CS 224N Assignment 2: February 10, 2018

Junwon Park (junwonpk@stanford.edu)

1. Introduction

2. Q-learning

3. Setup – TestEnv

1. What's the maximum sum of rewards that can be achieved in a single episode in the TestEnv? (γ = 1)

The maximum sum of rewards that can be achieved in a single episode in the TestEnv is 4.1.

One episode is 5 time steps and begins in state 0.
t = t: (s, a, s', r), total reward after t
t = 1: (0, 2, 2, 0.0) 0.0
t = 2: (2, 1, 1, 2.0) 2.0
t = 3: (1, 2, 2, 0.0) 2.0
t = 4: (2, 1, 1, 2.0) 4.0
t = 5: (1, 0, 0, 0.1) 4.1

2. Implement get action and update functions in q1 schedule.py. Test your implementation by running python q1 schedule.py.

Done.

4. Linear Approximations

1. Show that Equations (1) and (2) are exactly the same when $\hat{q}(s,a,\boldsymbol{w}) = \boldsymbol{w}^T x(s,a)$, where $\boldsymbol{w} \in \mathbb{R}^{|S||A|}$, $x: S \times A \to \mathbb{R}^{|S||A|}$, and

$$x(s,a)_{s',a'} = \begin{cases} 1 \; if \; s' = s, a' = a \\ \quad 0 \; otherwise \end{cases}$$

Proof:
Let $\hat{q}(s,a,\boldsymbol{w}) = \boldsymbol{w}^T x(s,a)$, where $\boldsymbol{w} \in \mathbb{R}^{|S||A|}$, $x: S \times A \to \mathbb{R}^{|S||A|}$, and

$$x(s,a)_{s',a'} = \begin{cases} 1 \; if \; s' = s, a' = a \\ \quad 0 \; otherwise \end{cases}$$

and consider

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

. We will show that this equation is same as

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

from which the result immediately follows.

Since $\hat{q}(s,a,\boldsymbol{w}) = \boldsymbol{w}^T x(s,a)$, we have

$$\boldsymbol{w} = \boldsymbol{w} + \alpha\left(r + \gamma \max_{a'} \hat{q}(s',a',\boldsymbol{w}) - \hat{q}(s,a,\boldsymbol{w})\right) \nabla_{\boldsymbol{w}} \hat{q}(s,a,\boldsymbol{w})$$
$$= \boldsymbol{w} + \alpha\left(r + \gamma \max_{a'} \boldsymbol{w}^T x(s',a') - \boldsymbol{w}^T x(s,a)\right) \nabla_{\boldsymbol{w}} \boldsymbol{w}^T x(s,a)$$

Since $\nabla_{\boldsymbol{w}} \boldsymbol{w}^T x(s,a) = x(s,a)$,

$$\boldsymbol{w} = \boldsymbol{w} + \alpha \left(r + \gamma \max_{a'} \boldsymbol{w}^T x(s', a') - \boldsymbol{w}^T x(s,a) \right) x(s,a)$$

We multiply the equation by x(s,a) and get

$$x(s,a)\boldsymbol{w} = x(s,a)\boldsymbol{w} + \alpha \left(r + \gamma \max_{a'} \boldsymbol{w}^T x(s', a') - \boldsymbol{w}^T x(s,a) \right) x(s,a)^2$$

Since $x(s,a)$ is a one-hot feature vector,

$$w_{s,a} = w_{s,a} + \alpha \left(r + \gamma \max_{a'} w_{s',a'} - w_{s,a} \right)$$

at (s,a).

Both Q and w are vectors with the same dimensions, where (s,a) is an index. Updating $w_{s,a}$ is equal to updating Q(s,a). Therefore, the two equations are exactly the same, as required.

2. Derive the gradient with regard to the value function parameter $\boldsymbol{w} \in \mathbb{R}^n$ given $\hat{q}(s, a, \boldsymbol{w}) = \boldsymbol{w}^T x(s,a)$ for any function $x(s,a) \longmapsto x \in \mathbb{R}^n$ and write the update rule for w.

$$\nabla_{\boldsymbol{w}} \boldsymbol{w}^T x(s,a) = x(s,a)$$

Update Rule for w:

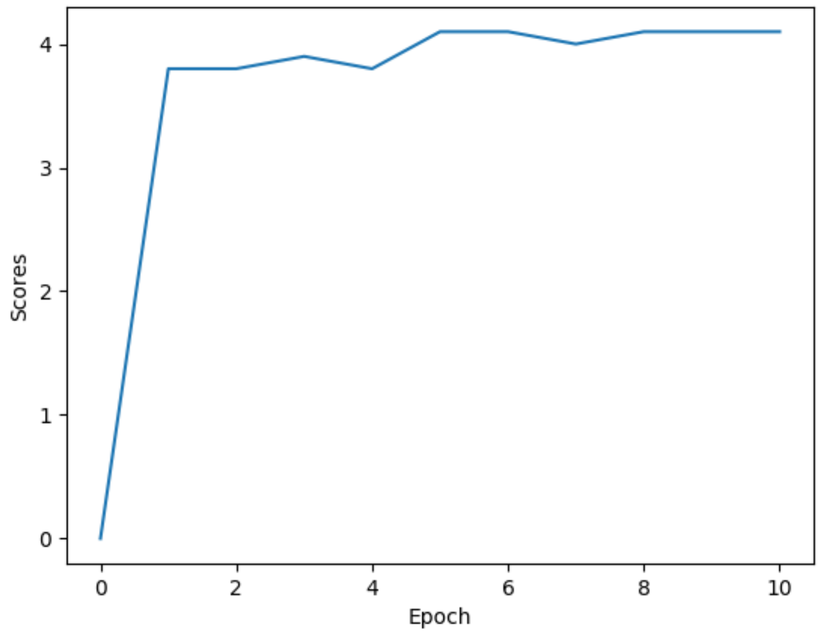
$$\boldsymbol{w} = \boldsymbol{w} + \alpha \left(r + \gamma \max_{a'} \boldsymbol{w}^T x(s', a') - \boldsymbol{w}^T x(s,a) \right) x(s,a)$$

3. Implement linear approximation with Tensorflow. Fill in the relevant functions in q2_linear.py.

Done.

4. Do you reach the optimal achievable reward on the test env? Attach the plot scores.png.

Yes. Optimal achievable reward of 4.1 was achieved on the test env.



5. Implementing DeepMind's DQN

1. Why do we use the last 4 time steps as input to the network for playing Atari games?

We use 4 time steps as input in order to encode information about velocity (since the ball at a given position might be flying in any direction or with any speed and we wouldn't know with a stationery image).

2. In DQN, what would be a benefit of experience replay? What would be a benefit of the target network?

Benefits of experience replay and target network are that they can mitigate the following two problems: correlations between samples and non-stationery targets of value function approximation.

Experience replay helps **removing correlation** between updates by storing a replay buffer from prior experience. It samples an experience tuple (s, a, r, s') from the buffer, computes target value for the sample d s, and uses SGD to update network weights.

Target network **improves stability** by fixing the target network weights used in target calculation for multiple updates.

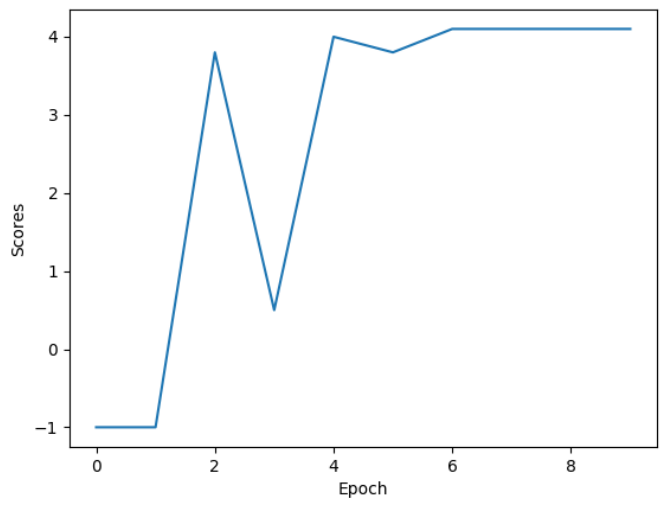
3. In DQN, what would be a benefit of representing the Q function as $\hat{q}(s, \boldsymbol{w}) \in \mathbb{R}^K$ where K is the total number of actions, instead of $\hat{q}(s, a, \boldsymbol{w})$ as a scalar for each (s, a) pair?

If we represent Q function as $\hat{q}(s, a, \boldsymbol{w})$ as a scalar for each (s,a) pair, then we need lots of computations individually done. If we build a vector $\hat{q}(s, \boldsymbol{w}) \in \mathbb{R}^K$ to represent the Q function, then we can use parallel processing (like multithread/GPU) to compute values through similar computations at the same time. Therefore, the benefit is computation efficiency.

4. Implement the deep Q-network as described in [2] by filling get_q_values_op in q3_nature.py. Test implementation on the test environment by running python q3_nature.py.

Done.

5. Attach the plot of scores. Compare the model with linear approximation. What about the final performance? Training time?

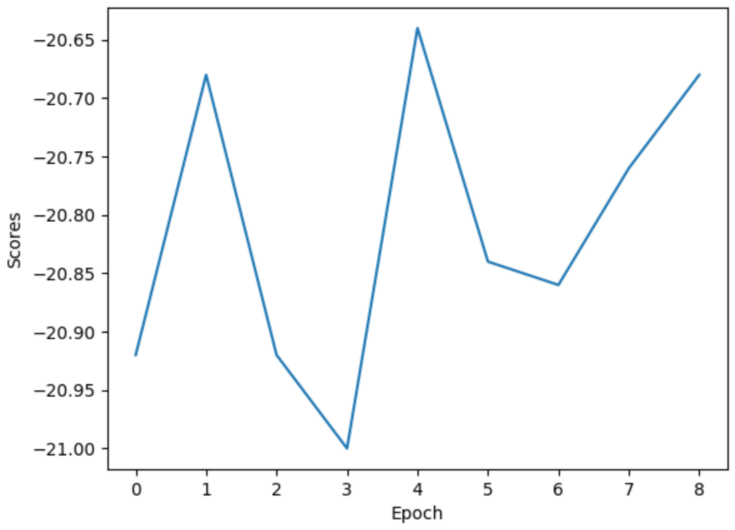


Compared to the linear approximation, DQN takes more epochs to reach the optimal or close-to-optimal score. Final performance is the same: both achieved the optimal score of 4.1 in the test environment. For training time, DQN took 26 seconds and linear approximation took 12 seconds, so DQN took more than double the time to train.

6. What's the number of parameters of this model (for Pong) if the input to the Q network is a tensor of shape (80, 80, 4) with "SAME" padding (with zero padding, default in Tensorflow)? Compare with linear approximation.

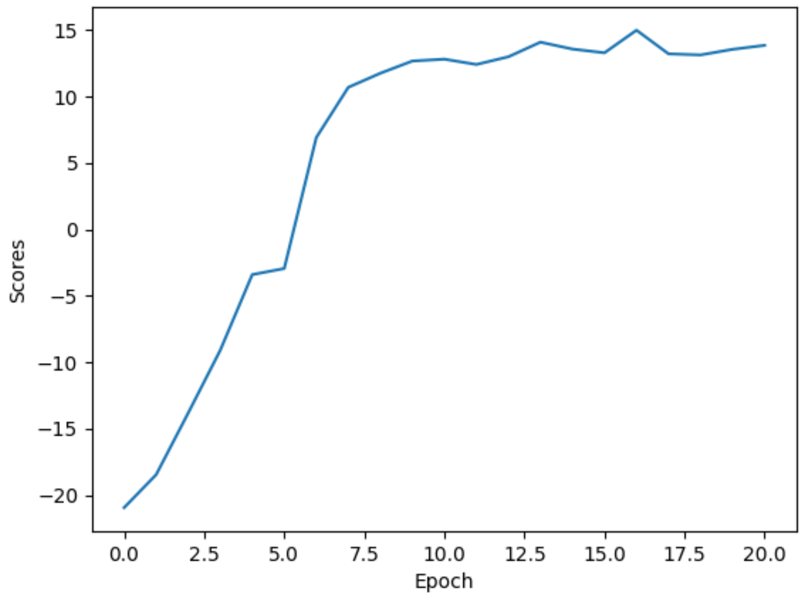
Number of parameters of the DQN is
(8*8*4) * 32 + 32
+ (4*4*32) * 64 + 64
+ (3*3*64) * 64 + 64
+ (10*10*64) * 512 + 512
+ 512 * 6 + 6
= 3,358,374
Linear approximation is 80 * 80 * 4 * 6 + 6 = 153,606.
DQN has a much larger set of parameters.

7. Launch linear approximation on pong with q4_train_atari_linear.py. What do you notice?

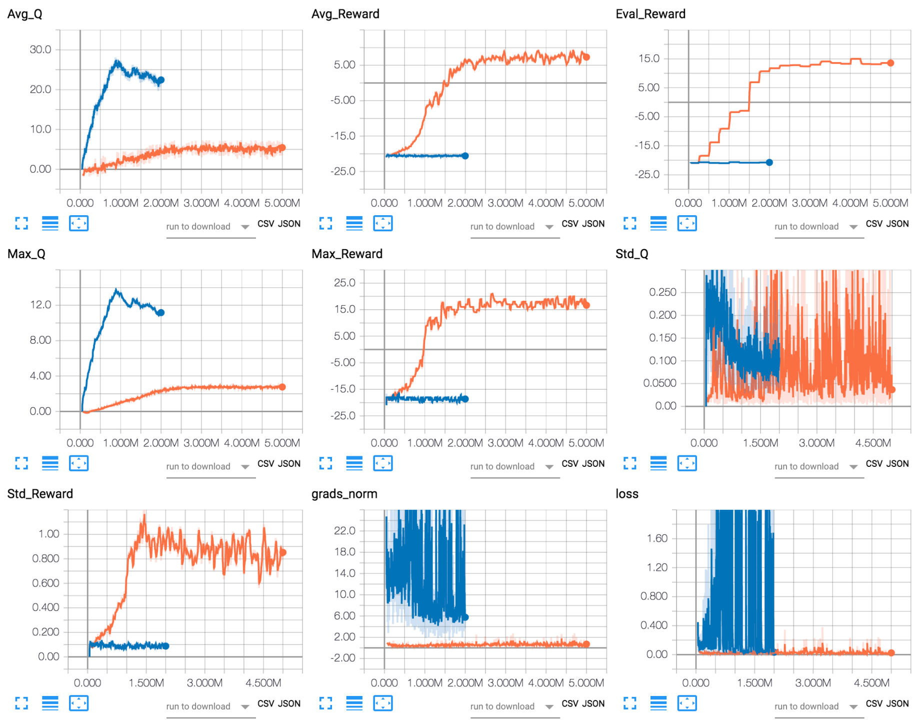


Linear approximation failed to achieve a good score on pong even after 2 million time steps of training.

8. Run python q5_train_atari_nature.py.



9. Compare the performance with previous linear Q-network/approximation. How can you explain the gap in the performance?



Orange is DQN and blue is linear approximation.

DQN achieved a great performance whereas the previous linear approximation did not. The gap may come from the fact that this is a complex task and linear approximation does not have enough parameters to capture the complexity of the game, whereas DQN does. We can see from Tensorboard that linear approximation is failing to do loss minimization.

6. Bonus: Architecture of the Network

7. Uniform Approximation

1. Choose w and b to implement Boolean AND. Choose w and b to implement Boolean OR.

AND:

$$\begin{cases} w_1 + w_2 + b > 0 \\ w_1 + b \le 0 \\ w_2 + b \le 0 \\ b \le 0 \end{cases}$$

w = [1, 1], b = -1.5

OR:

$$\begin{cases} w_1 + w_2 + b > 0 \\ w_1 + b > 0 \\ w_2 + b > 0 \\ b \le 0 \end{cases}$$

w = [1, 1], b = -0.1

2. Under the same conditions, what Boolean function of two variables cannot be represented? Briefly explain why the function cannot be implemented.

XOR. For XOR to be implemented, we need **w** and b that can solve the following system of inequalities:

$$\begin{cases} w_1 + w_2 + b \le 0 \\ w_1 + b > 0 \\ w_2 + b > 0 \\ b \le 0 \end{cases}$$

Last inequality means b must be 0 or negative.

$$w_1 + w_2 + b = (w_1 + b) + (w_2 + b) - b$$

We know that $(w_1 + b)$ and $(w_2 + b)$ are both positive, so $(w_1 + b) + (w_2 + b)$ must be positive. We know that b is 0 or negative. Positive minus 0 or negative cannot be 0 or negative. Therefore, if below three inequalities in the system of inequalities are true, then first inequality leads to contradiction. Therefore, XOR cannot be represented.

3. We now allow a single layer of hidden units. Construct a neural network that represents the Boolean function from part 2.

Let there be two hidden units: z_1 and z_2. We design a neural network as follows:
Output Layer: $y = f(w^T \boldsymbol{z} + b)$ is an AND operation designed as part a: **w** = [1, 1], b = -1.5
Hidden Unit 1: $z_1 = f(w^T \boldsymbol{x} + b)$ is an OR operation designed as part a: **w** = [1, 1], b = 0
Hidden Unit 2: $z_2 = f(w^T \boldsymbol{x} + b)$ is a NOT AND operation designed as: **w** = [-1, -1], b = -1.5

Since XOR is equivalent with (OR) AND (NOT AND), the above neural network represents XOR.

4. Describe a general procedure to construct a neural network with single hidden layer to represent any boolean function.

Any Boolean function can be written using only AND, OR, NOT AND, and NOT OR. AND and OR can be represented as shown in part 1, and NOT AND and NOT OR can be generated by flipping signs for w and b values from representation from part 1. Every propositional formula can be converted into conjunctive normal form. Therefore, we could design a neural network where output layer is a conjunction of the single hidden layer and the single hidden layer could encode the Boolean clauses being combined by conjunction.

5. Why would you want to consider neural networks with multiple hidden layers?

Though deep neural networks are no more expressive than one layer neural networks, multiple hidden layers can be exponentially more compact in terms of number of nodes to needed to represent a function. This leads to memory, computation, and data efficiency.