

# Lecture 10: Deep Q Learning

Bolei Zhou

The Chinese University of Hong Kong

*bzhou@ie.cuhk.edu.hk*

March 28, 2020

# Today's Plan

- 1 Review on value function approximation
- 2 Deep Q Networks (DQNs)
- 3 Code example and the extension of DQNs
- 4 Project proposal review session
- 5 Homework 1 review session

# Scaling up RL

- ➊ Generalization of RL to tackle practical problems such as self-driving cars, Atari, consumer marketing, healthcare, education
- ➋ Most of these practical problems have enormous state and/or action spaces
- ➌ It requires the representations of models/values/policies that can generalize across states and/or actions
- ➍ Solution: to represent a value function with a parameterized function instead of a lookup table

$$\hat{v}(s, \mathbf{w}) \approx v_{\pi}(s)$$

$$\hat{q}(s, a, \mathbf{w}) \approx q_{\pi}(s, a)$$

# Review on Stochastic Gradient Descend for Function Approximation

- ① Goal: Find the parameter vector  $\mathbf{w}$  that minimizes the loss between a true value function  $v_\pi(s)$  and its approximation  $\hat{v}_\pi(s, \mathbf{w})$  as represented with a particular function approximator parameterized by  $\mathbf{w}$
- ② The mean square error loss function is as

$$J(\mathbf{w}) = \mathbb{E}_\pi \left[ (v_\pi(S) - \hat{v}(s, \mathbf{w}))^2 \right]$$

- ③ Follow the gradient descend to find a local minimum

$$\begin{aligned}\Delta \mathbf{w} &= -\frac{1}{2} \alpha \nabla_{\mathbf{w}} J(\mathbf{w}) \\ \mathbf{w}_{t+1} &= \mathbf{w}_t + \Delta \mathbf{w}\end{aligned}$$

# Linear Function Approximation

- 1 Represent value function by a linear combination of features

$$\hat{v}(S, \mathbf{w}) = \mathbf{x}(S)^T \mathbf{w} = \sum_{j=1}^n x_j(S) w_j$$

- 2 Objective function is  $J(\mathbf{w}) = \mathbb{E}_{\pi} \left[ (v_{\pi}(S) - \mathbf{x}(S)^T \mathbf{w})^2 \right]$
- 3 Update is as simple as  $\Delta \mathbf{w} = \alpha (v_{\pi}(S) - \hat{v}(S, \mathbf{w})) \mathbf{x}(S)$
- 4 But there is no oracle for the true value  $v_{\pi}(S)$ , we substitute with the target from MC or TD
  - 1 for MC policy evaluation,

$$\Delta \mathbf{w} = \alpha (G_t - \hat{v}(S_t, \mathbf{w})) \mathbf{x}(S_t)$$

- 2 for TD policy evaluation,

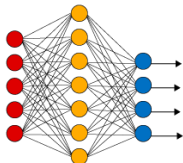
$$\Delta \mathbf{w} = \alpha (R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w})) \mathbf{x}(S_t)$$

# Linear vs Nonlinear Value Function Approximation

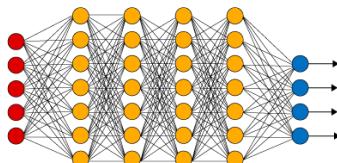
- ① Linear VFA often works well given the right set of features
- ② But it requires manual designing of the feature set
- ③ Alternative is to use a much richer function approximator that is able to directly learn from states without requiring the feature design
- ④ Nonlinear function approximator: Deep neural networks

# Deep Neural Networks

Simple Neural Network



Deep Learning Neural Network



● Input Layer

● Hidden Layer

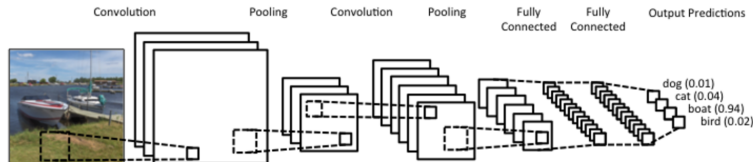
● Output Layer

- 1 Multiple layers of linear functions, with non-linear operators between layers

$$f(\mathbf{x}; \theta) = \mathbf{W}_{L+1}^T \sigma(\mathbf{W}_L^T \sigma(\dots \sigma(\mathbf{W}_1^T \mathbf{x} + \mathbf{b}_1) + \dots + \mathbf{b}_{L-1}) + \mathbf{b}_L) + \mathbf{b}_{L+1}$$

- 2 The chain rule to backpropagate the gradient to update the weights using the loss function  $L(\theta) = \frac{1}{n} \sum_{i=1}^n \left( y^{(i)} - f(\mathbf{x}; \theta) \right)^2$

# Convolutional Neural Networks



- 1 Convolution encodes the local information in 2D feature map
- 2 Layers of convolution, reLU, batch normalization, etc.
- 3 CNNs are widely used in computer vision (more than 70% top conference papers using CNNs)
- 4 A detailed introduction on CNNs:  
<http://cs231n.github.io/convolutional-networks/>

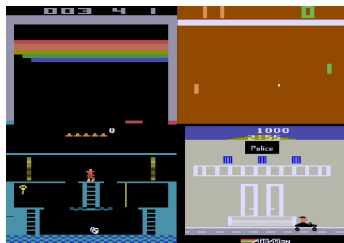


# Deep Reinforcement Learning

- ① Frontier in machine learning and artificial intelligence
- ② Deep neural networks are used to represent
  - ① Value function
  - ② Policy function (policy gradient methods to be introduced)
  - ③ World model
- ③ Loss function is optimized by stochastic gradient descent (SGD)
- ④ Challenges
  - ① Efficiency: too many model parameters to optimize
  - ② The Deadly Triad for the danger of instability and divergence in training
    - ① Nonlinear function approximation
    - ② Bootstrapping
    - ③ Off-policy training

# Deep Q-Networks (DQN)

- 1 DeepMind's **Nature** paper: Mnih, Volodymyr; et al. (2015).  
**Human-level control through deep reinforcement learning**
- 2 DQN represents the action value function with neural network approximator
- 3 DQN reaches a professional human gaming level across many Atari games using the same network and hyperparameters



4 Atari Games: Breakout, Pong, Montezuma's Revenge, Private Eye

# Recall: Action-Value Function Approximation

- 1 Approximate the action-value function

$$\hat{q}(S, A, \mathbf{w}) \approx q_\pi(S, A)$$

- 2 Minimize the MSE (mean-square error) between approximate action-value and true action-value (assume oracle)

$$J(\mathbf{w}) = \mathbb{E}_\pi[(q_\pi(S, A) - \hat{q}(S, A, \mathbf{w}))^2]$$

- 3 Stochastic gradient descend to find a local minimum

$$\Delta \mathbf{w} = \alpha(q_\pi(S, A) - \hat{q}(S, A, \mathbf{w})) \nabla_{\mathbf{w}} \hat{q}(S, A, \mathbf{w})$$

# Recall: Incremental Control Algorithm

Same to the prediction, there is no oracle for the true value  $q_\pi(S, A)$ , so we substitute a target

- ① For MC, the target is the return  $G_t$

$$\Delta \mathbf{w} = \alpha (\textcolor{red}{G}_t - \hat{q}(S_t, A_t, \mathbf{w})) \nabla_{\mathbf{w}} \hat{q}(S_t, A_t, \mathbf{w})$$

- ② For Sarsa, the target is the TD target  $R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, \mathbf{w})$

$$\Delta \mathbf{w} = \alpha (\textcolor{red}{R}_{t+1} + \gamma \hat{q}(\textcolor{red}{S}_{t+1}, \textcolor{red}{A}_{t+1}, \mathbf{w}) - \hat{q}(S_t, A_t, \mathbf{w})) \nabla_{\mathbf{w}} \hat{q}(S_t, A_t, \mathbf{w})$$

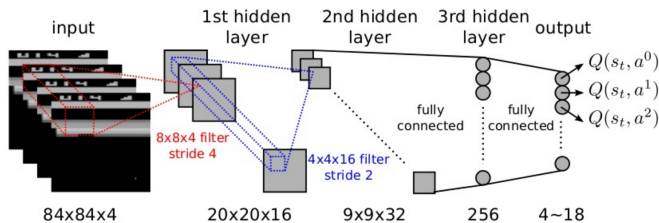
- ③ For Q-learning, the target is the TD target

$$R_{t+1} + \gamma \max_a \hat{q}(S_{t+1}, a, \mathbf{w})$$

$$\Delta \mathbf{w} = \alpha (\textcolor{red}{R}_{t+1} + \gamma \max_a \hat{q}(\textcolor{red}{S}_{t+1}, \textcolor{red}{a}, \mathbf{w}) - \hat{q}(S_t, A_t, \mathbf{w})) \nabla_{\mathbf{w}} \hat{q}(S_t, A_t, \mathbf{w})$$

# DQN for Playing Atari Games

- 1 End-to-end learning of values  $Q(s, a)$  from input pixel frame
- 2 Input state  $s$  is a stack of raw pixels from latest 4 frames
- 3 Output of  $Q(s, a)$  is 18 joystick/button positions
- 4 Reward is the change in score for that step
- 5 Network architecture and hyperparameters fixed across all games

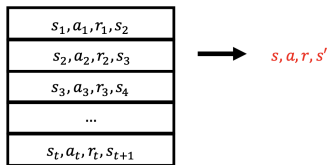


# Q-Learning with Value Function Approximation

- ① Two of the issues causing problems:
  - ① Correlations between samples
  - ② Non-stationary targets
- ② Deep Q-learning (DQN) addresses both of these challenges by
  - ① Experience replay
  - ② Fixed Q targets

# DQNs: Experience Replay

- 1 To reduce the correlations among samples, store transition  $(s_t, a_t, r_t, s_{t+1})$  in replay memory  $\mathcal{D}$



- 2 To perform experience replay, repeat the following
  - 1 sample an experience tuple from the dataset:  $(s, a, r, s') \sim \mathcal{D}$
  - 2 compute the target value for the sampled tuple:  $r + \gamma \max_{a'} \hat{Q}(s', a', \mathbf{w})$
  - 3 use stochastic gradient descent to update the network weights

$$\Delta \mathbf{w} = \alpha \left( r + \gamma \max_{a'} \hat{Q}(s', a', \mathbf{w}) - Q(s, a, \mathbf{w}) \right) \nabla_{\mathbf{w}} \hat{Q}(s, a, \mathbf{w})$$

# DQNs: Fixed Targets

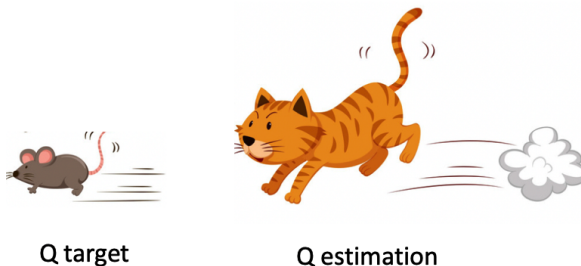
- 1 To help improve stability, fix the target weights used in the target calculation for multiple updates
- 2 Let a different set of parameter  $\mathbf{w}^-$  be the set of weights used in the target, and  $\mathbf{w}$  be the weights that are being updated
- 3 To perform experience replay with fixed target, repeat the following
  - 1 sample an experience tuple from the dataset:  $(s, a, r, s') \sim \mathcal{D}$
  - 2 compute the target value for the sampled tuple:  
 $r + \gamma \max_{a'} \hat{Q}(s', a', \mathbf{w}^-)$
  - 3 use stochastic gradient descent to update the network weights

$$\Delta \mathbf{w} = \alpha \left( r + \gamma \max_{a'} \hat{Q}(s', a', \mathbf{w}^-) - Q(s, a, \mathbf{w}) \right) \nabla_{\mathbf{w}} \hat{Q}(s, a, \mathbf{w})$$



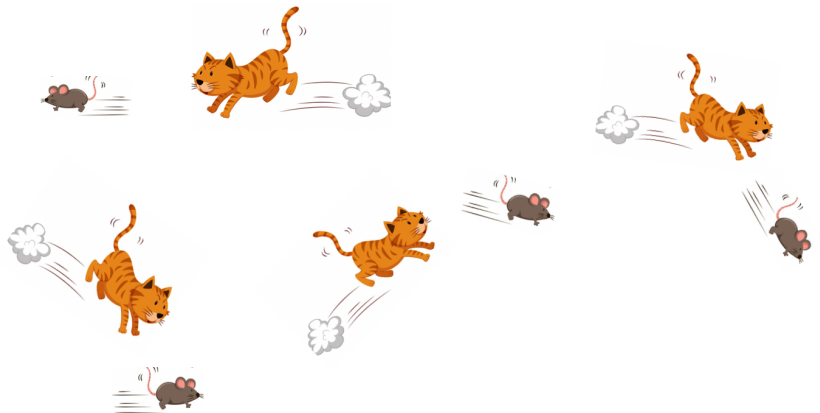
# Why fixed target

- ① In the original update, both  $Q$  estimation and  $Q$  target shifts at each time step
- ② Imagine a cat ( $Q$  estimation) is chasing after a mouse ( $Q$  target)
- ③ The cat must reduce the distance to the mouse



# Why fixed target

- 1 Both the cat and mouse are moving,



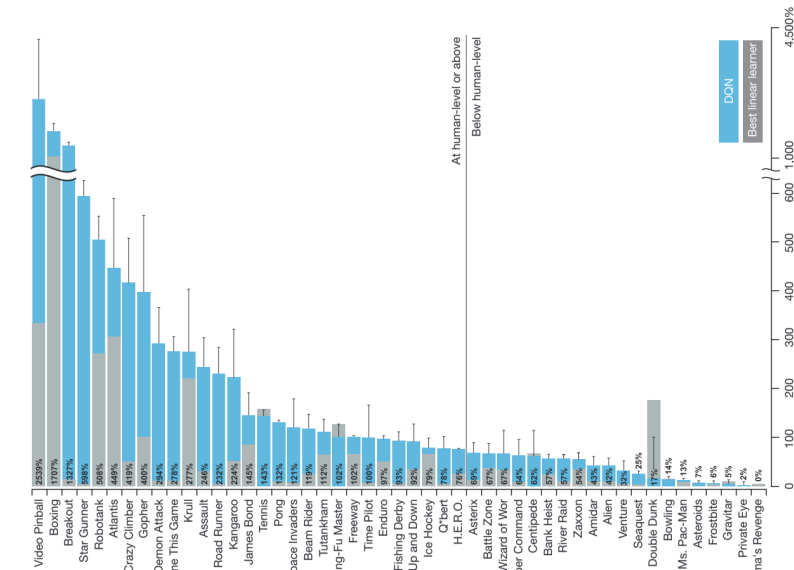
# Why fixed target

- 1 This could lead to a strange path of chasing (an oscillated training history)



- 2 Solution: fix the target for a period of time during the training

# Performance of DQNs on Atari



# Abalation Study on DQNs

## ① Game score under difference conditions

	Replay Fixed-Q	Replay Q-learning	No replay Fixed-Q	No replay Q-learning
Breakout	316.81	240.73	10.16	3.17
Enduro	1006.3	831.25	141.89	29.1
River Raid	7446.62	4102.81	2867.66	1453.02
Seaquest	2894.4	822.55	1003	275.81
Space Invaders	1088.94	826.33	373.22	301.99

# Demo of DQNs

- 1 Demo of deep q-learning for Breakout:  
<https://www.youtube.com/watch?v=V1eYniJ0Rnk>
- 2 Demo of Flappy Bird by DQN:  
<https://www.youtube.com/watch?v=xM62SpKAZHU>
- 3 Code of DQN in PyTorch: <https://github.com/cuhkrlcourse/DeepRL-Tutorials/blob/master/01.DQN.ipynb>
- 4 Code of Flappy Bird:  
<https://github.com/xmfbit/DQN-FlappyBird>

# Summary of DQNs

- 1 DQN uses experience replay and fixed Q-targets
- 2 Store transition  $(s_t, a_t, r_{t+1}, s_{t+1})$  in replay memory  $\mathcal{D}$
- 3 Sample random mini-batch of transitions  $(s, a, r, s')$  from  $\mathcal{D}$
- 4 Compute Q-learning targets w.r.t. old, fixed parameters  $\mathbf{w}^-$
- 5 Optimizes MSE between Q-network and Q-learning targets using stochastic gradient descent

# Improving DQN

- ① Success in Atari has led to a huge excitement in using deep neural networks to do value function approximation in RL
- ② Many follow-up works on improving DQNs
  - ① **Double DQN**: Deep Reinforcement Learning with Double Q-Learning. Van Hasselt et al, AAAI 2016
  - ② **Dueling DQN**: Dueling Network Architectures for Deep Reinforcement Learning. Wang et al, best paper ICML 2016
  - ③ **Prioritized Replay**: Prioritized Experience Replay. Schaul et al, ICLR 2016
- ③ A nice tutorial on the relevant algorithms:  
<https://github.com/cuhkrlcourse/DeepRL-Tutorials>



# Improving DQN: Double DQN

- 1 Handles the problem of the overestimation of Q-values
- 2 Idea: use the two networks to decouple the action selection from the target Q value generation
- 3 Vanilla DQN:

$$\Delta \mathbf{w} = \alpha \left( r + \gamma \max_{a'} \hat{Q}(s', a', \mathbf{w}^-) - Q(s, a, \mathbf{w}) \right) \nabla_{\mathbf{w}} \hat{Q}(s, a, \mathbf{w})$$

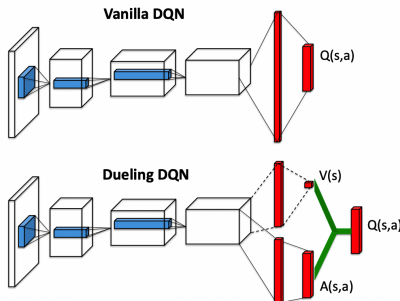
- 4 Double DQN:

$$\Delta \mathbf{w} = \alpha \left( r + \gamma \hat{Q}(s', \arg \max_{a'} Q(s', a', \mathbf{w}), \mathbf{w}^-) - Q(s, a, \mathbf{w}) \right) \nabla_{\mathbf{w}} \hat{Q}(s, a, \mathbf{w})$$

- 5 Code: [https://github.com/cuhkrlcourse/DeepRL-Tutorials/blob/master/03.Double\\_DQN.ipynb](https://github.com/cuhkrlcourse/DeepRL-Tutorials/blob/master/03.Double_DQN.ipynb)

# Improving DQN: Dueling DQN

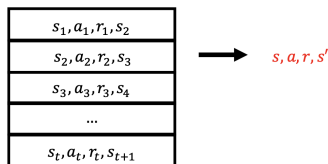
- 1 One branch estimates  $V(s)$ , other branch estimates the advantage for each action  $A(s, a)$ . Then  $Q(s, a) = A(s, a) + V(s)$



- 2 By decoupling the estimation, intuitively the DuelingDQN can learn which states are (or are not) valuable without having to learn the effect of each action at each state
- 3 Code: [https://github.com/cuhkrlcourse/DeepRL-Tutorials/blob/master/04.Dueling\\_DQN.ipynb](https://github.com/cuhkrlcourse/DeepRL-Tutorials/blob/master/04.Dueling_DQN.ipynb)

# Improving DQN: Prioritized Experience Replay

- 1 Transition  $(s_t, a_t, r_{t+1}, s_{t+1})$  is stored in and sampled from the replay memory  $\mathcal{D}$



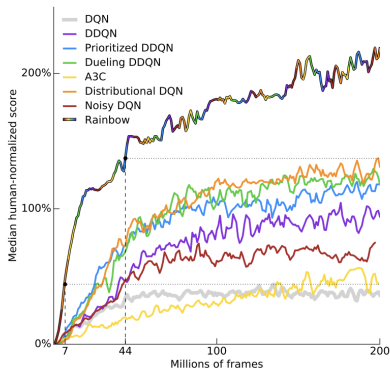
- 2 Priority is on the experience where there is a big difference between our prediction and the TD target, since it means that we have a lot to learn about it.
- 3 Define a priority score for each tuple  $i$

$$p_i = |r + \gamma \max_{a'} Q(s_{i+1}, a', \mathbf{w}^-) - Q(s_i, a_i, \mathbf{w})|$$

- 4 Code: [https://github.com/cuhkrlcourse/DeepRL-Tutorials/blob/master/06.DQN\\_PriorityReplay.ipynb](https://github.com/cuhkrlcourse/DeepRL-Tutorials/blob/master/06.DQN_PriorityReplay.ipynb)

# Improving over the DQN

- 1 Rainbow: Combining Improvements in Deep Reinforcement Learning. Matteo Hessel et al. AAAI 2018.  
<https://arxiv.org/pdf/1710.02298.pdf>
- 2 It examines six extensions to the DQN algorithm and empirically studies their combination



# Homework

- ① Go through the Jupyter tutorial and training your own gaming agent:  
<https://github.com/cuhkrlcourse/DeepRL-Tutorials>
- ② Good resource for your course project
  - ① Make sure it works for simple environment such as Pong
- ③ Homework 2 comes out:  
<https://github.com/cuhkrlcourse/ierg6130-assignment>
- ④ Next week: Policy-based RL