Deep Q Network (DQN)

Annoucements

We will release HW2 tonight (Q-learning, TD, and simulation lemma)

We will release the first reading quiz today

Recap: Q-learning

Tabular Q Learning: maintain a table $\hat{Q}$ of size S x A

$$\hat{Q}(s,a) \Leftarrow \hat{Q}(s,a) + \eta \left(r + \gamma \max_{a'} \hat{Q}(s',a') - \hat{Q}(s,a) \right)$$

Data collection via ϵ-greedy:

W/ prob ϵ, select action uniform randomly

W/ prob $1-\epsilon$, select greedy action $\arg\max_{a} \hat{Q}(s,a)$

Today

Consider large-scale MDPs,

how to estimate $Q^\star(s, a)$ using function approximation (e.g., neural network)

Deep Q-network (DQN) is the earliest example of showing Deep Learning + RL is powerful

doi:10.1038/nature14236

Human-level control through deep reinforcement learning

Volodymyr Mnih[1]*, Koray Kavukcuoglu[1]*, David Silver[1]*, Andrei A. Rusu[1], Joel Veness[1], Marc G. Bellemare[1], Alex Graves[1], Martin Riedmiller[1], Andreas K. Fidjeland[1], Georg Ostrovski[1], Stig Petersen[1], Charles Beattie[1], Amir Sadik[1], Ioannis Antonoglou[1], Helen King[1], Dharshan Kumaran[1], Daan Wierstra[1], Shane Legg[1] & Demis Hassabis[1]

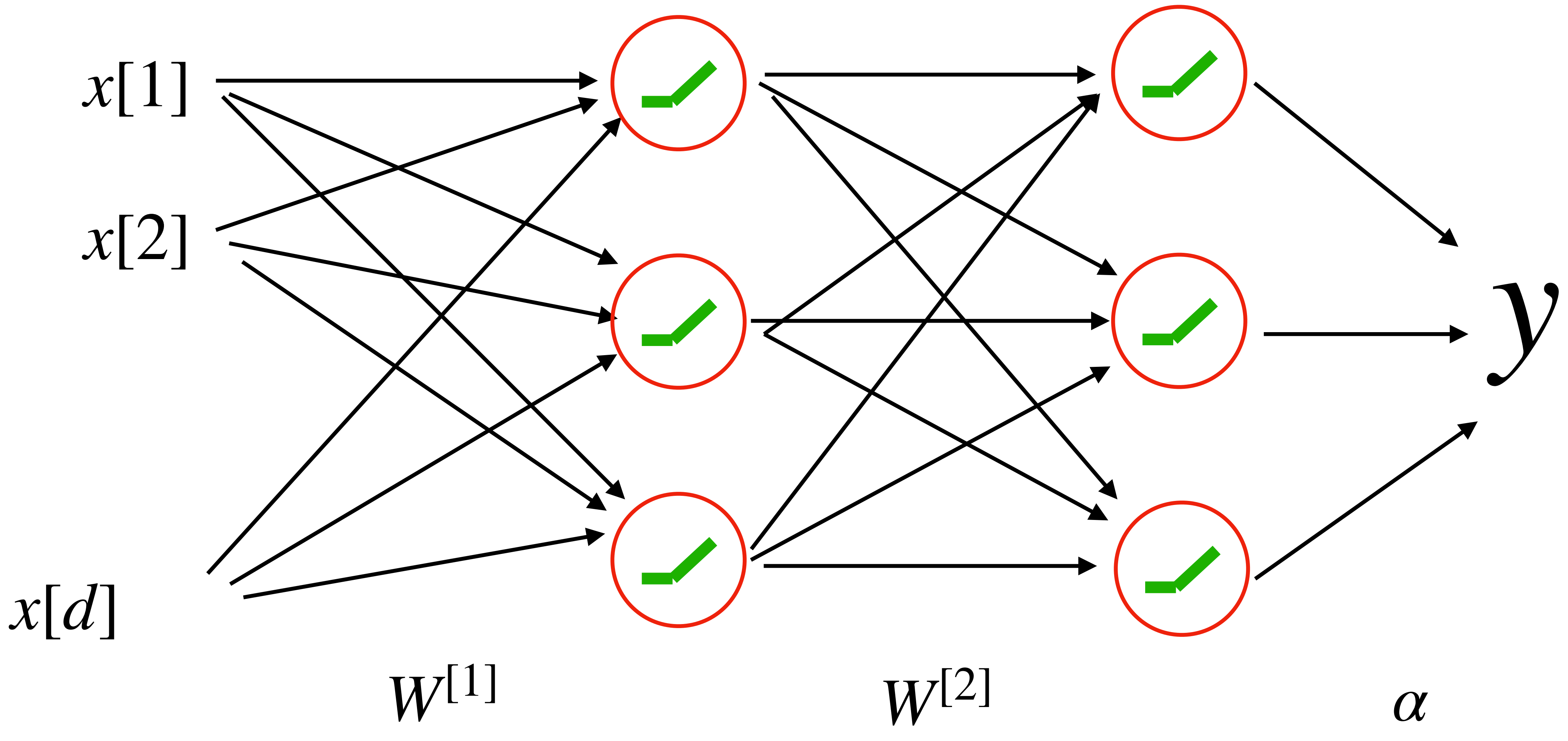
Outline:

1. Q Learning w/ function approximation

2. Replay buffer, batch optimization and target network

Q-Learning w/ function approximation

We will model $Q^\star$ using a function approximator



$$x = [s^\top, a, 1]^\top$$

$$y = \alpha^\top \text{ReLU}\left(W^{[2]}\text{ReLU}\left(W^{[1]}x\right)\right) + b$$

$$Q_\theta(s, a) : S \times A \to \mathbb{R}$$

Assumption: differentiable $\nabla_\theta Q_\theta(s, a)$

(The DQN paper uses ConvNet as s is an image frame of the game)

Attempt 1: Q Learning w/ function approximation

Initialize θ^0. Set initial state $s \in \mathcal{S}$

For t = 0 to T

Take action a based on ϵ-greedy of Q_{θ^t}, get reward r and next state $s' \sim P(\cdot \mid s, a)$

Form Q-target $r + \gamma \max_{a'} Q_{\theta_t}(s', a')$

Update to θ^{t+1}:

Set $s \Leftarrow s'$

Q Learning w/ function approximation

Update parameters using SGD on the Bellman error loss

$$\ell_{be}(\theta) := \left(Q_\theta(s,a) - y\right)^2, \text{ where } y = r(s,a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)} \max_{a'} Q_\theta(s',a')$$

Issues of this simple approach

1. Inefficient — it throws away all historical data
(your network could forget old experiences, i.e., catastrophic forgetting)

2. instablity — Training is quite unstable (we saw it from the past Cartpole Demo)

Outline:

1. Q Learning w/ function approximation

2. Replay buffer, batch optimization and target network

Q-Learning w/ function approximation

First improvement: Replay buffer

$$\mathscr{D}_{rb} = \begin{bmatrix} \cdots \\ (s, a, r, s') \\ \cdots \\ \cdots \end{bmatrix}$$

A dataset that contains all historical
State-action-reward-next state tuples

Q-Learning w/ function approximation

With replay buffer, we can use mini-batch SGD to update Bellman error loss

Given Q_{θ^t} and replay buffer $\mathscr{D}_{rb}$, randomly sample a mini-batch $\mathscr{B}$ from $\mathscr{D}_{rb}$

$$\theta^{t+1} = \theta^t - \eta \frac{1}{|\mathscr{B}|} \sum_{(s,a,r,s') \in \mathscr{B}} \left(Q_{\theta^t}(s, a) - r - \max_{a'} Q_{\theta^t}(s', a') \right) \nabla_\theta Q_{\theta_t}(s, a)$$

Q-Learning w/ function approximation

Second improvement: Target network (making Q learning more stable)

Recall that Q learning can be understood as running SGD on an **evolving** loss function

$$\ell_{be}(\theta) := \left(Q_\theta(s,a) - y\right)^2, \text{ where } y = \underbrace{r(s,a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)} \max_{a'} Q_\theta(s',a')}_{\text{regression target}}$$

Source of instability: target changes immediately whenever we update θ

Q-Learning w/ function approximation

Second improvement: Target network (making Q learning more stable)

Introducing target network $Q_{\tilde{\theta}}$ to **slow down** the evolution of the BE loss

(e.g., set $\tilde{\theta}$ as a copy of an older version of θ)

$$\ell_{be}(\theta) := \left(Q_\theta(s,a) - y\right)^2, \text{ where } y = r(s,a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)} \max_{a'} Q_{\tilde{\theta}}(s',a')$$

Use a target network that slowly catchs up θ

Attempt 2: Deep Q networ (DQN)

Initialize θ and replay buffer $\mathcal{D}_{rb}$. Set $\tilde{\theta} = \theta$, Set initial state $s \in \mathcal{S}$

While true:

Take action a based on ϵ-greedy of Q_θ, get reward r and next state $s' \sim P(\cdot \mid s, a)$

Add (s, a, r, s') to $\mathcal{D}_{rb}$

Sample mini-batch $\mathcal{B}$ from $\mathcal{D}_{rb}$

Update parameters:

$$\theta \Leftarrow \theta - \eta \frac{1}{|\mathcal{B}|} \sum_{(s,a,r,s') \in \mathcal{B}} \left(Q_\theta(s, a) - r - \max_{a'} Q_{\tilde{\theta}}(s', a') \right) \nabla_\theta Q_\theta(s, a)$$

Every C step, set $\tilde{\theta} = \theta$

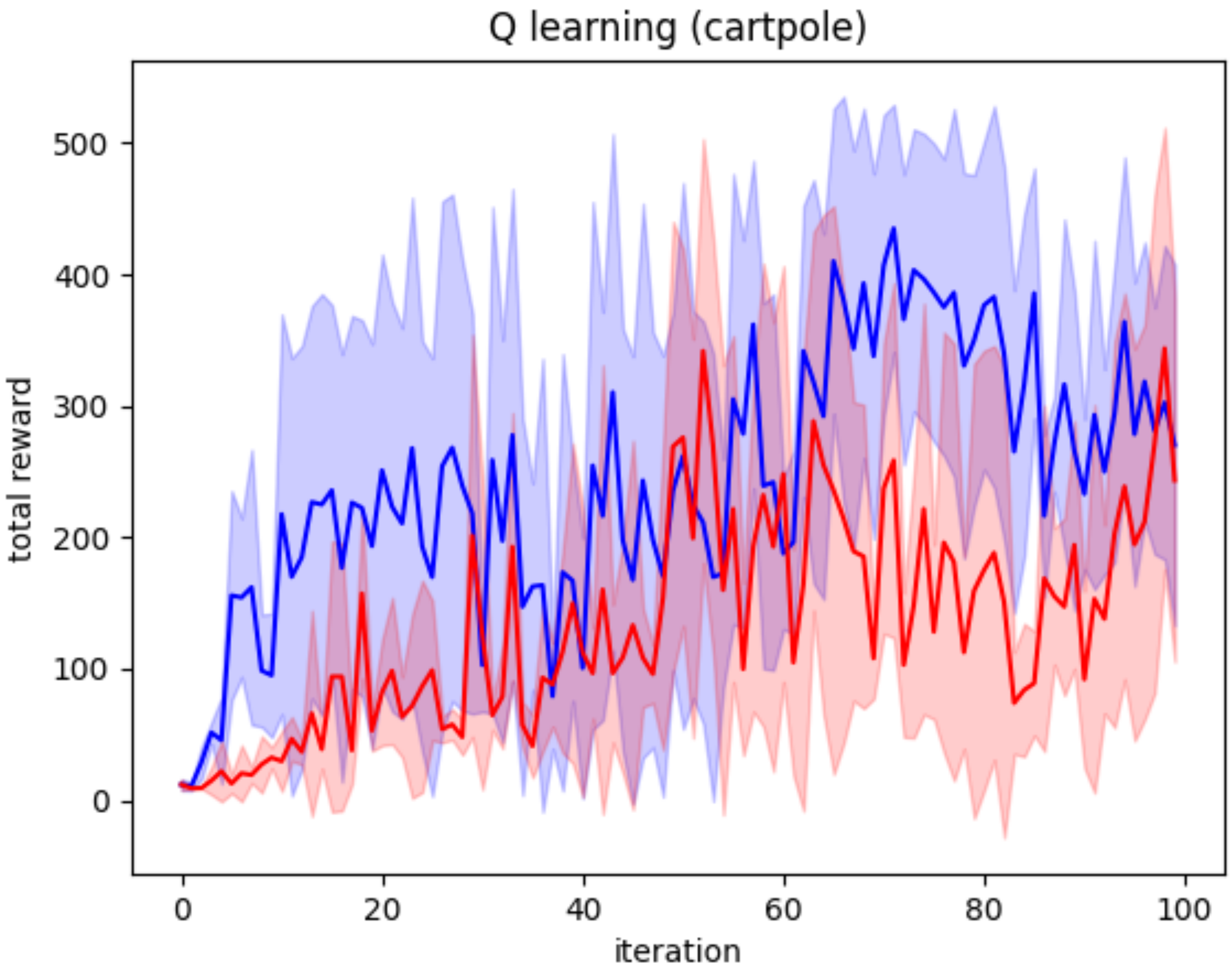
When C is large…

DQN is performing standard regression between two target updates..

$$\min_{\theta} \sum_{s,a,r,s' \in \mathcal{D}_{rb}} \left(Q_\theta(s,a) - \left(r + \max_{a'} Q_{\tilde{\theta}}(s',a') \right) \right)^2$$

Q: What is the Bayes optimal of this regression problem?

DQN vs Naive Q-learning



BLUE: DQN

RED: Q-learning

Summary

1. Using function approximation to handle large state space

2. Making Q-learning closer to the supervised learning (i.e., regression) framework:

- Replay buffer + mini-batch SGD
- Target network to simulate a standard regression setting