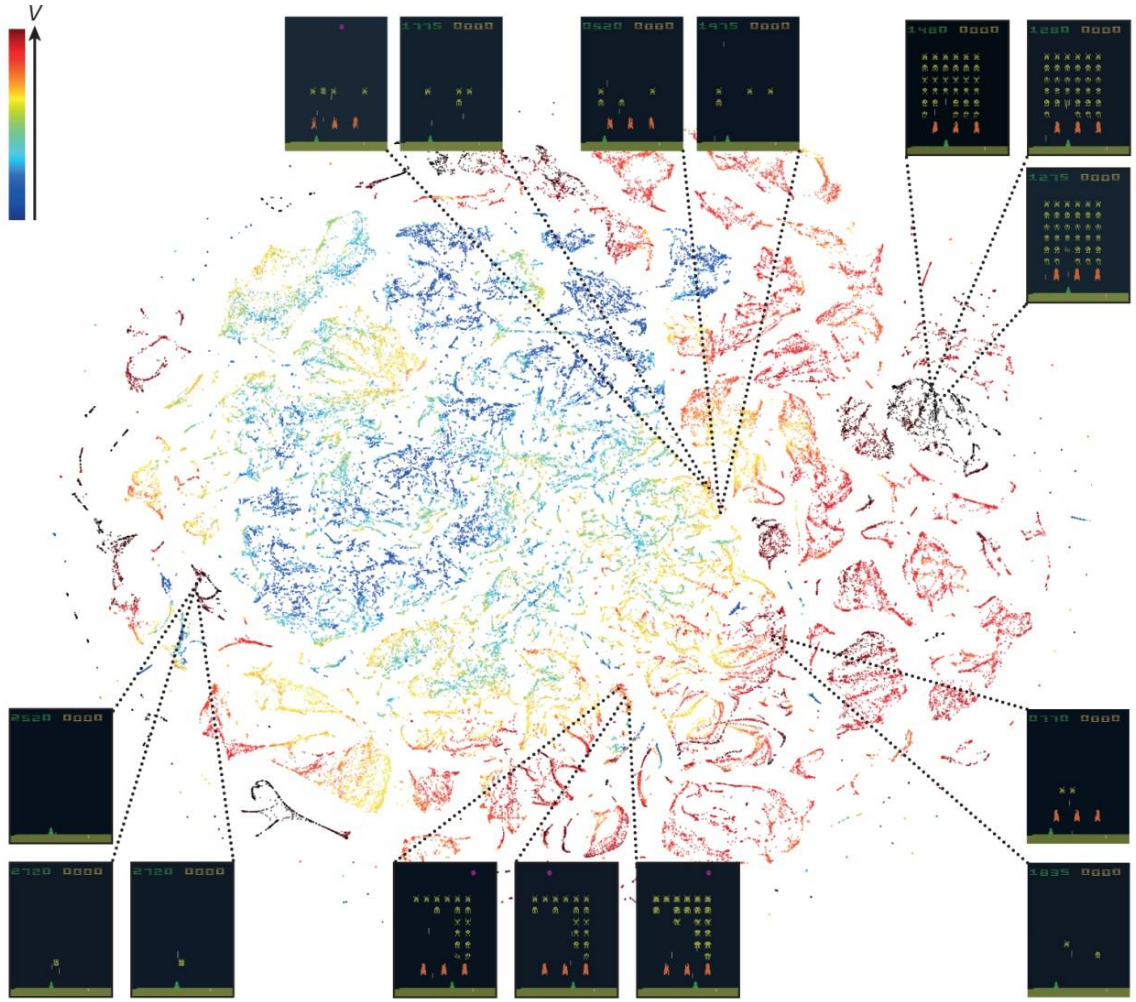




Close-up view

Image source: Vlodimir Minh et al.

Learned Representation



- **t-SNE embedding of DQN last hidden layer (Space Inv.)**

Image source: Vlodimir Minh et al.

Improvements since Nature DQN

- **Double DQN**
  - Remove upward bias caused by $\max_a Q(s,a,\mathbf{w})$
  - Current Q-network w is used to select actions
  - Older Q-network w^- is used to evaluate actions

$$L(\mathbf{w}) = \left(r + \gamma Q\left(s', \operatorname*{argmax}_a Q(s',a',\mathbf{w}), \mathbf{w}^-\right) - Q(s,a,\mathbf{w})\right)^2$$

- **Prioritised replay**
  - Weight experience according to surprise
  - Store experience in priority queue according to DQN error

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

$\Rightarrow$ Emphasize state transitions from which one can learn the most.

Improvements since Nature DQN (2)

- **Duelling network**
  - Split Q-network into two channels
  - Action-independent value function $V(s, v)$
  - Action-dependent advantage function $A(s, a, \mathbf{w})$

$$Q(s, a) = V(s, v) + A(s, a, \mathbf{w})$$

  - Intuition: network can learn which states are valuable without having to learn the effect of each action for each state.

- **Combined Algorithm**
  - $3\times$ mean Atari score vs. Nature DQN

Topics of This Lecture

- Recap: Reinforcement Learning
  - Key Concepts
  - Temporal Difference Learning
- Deep Reinforcement Learning
  - Value based Deep RL
  - Policy based Deep RL
  - Model based Deep RL
- Applications

Deep Policy Networks

- **Idea**
  - **Represent policy by deep network with weights** u

$$a = \pi(a|s,\mathbf{u}) \ \ \textbf{or} \ \ a = \pi(s,\mathbf{u})$$

  - **Define objective function as total discounted reward**

$$L(\mathbf{u}) = \mathbb{E}[r_1 + \gamma r_2 + \gamma^2 r_3 + \ \dots | \, \pi(\cdot,\mathbf{u})]$$

  - **Optimize effective end-to-end by SGD**
  - **I.e., adjust policy parameters** $\mathbf{u}$ **to achieve more reward**

Slide credit: David Silver

Policy Gradients

- **How to make high-value actions more likely**
  - **The gradient of the stochastic policy $\pi(s, \mathbf{u})$ is given by**

$$\frac{\partial L(\mathbf{u})}{\partial \mathbf{u}} = \frac{\partial}{\partial \mathbf{u}} \mathbb{E}[r_1 + \gamma r_2 + \gamma^2 r_3 + \ \dots | \, \pi(\cdot, \mathbf{u})]$$

$$= \ \dots ?$$

- **Wait - how do we calculate that?**
  - **Any ideas?**

Policy Gradients

- **Deriving the gradient of an expectation**
  - **General case**

$$
\begin{aligned}
\nabla_\theta \mathbb{E}_{p(x;\theta)}[f(x)] &= \nabla_\theta \sum_x p(x;\theta) f(x) \\
&= \sum_x \nabla_\theta p(x;\theta) f(x) \\
&= \sum_x p(x;\theta) \frac{\nabla_\theta p(x;\theta)}{p(x;\theta)} f(x) \\
&= \sum_x p(x;\theta) \nabla_\theta \log p(x;\theta)\, f(x) \\
&= \mathbb{E}_{p(x;\theta)}[\nabla_\theta \log p(x;\theta)\, f(x)]
\end{aligned}
$$

Policy Gradients

- **How to make high-value actions more likely**
  - **The gradient of a stochastic policy $\pi(s, \mathbf{u})$ is given by**

$$\frac{\partial L(\mathbf{u})}{\partial \mathbf{u}} = \frac{\partial}{\partial \mathbf{u}} \mathbb{E}_\pi[r_1 + \gamma r_2 + \gamma^2 r_3 + \ \dots \mid \pi(\cdot, \mathbf{u})]$$

$$= \mathbb{E}_\pi \left[\frac{\partial \log \pi(a|s, \boldsymbol{u})}{\partial \boldsymbol{u}} Q_\pi(s, a) \right]$$

  - **The gradient of a deterministic policy $a = \pi(s)$ is given by**

$$\frac{\partial L(\mathbf{u})}{\partial \mathbf{u}} = \mathbb{E}_\pi \left[\frac{\partial Q_\pi(s, a)}{\partial a} \frac{\partial a}{\partial \mathbf{u}} \right]$$

if a is continuous and Q is differentiable.

Actor-Critic Algorithm

- **Procedure**
  - Estimate value function $Q(s, a, \mathbf{w}) \approx Q_\pi(s, a)$
  - Update policy parameters u by stochastic gradient ascent

$$\frac{\partial L(\mathbf{u})}{\partial \mathbf{u}} = \frac{\partial \log \pi(a|s, \boldsymbol{u})}{\partial \boldsymbol{u}} Q(s, a, \mathbf{w})$$

stochastic policy

  - or

$$\frac{\partial L(\mathbf{u})}{\partial \mathbf{u}} = \frac{\partial Q(s, a, \mathbf{w})}{\partial a} \frac{\partial a}{\partial \mathbf{u}}$$

deterministic policy

Asynchronous Advantage Actor-Critic (A3C)

- **Further improvement**
  - **Estimate state-value function**

$$V(s) \approx \mathbb{E}[r_{t+1} + \gamma r_{t+2} + \ \dots \mid s]$$

  - **Q-value estimated by an n-step sample**

$$q_t = r_{t+1} + \gamma r_{t+2} + \ \dots + \gamma^{n-1} r_{t+n} + \gamma^n V(s_{t+n}, \mathbf{v})$$

  - **Actor is updated towards target**

$$\frac{\partial L(\mathbf{u})}{\partial \mathbf{u}} = \frac{\partial \log \pi(a_t | s_t, \boldsymbol{u})}{\partial \boldsymbol{u}} (q_t - V(s_t, \mathbf{v}))$$

  - **Critic is updated to minimize MSE w.r.t. target**

$$L_{\mathbf{v}} = \left(q_t - V(s_t, \mathbf{v})\right)^2$$

$\Rightarrow$ ***Combined effect: 4× mean Atari score vs. Nature DQN***

Deep Policy Gradients (DPG)

- DPG is the continuous analogue of DQN
  - Experience replay: build data-set from agent's experience
  - Critic estimates value of current policy by DQN

$$L_{\mathbf{w}}(\mathbf{w}) = \left(r + \gamma Q(s', \pi(s', \mathbf{u}^{-}), \mathbf{w}^{-}) - Q(s, a, \mathbf{w})\right)^2$$

  - To deal with non-stationarity, targets $\mathbf{u}^{-}, \mathbf{w}^{-}$ are held fixed
  - Actor updates policy in direction that improves Q

$$\frac{\partial L_{\mathbf{u}}(\mathbf{u})}{\partial \mathbf{u}} = \frac{\partial Q(s, a, \mathbf{w})}{\partial a} \frac{\partial a}{\partial \mathbf{u}}$$

  - In other words critic provides loss function for actor.

Summary

- The future looks bright!
  - Soon, you won't have to play video games anymore…
  - Your computer can do it for you (and beat you at it)

- Reinforcement Learning is a very promising field
  - Currently limited by the need for data
  - At the moment, mainly restricted to simulation settings

Topics of This Lecture

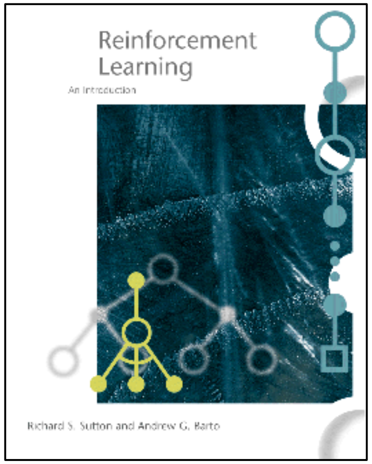
- Recap: Reinforcement Learning
  - Key Concepts
  - Temporal Difference Learning
- Deep Reinforcement Learning
  - Value based Deep RL
  - Policy based Deep RL
  - Model based Deep RL
- Applications

Often Used in Games, E.g. Alpha Go

References and Further Reading

- More information on Reinforcement Learning can be found in the following book

Richard S. Sutton, Andrew G. Barto
Reinforcement Learning: An Introduction
MIT Press, 1998



- The complete text is also freely available online

https://webdocs.cs.ualberta.ca/~sutton/book/ebook/the-book.html

References and Further Reading

- **DQN paper**
  - www.nature.com/articles/nature14236

- **AlphaGo paper**
  - www.nature.com/articles/nature16961