Sofia Lima

15-686 Neural Computation

Dr. Tai-Sing Lee

30 April, 2023

Problem Set 6: Reinforcement Learning

Setup Download Deep Learning Toolbox (already installed)

Q-Learning tries to quantify the expected total reward as a function of state-action pair, $Q(S, a)$.

Two approaches to estimate the Q function: (Part 1) Simply discretizing all possible state and all possible actions; (Part 2) Use a neural network to take in the state and output the Q-function for each possible actions.

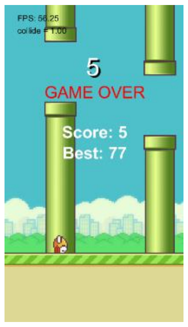
All screen recordings can be found in my HW6_vids google drive folder: https://drive.google.com/drive/folders/1i93iNXyhQ2OzB356nW2Q3A0xBiuFucxe?usp=share_link

I also provide links to individual videos corresponding to the questions below. Importantly, for all comparisons, these videos capture trainings without save-resuming the weights for Q-learning so that I can make accurate conclusions about the effects of each hyperparameters on learning in part 1c and part 2. For this same reason, I conduct the hyperparameter tuning such that all other configurations in part 1c are the same as in part1b.

Part 1: To Flap or Not to Flap – Q-learning with Tables (6 pts)

In this part of the assignment, we implement deep reinforcement learning. Perform Q-learning with two Q tables.

1a: **Flappy Bird Simulator** (0.5 pts)



Use a Matlab interface game simulator. Screenshot results from manual mode.

Shown to the right is a screenshot from manual mode surviving for at least 4 pillars. 5 was my best score. I use the relative positions of the bird and pipes to make my decision to flap or not.

1b: **Implementing Q-learning** (2.5 pts)

Reinforcement learning deals with optimizing action policy without knowing the exact objective function. By trial-and-error, the agent explores consequences.

State Space State is distance which are parameters in `flap` function. Based on the Q-values evaluated at the two time steps, we can update the Q table of the action a_t that is chosen,

$$Q(S_t, a_t)_{new} = Q(S_t, a_t)_{current} + \eta \left\{ r_{t+1} + \max_{a'} \gamma Q(S_{t+1}, a')_{current} - Q(S_t, a_t)_{current} \right\}$$

Upload a screen recording to youtube (or google drive) and embed the url. Include `Q.mat`.

Task List: Required Implementation Checklist

1. Task 1: (0.5 pt) Construct Q table in `flappybird_qtable.m`.
2. Task 2: (0.5 pt) Compute the State Next State (S_{t+1}) in both `flappybird_qtable.m` and `test_flappybird_qtable.m`.
3. Task 3: (0.5 pt) Choose the action based on the Q table in both `flappybird_qtable.m` and `test_flappybird_qtable.m`. Although during test time, your action should be strictly maximizing the Q value, it's not necessary the case during training (exploration v.s. exploitation trade-off).
4. Task 4: (1 pt) Update the TD learning rule in `flappybird_qtable.m`:

$$Q(S_t, a_t)_{new} = Q(S_t, a_t)_{current} + \eta \left\{ r_{t+1} + \max_{a'} \gamma Q(S_{t+1}, a')_{current} - Q(S_t, a_t)_{current} \right\}$$

Hint 1: State variable read-out distX and distY

For task 2, in the simulator `flappybird_qtable.m`, compute the state variables which is the difference between the position of the bird and pipe. The bird's x (horizontal) and y (vertical) positions are stored in `Bird.ScreenPos(1)` and `Bird.ScreenPos(2)`; `pipe_x` and `pipe_y` are given. To constrain the model space, we divide the distance by 4 and constrain the distance to be an integer 1 and 200.

Hint 2: Better understanding of the simulator

The while loop reduces the burden of rendering every frame. There are 3 while loops:

1. The first is located right below "`% Main Game`" and right above "`disp('—BEGIN of NEW EPISODE—')`". Each iteration is one episode where the bird initialize and fly. When the bird crashes, an episode is ended and a new episode would start.
2. The second, right under "`% ONE EPISODE ROLL OUT`", controls rendering.
3. The third is located below "`% Please see Hint1 in Part1b for understanding of the while loop`". Each time step, the agent will decide the action based on the state value. Each state will decide the action executed next. Each iteration checks if the agent is still alive, based on the actual reward in the current state.

Hint 3: Suggested Set of Parameters

To help Task 4, some baseline parameters that has higher probability to converge are provided. For learning rates, η can be 0.5. For reward signal, +10 for stay alive and -1000 for crash. The state space can be discretized and downscaled by factor of 4 such that the search space would be smaller. Memory size would influence learning. For Adam optimization learning rate, 0.01 works well. Increase the neural network's hidden layer would provide additional fitting power hence can make the agent learn fast.

Hint 4: Training and Testing

`test_flappybird_qtable.m` is a separate testing script. The training script `flappybird_qtable.m` will generate a `Q.mat` file which contains the trained Q-table, the training episodes and the total time (seconds) used to train the Q-table. Save-resuming with `flappybird_qtable.m` and `Q.mat` will save neural network weights into `nn.mat` file. Complete Task 2 and Task 3 to load a specific sequence of scenes so that everyone's Q table can be tested with the same sequence for evaluation purposes.

Shown below are the codes to accomplish tasks 1 through 4.

1.
```
Q = zeros(500,500,2);
```

2.
```
% horizontal distance from tube lower-edge
dx = round((tube_x - agent_x)/4);
dx = max(dx,1);
% disp(dx)

% vertical distance from tube right edge
dy = round((tube_y - agent_y)/4);
dy = max(dy,1);
% disp(dy)
```

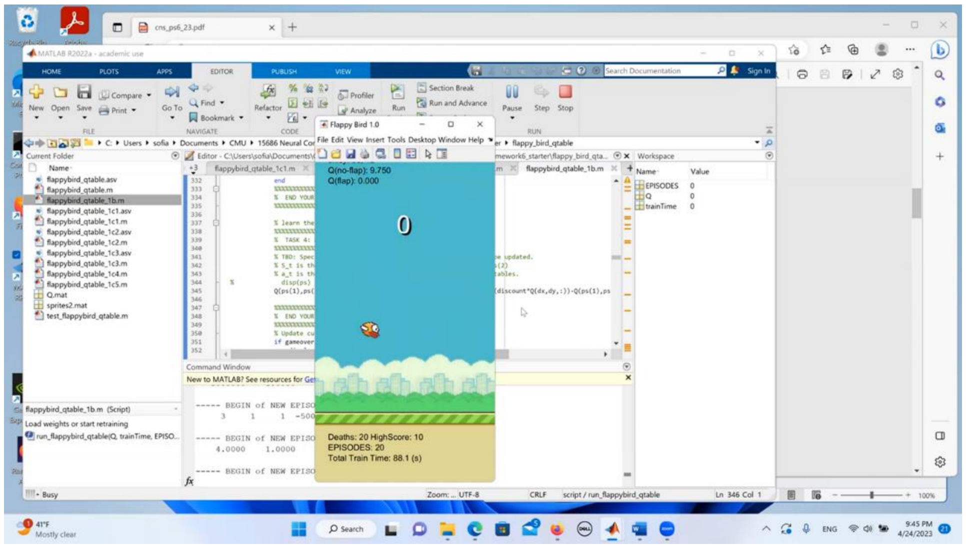
3.
```
if ((Q(dx,dy,2)>Q(dx,dy,1)) && ~gameover)
        a = 2; % 2 is flap
        FlyKeyStatus = true;
      else
         a = 1; % 1 is no flap
      end
```

4.
```
Q(ps(1),ps(2),ps(3)) = Q(ps(1),ps(2),ps(3)) + lr*(reward+max(discount*Q(dx,dy,:))-Q(ps(1),ps(2),ps(3)));
```

Below is the link to the screen recording of training after implementing the required tasks for Q-Learning with Q-tables:

https://drive.google.com/file/d/1iXNTzO6XnAaJCzZ2yCXA0Pu2STCgCVZq/view?usp=share_link

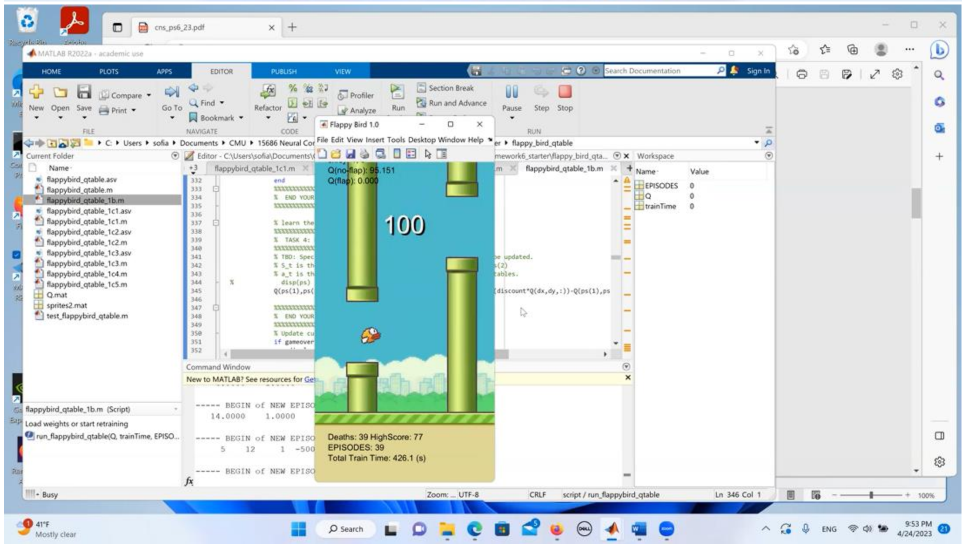
At the 20th death, the bird reaches 10 pipes. After 10 minutes of training, the bird reaches over 100 pipes. These are the baselines I use to make comparisons in part c.


Q(no-flap): 9.750
Q(flap): 0.000

0

```
----- BEGIN of NEW EPISO
     3     1     1  -500

----- BEGIN of NEW EPISO
    4.0000    1.0000

----- BEGIN of NEW EPISO
```

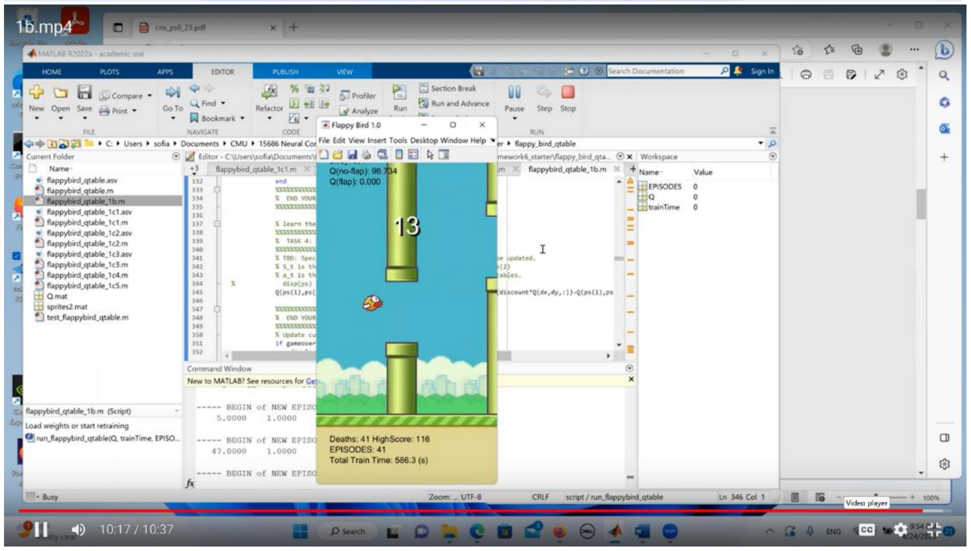
Deaths: 20 HighScore: 10
EPISODES: 20
Total Train Time: 88.1 (s)


Q(no-flap): 95.151
Q(flap): 0.000

100

```
----- BEGIN of NEW EPISO
   14.0000    1.0000

----- BEGIN of NEW EPISO
     5    12     1  -500

----- BEGIN of NEW EPISO
```

Deaths: 39 HighScore: 77
EPISODES: 39
Total Train Time: 426.1 (s)


1b.mp4

Q(no-flap): 96.734
Q(flap): 0.000

13

```
----- BEGIN of NEW EPISO
    5.0000    1.0000

----- BEGIN of NEW EPISO
   47.0000    1.0000

----- BEGIN of NEW EPISO
```

Deaths: 41 HighScore: 116
EPISODES: 41
Total Train Time: 586.3 (s)

10:17 / 10:37

1c: **Exploring design issues** (3 pts)

1. Reward Model: (0.5 pts)
   (Recommendations) Use the provided constants `PIPEGAPSIZE` and `BIRDHEIGHT` to calculate perceptual measurement. Reason based on $s(t)$ to obtain additional perceptual hint to add the learning (e.g. add additional reward or punishment if y falls within or outside certain range). Use perception measurements (`distX`, `distY`, and the scores) to derive other measurements and assign reward r to each measurement.

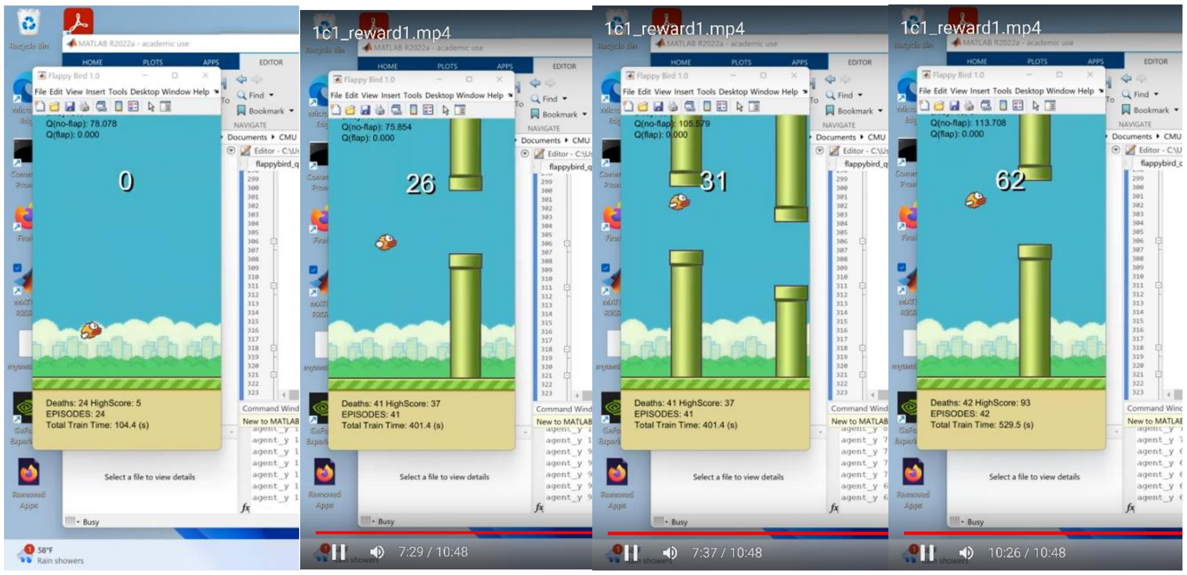
Shown below are the codes to accomplish additional reward as a function of bird position relative to the pipe; specifically, for the bird being below the top pipe, and above the bottom pipe.

reward = 10 + (GapY(1) < agent_y) + (GapY(2) > agent_y);

Below is the link to the screen recording of training after implementing this reward function:

https://drive.google.com/file/d/1c61cyo37PuNWXGfu6jgFGVe0D2ogMEKu/view?usp=share_link

The bird was able to reach about 90 pipes within 10 minutes of training with this reward scheme and was consistently similar to baselines in 1b.



```
Q(no-flap): 78.078
Q(flap): 0.000
Deaths: 24 HighScore: 5
EPISODES: 24
Total Train Time: 104.4 (s)
```

```
Q(no-flap): 75.854
Q(flap): 0.000
Deaths: 41 HighScore: 37
EPISODES: 41
Total Train Time: 401.4 (s)
7:29 / 10:48
```

```
Q(no-flap): 105.579
Q(flap): 0.000
Deaths: 41 HighScore: 37
EPISODES: 41
Total Train Time: 401.4 (s)
7:37 / 10:48
```

```
Q(no-flap): 113.708
Q(flap): 0.000
Deaths: 42 HighScore: 93
EPISODES: 42
Total Train Time: 529.5 (s)
10:26 / 10:48
```

Other reward schemes I tried:

Reward scheme 2 (scale reward components):

```
if ~gameover
        reward = 10 + 10* (GapY(1) < agent_y) + 10* (GapY(2) > agent_y);
    else
        reward = -1000;
    end
```

Reward scheme 3 (scale reward relative to penalty):

```
if ~gameover
        reward = 100 + 10* (GapY(1) < agent_y) + 10* (GapY(2) > agent_y);
      else
        reward = -1000;
      end
```

Reward scheme 4 (add penalty as function of position):

```
if ~gameover
        reward = 10 + (GapY(1) < agent_y) + (GapY(2) > agent_y);
      else
        reward = -1000 - 100*(agent_y < GapY(1)) - 100*(agent_y > GapY(2));
      end
```

Reward scheme 5 (add reward for AND):

```
if ~gameover
        reward = 10 + ((GapY(1) < agent_y) && (GapY(2) > agent_y)) + (GapY(1) < agent_y) + (GapY(2) > agent_y);
      else
        reward = -1000
      end
```

All performed worse than baselines. Training videos for other settings are in the google drive folder link provided on the first page of this submission.

2. Sparse Reward: (0.5 pts)
   Sparse reward is one type of reward – each time the bird passes through the pipe, it will only get a reward probabilistically. Try at least 5 probabilities. What is the minimum probability of reward (among the ones you chose to try) that will allow the bird to learn? Your access to $s(t)$ can be considered sensory perception.

Shown below are the codes to accomplish sparse reward. The MATLAB function randomsample was used so that no reward (0) was chosen 80% of the time and reward (10) was chosen 20% of the time.

```
reward = 0 + randsample([0,10],1,true,[0.8,0.2]);
```

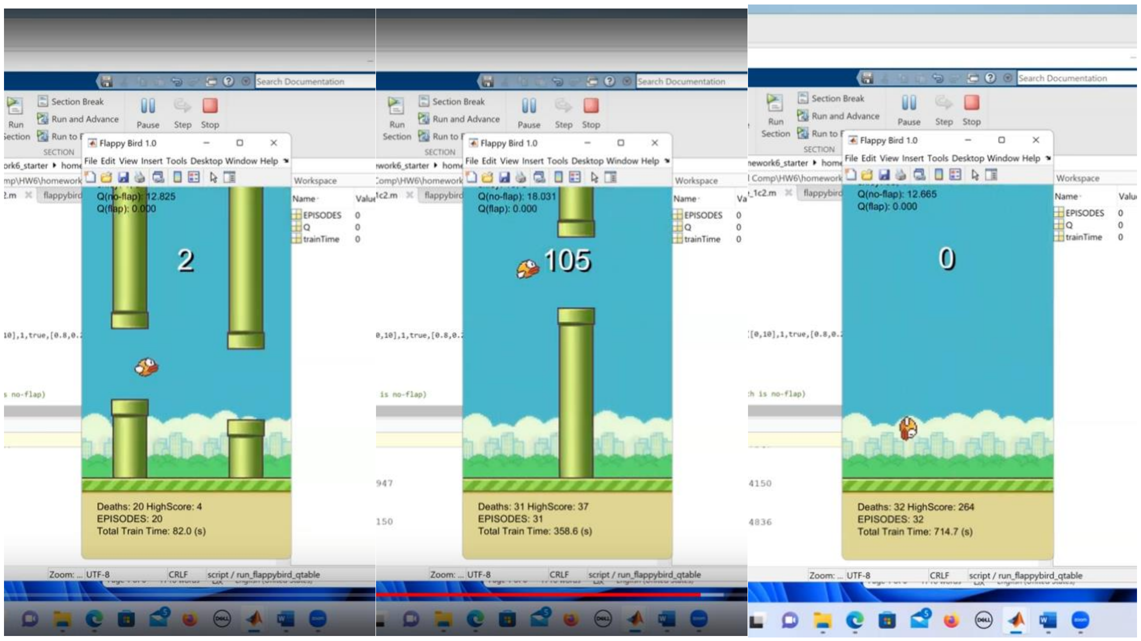
Below is the link to the screen recording of training after implementing this reward function:

https://drive.google.com/file/d/1G1pUHmZ-MlwSW-A_Yw8KEC3lX5R6tVf9/view?usp=share_link

The training results varied for a given value of reward probability but with 20% probability of reward, the rate of learning was similar to baselines in part 1b.

Other reward probability values that I tried as the last argument of the randomsample function were: [0.9,0.1], [0.85,0.15], [0.82,0.18], [0.81,0.19], [0.2,0.8]. Training videos for other settings are in the google drive folder link provided on the first page of this submission.

I tried many around [0.8,0.2] because Flappy was able to reach over 260 pipes in 32 deaths with this configuration. All values above [0.9,0.1] performed similar to baseline, but with such low reward probability as 10%, the bird is unable to learn well at all and can not reach 10 pipes in even 60 deaths.



```
Q(no-flap): 12.825
Q(flap): 0.000
Deaths: 20 HighScore: 4
EPISODES: 20
Total Train Time: 82.0 (s)
```

```
Q(no-flap): 18.031
Q(flap): 0.000
Deaths: 31 HighScore: 37
EPISODES: 31
Total Train Time: 358.6 (s)
```

```
Q(no-flap): 12.665
Q(flap): 0.000
Deaths: 32 HighScore: 264
EPISODES: 32
Total Train Time: 714.7 (s)
```

3. Learning rate η and discount factor γ. (0.5 pts)
   Using η=0.5, γ=0.9 may not be optimal. What difference these parameters make?

I tried 6 configurations for learning rate and discount factor.

Configuration 1 (decrease discount factor):

lr = 0.5;
discount = 0.1;

Configuration 2 (decrease both at the same scale):

lr = 0.05;
discount = 0.09;

Configuration 3 (increase discount factor):

lr = 0.5;
discount = 5.0;

Configuration 4 (increase learning rate, decrease discount factor):

lr = 0.9;
discount = 0.1;

Configuration 5 (increase learning rate):

lr = 5.0;
discount = 0.9;

Configuration 6 (increase both at the same scale):

lr = 0.5;
discount = 9.0;

Most performed worse than baselines. The small decreases in configurations 1 and 2 resulted in training performances similar to baseline. With a high discount factor or a high learning rate of 5.0, the system could not learn to reach even 1 pipe in 100 deaths.

Below is the link to screen recording to training after setting configuration 1:

https://drive.google.com/file/d/1XYLwSajirVqsjiflCfK-Z727D8Ye6NMG/view?usp=share_link

Training videos for other settings are in the google drive folder link provided on the first page of this submission.

4. Discretization of the state variables *S*. (0.5 pts)
   We discretize space into 500 x 500 entries in the Q table. What is the effect of using a coarser discretization, like 250 or 128 or 64?

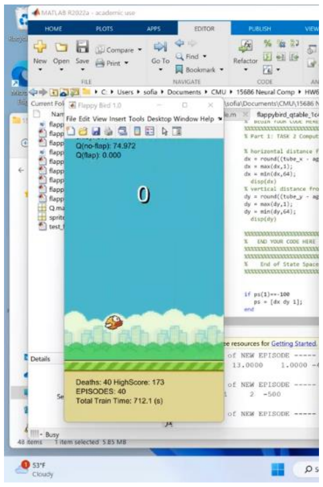
Shown below are the codes to accomplish coarser discretizations.

Q = zeros(250,250,2);

Q = zeros(128,128,2);

Q = zeros(64,64,2);

All performed similar to baselines with the coarsest discretization 64 x 64 resulting in a relatively high number of pipes in around 40 deaths. With this course of a discretization, I also had to put a max on dx to not exceed 64 as the index of the Q table. To the right is a screenshot.



```
% Part 1: TASK 2 Comput
% horizontal distance f
dx = round((tube_x - ag
dx = max(dx,1);
dx = min(dx,64);
  disp(dx)
% vertical distance fro
dy = round((tube_y - ag
dy = max(dy,1);
dy = min(dy,64);
  disp(dy)

%   END YOUR CODE HERE

%   End of State Space

if ps(1)==-100
   ps = [dx dy 1];
end
```

Deaths: 40 HighScore: 173
EPISODES: 40
Total Train Time: 712.1 (s)

```
of NEW EPISODE -----
 13.0000    1.0000 -
of NEW EPISODE -----
     2   -500
of NEW EPISODE -----
```

Below is the link to screen recording to training after using a coarser discretization of 64 by 64:

https://drive.google.com/file/d/1ytKLQyoRb9YB0jBEp7vHhlK2ntoQnADr/view?usp=share_link

5. Exploration v.s. Exploitation. (1 pt)
   One of the oldest questions in RL: should the agent explore more (act not purely based on the best Q value during training) or exploit more during training (act strictly based on the best Q value function during training)? The current implementation is purely exploitation. Would this make it hard to converge since it could depend on the initialization and only form knowledge as it never experiences other options before? In this Task, explore if adding some randomness when choosing the action during training will help with the learning speed.

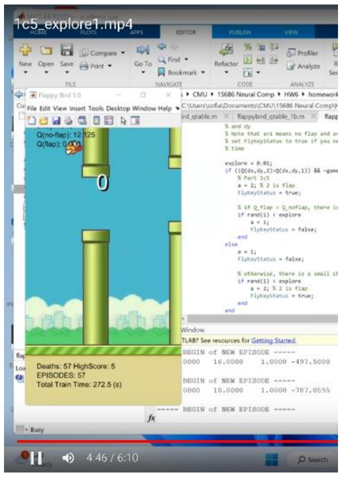
Shown below are the codes to accomplish an exploration scheme where there is a 1% chance that the bird will decide to go against the choice with maximum estimated likelihood.

```
explore = 0.01;
if ((Q(dx,dy,2)>Q(dx,dy,1)) && ~gameover) %%% provide a condition to command bird to flap
   % Part 1c5
   a = 2; % 2 is flap
   FlyKeyStatus = true;

   % if Q_flap > Q_noflap, there is a chance he won't flap
   if rand(1) < explore
      a = 1;
      FlyKeyStatus = false;
   end
else
   a = 1;
   FlyKeyStatus = false;

   % otherwise, there is a small chance he will still flap
   if rand(1) < explore
      a = 2; % 2 is flap
      FlyKeyStatus = true;
   end
end
```

Below is the link to screen recording to training after implementing this function with 1% exploration:

https://drive.google.com/file/d/1gjwHLs1d-Aefg_rRDKj4NY31-1-fNWT-/view?usp=share_link

With 1% exploration, the training performance was worse than baseline with the bird only able to reach 5 pipes in about 18 deaths and then 0 with every death after that. To the right is a screenshot.


1c5_explore1.mp4
HOME PLOTS APPS EDITOR PUBLISH VIEW
New Open Save Compare Print Go To Find Bookmark Refactor Profiler Analyze
FILE NAVIGATE CODE ANALYZE
Flappy Bird 1.0
File Edit View Insert Tools Desktop Window Help
CMU ▸ 15686 Neural Comp ▸ HW6 ▸ homework
C:\Users\sofia\Documents\CMU\15686 Neural Comp\
flappybird_qtable_1b.m
Q(no-flap): 12.125
Q(flap): 0.000
0
4:46 / 6:10

Deaths: 57 HighScore: 5
EPISODES: 57
Total Train Time: 272.5 (s)

```
% and dy
% Note that a=1 means no flap and a=
% set FlyKeyStatus to true if you ne
% time

explore = 0.01;
if ((Q(dx,dy,2)>Q(dx,dy,1)) && ~game
   % Part 1c5
   a = 2; % 2 is flap
   FlyKeyStatus = true;

   % if Q_flap > Q_noflap, there is
   if rand(1) < explore
      a = 1;
      FlyKeyStatus = false;
   end
else
   a = 1;
   FlyKeyStatus = false;

   % otherwise, there is a small ch
   if rand(1) < explore
      a = 2; % 2 is flap
      FlyKeyStatus = true;
   end
end
```

```
Window
TLAB? See resources for Getting Started.
 BEGIN of NEW EPISODE -----
0000    16.0000    1.0000 -497.5000

 BEGIN of NEW EPISODE -----
0000    10.0000    1.0000 -787.0595

----- BEGIN of NEW EPISODE -----
```

Other values of exploration I tried were 0% (to make sure the implementation does not affect baseline), and 2%. Training videos for other these settings are in the google drive folder link provided on the first page of this submission. Higher values were not worth recording because the bird could not learn to reach even 1 pipe with high exploration values.

Part 2: Q-Learning using Neural Network (6 pts)

In general, Q table might be large in dimension and difficult to discretize and compute. Use a neural network to learn the Q table implicitly, and predict the Q given the state input. Train a multi-layer perceptron (MLP) with two input nodes for the state *s*, dx and dy, and two output nodes, *Q(s,flap)* and *Q(s,no-flap)*. We choose the action with higher Q, and then after transition into state s', we compute $y = R + \gamma * \max_a\left(Q(s', a')\right)$ for the new state the bird became and use it as the teaching signal to train the Q-predicting network by minimizing the squared error $\|y - Q(s, a)\|^2$ where $Q(s, a)$ is the Q associated with the state and action of the last time frame.

To facilitate or speed up training, update the neural network while remembering a mini-batch of samples from some time steps (e.g. the last 100; each samples containing a pair of teaching signal *y* and state input *s*) and randomly select a subset of these samples to form a mini-batch to train the network.

2a: **Implementing Q-estimation neural network** (3 pts)

An MLP is provided with two state input notes, two Q output nodes, and one hidden layer. The weights are updated using the memory buffer and mini-batch scheme during learning.

`train_q_network.m` for compute gradients and backpropagation

`mytrain_nn_example.m` is a stand alone script showing gradient descent

A memory buffer of size *K* will store sample pairs (y, s_t). At the beginning, experiences will accumulate until K instances are recorded, and then the neural network will be updated, i.e. begin training. In the training loop, the program will construct mini-batches, each containing *k* samples randomly drawn from the buffer, and train with the mini-batch. After the initial phase, the new experience will be continuously added to the memory buffer (`update_records.m`) and learning will commence when the bird crashes or falls.

Fine-tune the parameters to improve the performance of the game. Make the same recording as Part1 and include the video url inside the report. Compare the learning speed and performance between the Q-table approach and the Q-neural network approach.

1. Task 1: (1 pt) Generalizing the neural networks
   The starter network has 1 hidden layer. Generalize this network by adding hidden layers and/or neurons to improve performance. Record the performance based on two metrics: (1) what is the minimum number of episodes of training needed to go through 10 pipes and 50 pipes? (2) What is the maximum number of pipes your bird can learn to pass within 50 episodes of training. Minimally, your bird should at least be able to pass through 10 pipes within 20 episodes of training. What are the determining factors to learn quickly?

Shown below are the codes to accomplish a more general network by adding one fully connected layer with 30 units and a ReLU layer:

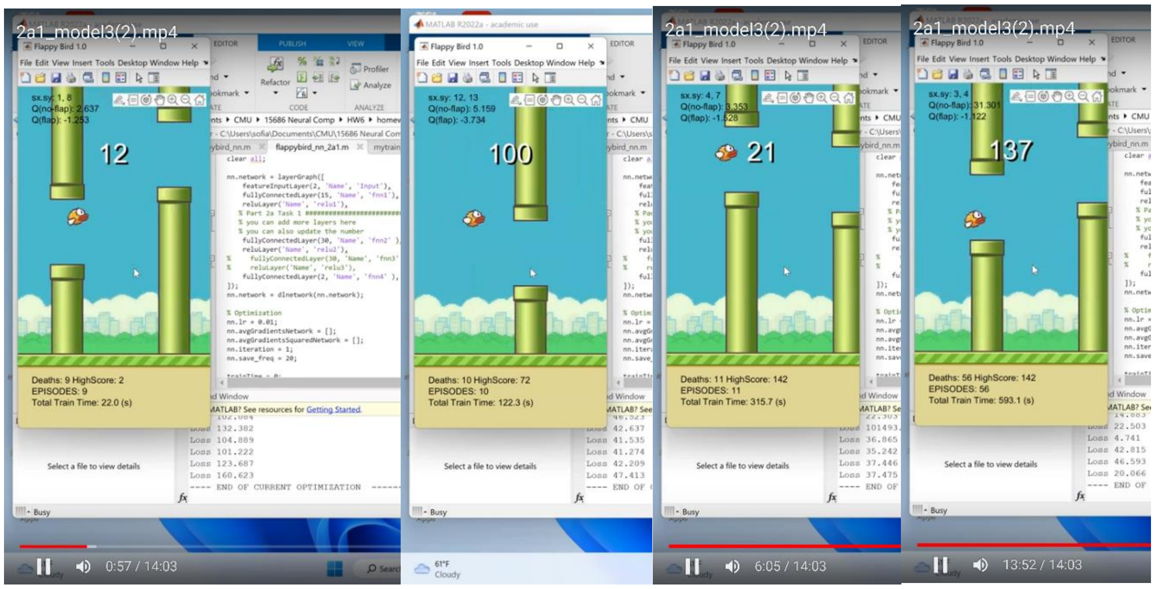
```
nn.network = layerGraph([
    featureInputLayer(2, 'Name', 'Input'),
    fullyConnectedLayer(15, 'Name', 'fnn1'),
    reluLayer('Name', 'relu1'),
    fullyConnectedLayer(30, 'Name', 'fnn2' ),
    reluLayer('Name', 'relu2'),
    fullyConnectedLayer(2, 'Name', 'fnn4' ),
]);
```

Below is the links to screen recordings to two training sessions after implementing this network:

https://drive.google.com/file/d/1XQlZQ1pnbLCrlhnUYV2CGfECnSR5zKqq/view?usp=share_link

https://drive.google.com/file/d/1nGe5gCNYTQc4-S4p0qDbBOBLnbF5H-Zq/view?usp=share_link

This architecture resulted in training performance similar to baseline. In one particularly successful training session, the bird was able to reach 100 pipes in only 10 deaths. Here, 10 is the minimum number of episodes to reach 10 pipes and 50 pipes. In 50 episodes, he can reach over 100 pipes.



sx.sy: 1, 8
Q(no-flap): 2.637
Q(flap): -1.253

Deaths: 9 HighScore: 2
EPISODES: 9
Total Train Time: 22.0 (s)

C:\Users\sofia\Documents\CMU\15686 Neural Com

flappybird_nn_2a1.m

```
clear all;

nn.network = layerGraph([
    featureInputLayer(2, 'Name', 'Input'),
    fullyConnectedLayer(15, 'Name', 'fnn1'),
    reluLayer('Name', 'relu1'),
   % Part 2a Task 1 ##########################
   % you can add more layers here
   % you can also update the number
    fullyConnectedLayer(30, 'Name', 'fnn2' ),
    reluLayer('Name', 'relu2'),
%     fullyConnectedLayer(30, 'Name', 'fnn3'
%     reluLayer('Name', 'relu3'),
    fullyConnectedLayer(2, 'Name', 'fnn4' ),
]);
nn.network = dlnetwork(nn.network);

% Optimization
nn.lr = 0.01;
nn.avgGradientsNetwork = [];
nn.avgGradientsSquaredNetwork = [];
nn.iteration = 1;
nn.save_freq = 20;
```

```
Loss 132.382
Loss 104.889
Loss 101.222
Loss 123.687
Loss 160.623
---- END OF CURRENT OPTIMIZATION ------
```

sx.sy: 12, 13
Q(no-flap): 5.159
Q(flap): -3.734

Deaths: 10 HighScore: 72
EPISODES: 10
Total Train Time: 122.3 (s)

```
Loss 42.637
Loss 41.535
Loss 41.274
Loss 42.209
Loss 47.413
---- END OF
```

sx.sy: 4, 7
Q(no-flap): 3.353
Q(flap): -1.528

Deaths: 11 HighScore: 142
EPISODES: 11
Total Train Time: 315.7 (s)

```
Loss 101493
Loss 36.865
Loss 35.242
Loss 37.446
Loss 37.475
---- END OF
```

sx.sy: 3, 4
Q(no-flap): 31.301
Q(flap): -1.122

Deaths: 56 HighScore: 142
EPISODES: 56
Total Train Time: 593.1 (s)

```
Loss 22.503
Loss 4.741
Loss 42.815
Loss 46.593
Loss 20.066
---- END OF
```

I tried many many other network MLP model architectures including more layers and layers with over 1000 neurons, but all performed similar to this architecture (2-layer with 15 then 30 hidden neurons) or worse. Training videos for some others are in the google drive folder link provided on the first page.

2. Task 2: (2 pts)
   In addition to fine-tuning the hyperparameters on layers and neurons, fine-tune the parameters of memory buffer (how big) and mini-batch (how many samples per batch, how many mini-batches per training episodes) scheme. Describe observations and report optimal buffer and batch sizes. Do they affect the learning speed? Discuss the following: (1) why mini-batching? Set K=k=1. When you increase batch size, what is the effect? Report observations and briefly explain. (2) Why random samples from a larger set of samples to form the mini-batch? Run your

code with mini-batch that does not use random sample to construct the batch. Report observations and explain.

Original Configuration (default):

RL.record_n = 100;
iterations = 20;

Using the architecture implemented in task 1, I tried 7 configurations for memory buffer size (`RL.record_n` in flappybird_nn_2a2.m) and mini-batch (`iterations` in train_q_network_2a2) where configurations 4 and 7:

Configuration 1 (no mini-batching):

RL.record_n = 1;
iterations = 1;

Configuration 2 (decrease both at the different scale):

RL.record_n = 50;
iterations = 4;

Configuration 3 (decrease both by half):

RL.record_n = 50;
iterations = 10;

Configuration 4 (decrease iterations → increase mini-batch):

RL.record_n = 100;
iterations = 10;

Configuration 5 (increase iterations → decrease mini-batch):

RL.record_n = 100;
iterations = 50;

Configuration 6 (increase both by 2X):

RL.record_n = 200;
iterations = 40;

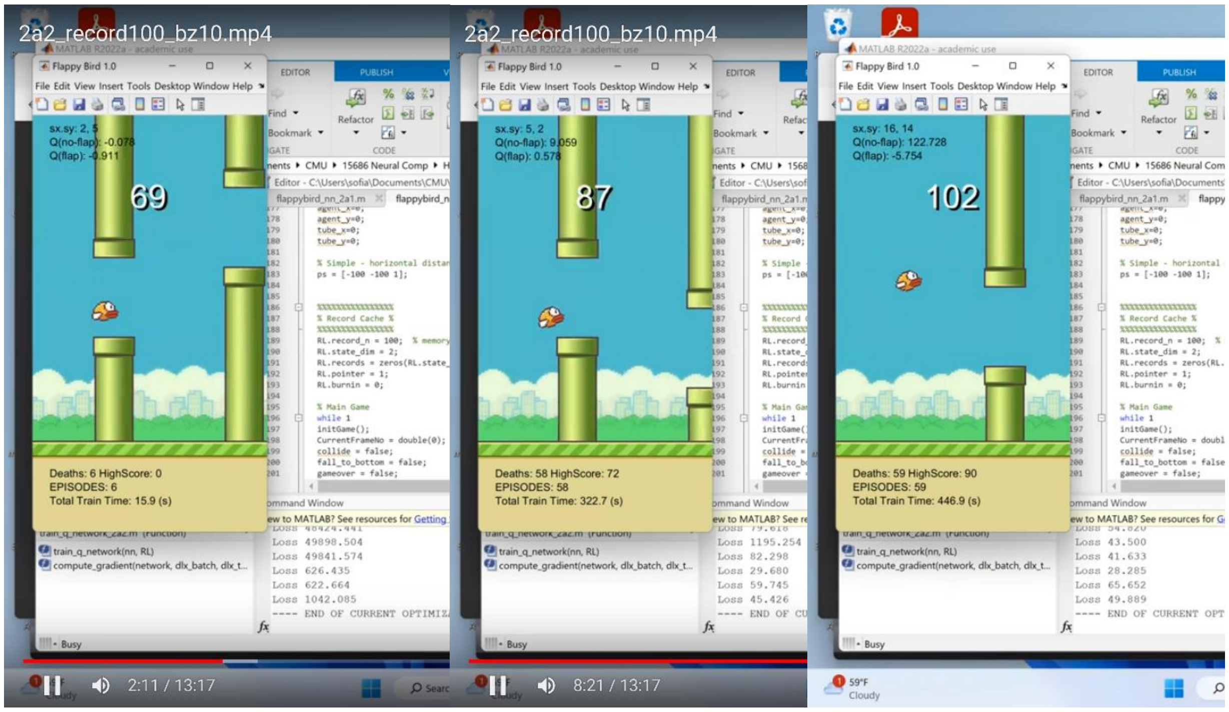
Configuration 7 (increase memory buffer size and decrease iterations → increase mini-batch):

RL.record_n = 1000;
iterations = 10;

Below is the link to screen recording to training after setting configuration 4:

https://drive.google.com/file/d/1RQt629RmoIHumxtCcqo0aXc2_wB9pI55/view?usp=share_link

Configurations where the mini-batch sample size decreased showed worse training performance. Increasing the mini-batch sample size generally showed better training performance. For example, Configuration 4, where the iterations decreased, resulting in a smaller divisor of the memory buffer and subsequently a larger mini-batch sample size, shows training performance similar to baseline where the bird is able to reach 100 pipes in about 50 deaths.



```
sx,sy: 2, 5
Q(no-flap): -0.078
Q(flap): -0.911
69
Deaths: 6 HighScore: 0
EPISODES: 6
Total Train Time: 15.9 (s)
```

```
agent_y=0;
tube_x=0;
tube_y=0;

% Simple - horizontal distan
ps = [-100 -100 1];


%%%%%%%%%%%%%%%%%
% Record Cache %
%%%%%%%%%%%%%%%%%
RL.record_n = 100;  % memory
RL.state_dim = 2;
RL.records = zeros(RL.state_
RL.pointer = 1;
RL.burnin = 0;

% Main Game
while 1
initGame();
CurrentFrameNo = double(0);
collide = false;
fall_to_bottom = false;
gameover = false;
```

```
train_q_network(nn, RL)
compute_gradient(network, dlx_batch, dlx_t...
```

```
Loss 49898.504
Loss 49841.574
Loss 626.435
Loss 622.664
Loss 1042.085
---- END OF CURRENT OPTIMIZ
```

```
sx,sy: 5, 2
Q(no-flap): 9.059
Q(flap): 0.578
87
Deaths: 58 HighScore: 72
EPISODES: 58
Total Train Time: 322.7 (s)
```

```
Loss 1195.254
Loss 82.298
Loss 29.680
Loss 59.745
Loss 45.426
---- END OF CU
```

```
sx,sy: 16, 14
Q(no-flap): 122.728
Q(flap): -5.754
102
Deaths: 59 HighScore: 90
EPISODES: 59
Total Train Time: 446.9 (s)
```

```
Loss 43.500
Loss 41.633
Loss 28.285
Loss 65.652
Loss 49.889
---- END OF CURRENT OPT
```

In Configuration 7, where the size of the memory buffer increases (and therefore the mini-batch sample size also increases), the simulation required many initial failure experiences to fill in the buffer, but after that showed good training performance with the capability also to reach 100 pipes.

These results reveal why mini-batching can improve training performance. The parameters for memory buffer size and mini-batch sampling size effects learning, where a balance is needed to be achieved for the learning problem. A small “memory pool” can impede learning because there is less information when updating the weights. Increasing the batch size affects the size of this “memory pool”.

Configuration 1 effectively removes random sampling and the training performance is abysmal with the bird not able to reach even 1 pipe in 50 deaths.

2b: **Continuous State** (1 pt)

One advantage of using neural network is the ability to work with continuous values; however, there are also disadvantages with using continuous states. Hint: locate dx and dy update in the code (Automatic Control section) and make them continuous. Evaluate the learning and performance of the continuous-state version of the program, relative to the discretized state version. Report observations. Discuss the advantages and disadvantages of using a continuous state model instead of a discretized state model.

Shown below are the codes to implement continuous value states:

dx = tube_x - agent_x;

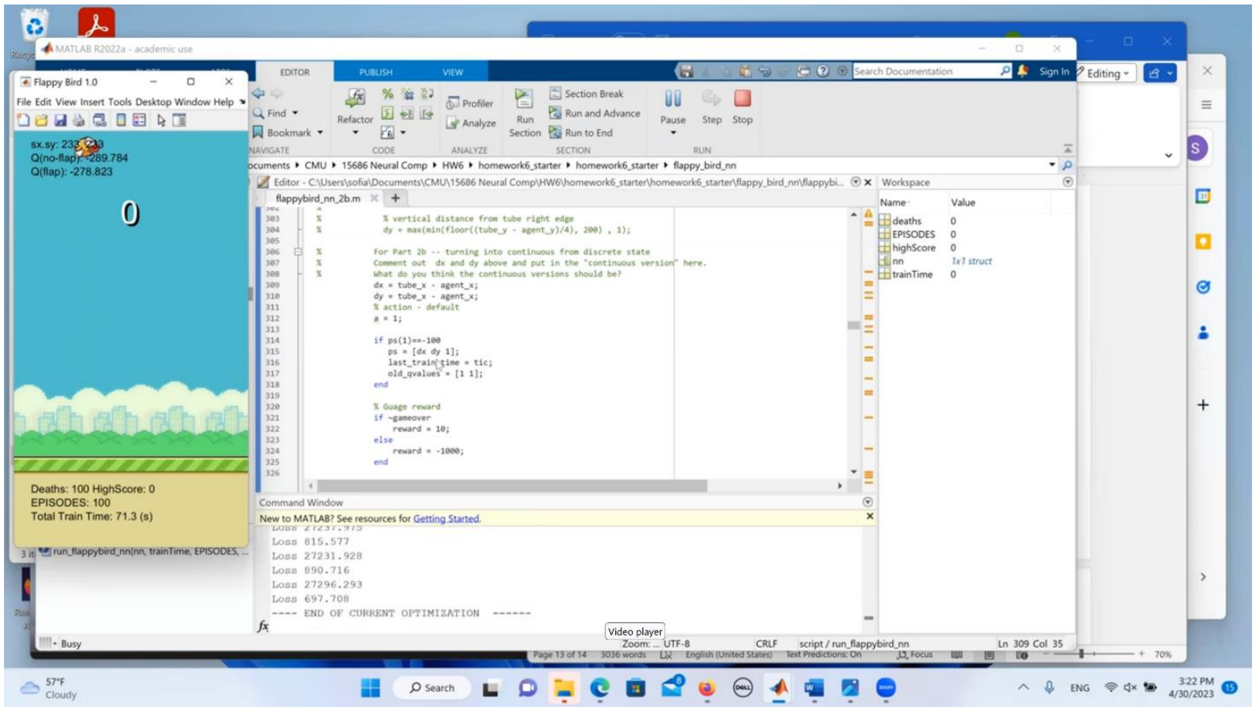
dy = tube_x - agent_x;

Below is the link to screen recording to training after implementing this continuous state version:

https://drive.google.com/file/d/1cScdSqmoY5AevpBuM5BcoqqyMaFI0SAc/view?usp=share_link

These new definitions of dx and dy resulted in worse training performance where Flappy could not reach even 1 pipe in 100 deaths.

Advantages of continuous state models is that you do not lose precise information. The disadvantage is that training requires a lot of data and/or a lot of time to learn action policies.



Flappy Bird 1.0

sx.sy: 233, 233
Q(no-flap): -289.784
Q(flap): -278.823

0

Deaths: 100 HighScore: 0
EPISODES: 100
Total Train Time: 71.3 (s)

flappybird_nn_2b.m

```
%               % vertical distance from tube right edge
%               dy = max(min(floor((tube_y - agent_y)/4), 200) , 1);

%           For Part 2b -- turning into continuous from discrete state
%           Comment out  dx and dy above and put in the "continuous version" here.
%           What do you think the continuous versions should be?
            dx = tube_x - agent_x;
            dy = tube_x - agent_x;
            % action - default
            a = 1;

            if ps(1)==-100
               ps = [dx dy 1];
               last_train_time = tic;
               old_qvalues = [1 1];
            end

            % Guage reward
            if ~gameover
                reward = 10;
            else
                reward = -1000;
            end
```

Command Window

```
Loss 27237.975
Loss 815.577
Loss 27231.928
Loss 890.716
Loss 27296.293
Loss 697.708
---- END OF CURRENT OPTIMIZATION ------
```

2c: **Exploring reward model and other hyper-parameters** (2 pt)

In part 1c, different reward models are experimented with. Provide some observations on the impact of reward models and the hyperparameters. Are there any differences on the impact of the reward models and hyperparameters? Try at least 3 learning rates.

Shown below are the codes to accomplish additional reward as a function of bird position relative to the pipe; specifically, for the bird being below the top pipe, and above the bottom pipe:

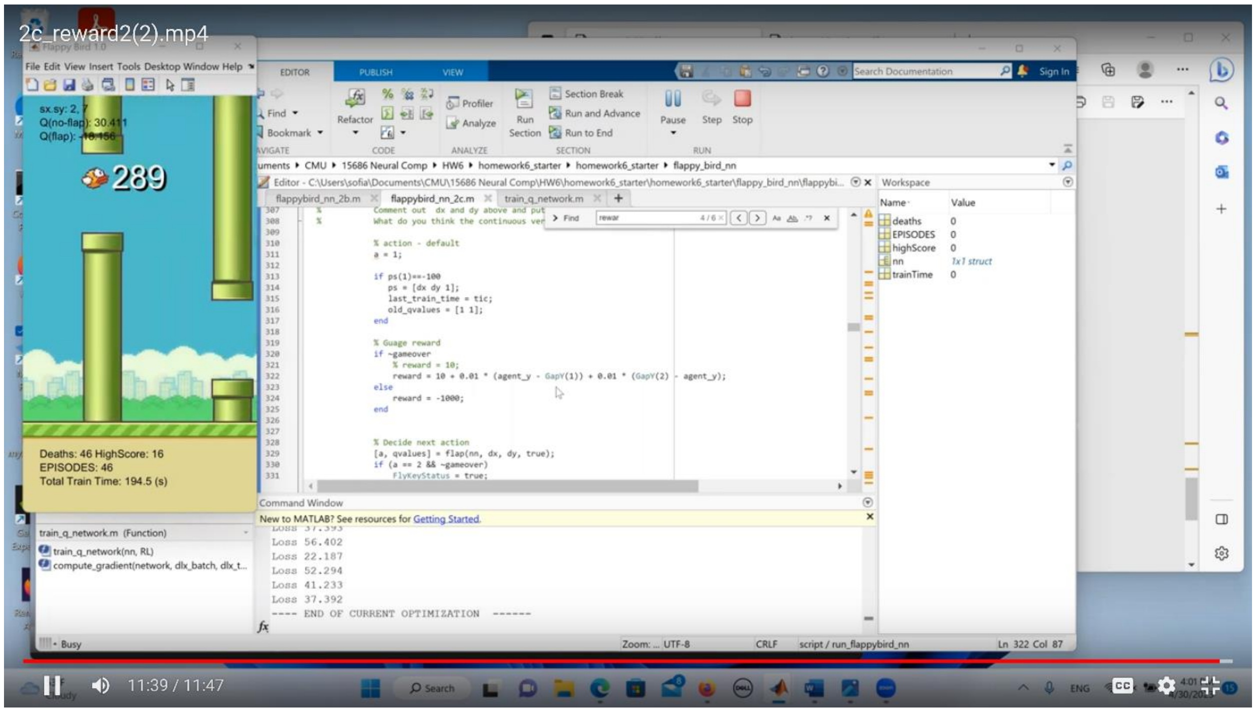
reward = 10 + 0.01 * (agent_y - GapY(1)) + 0.01 * (GapY(2) - agent_y);

This reward scheme is different from that implemented in part 1c1 where here the scale of the reward is proportional to the relative distances to the pipe. Overall the reward will be positive if the bird is in between the pipes, whereas one of the terms will be negative if the bird is not in the correct relative position. The scaling factor is necessary; without it the bird could not reach even 1 pipe in 60 deaths.

Below is a link to screen recording to training after implementing this reward scheme:

https://drive.google.com/file/d/19m_hwvlnq2B3gxVEUEAa4U21EiViZpFK/view?usp=share_link

The reward scheme improved the training performance with the bird consistently able to reach over 200 pipes in under 10 minutes of training.

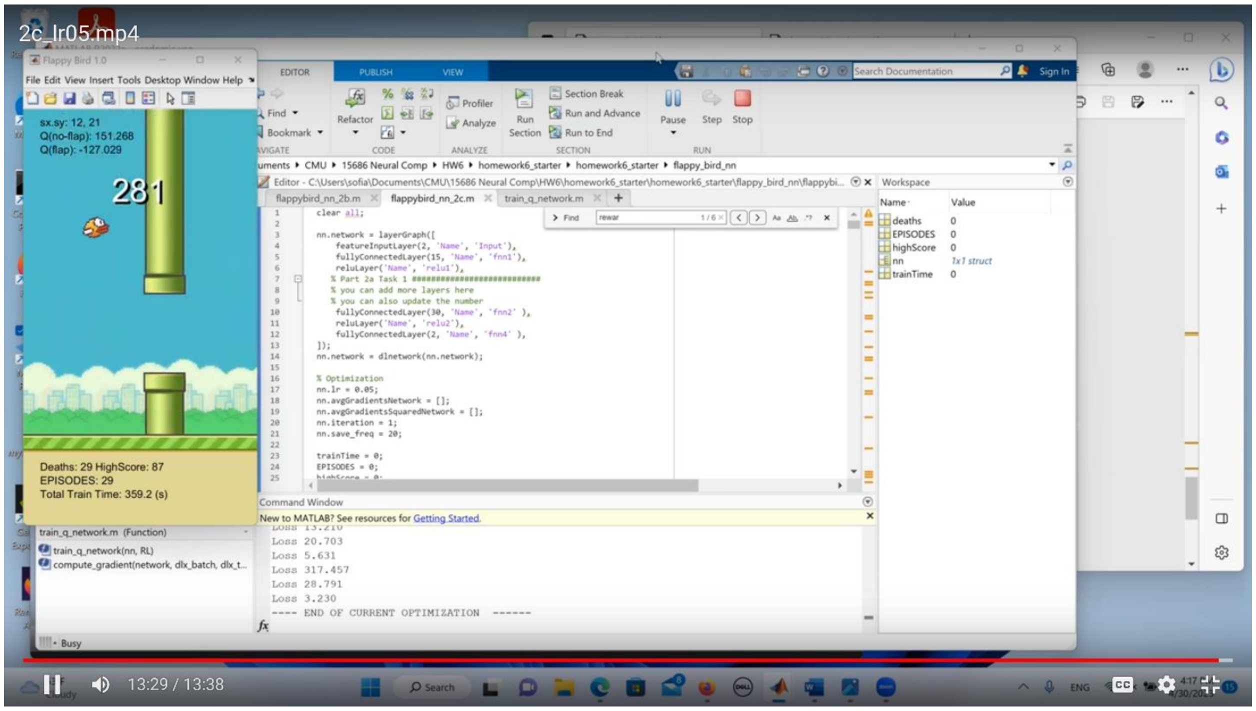


```
Deaths: 46 HighScore: 16
EPISODES: 46
Total Train Time: 194.5 (s)
```

```
%           Comment out  dx and dy above and put
%           What do you think the continuous ver

            % action - default
            a = 1;

            if ps(1)==-100
               ps = [dx dy 1];
               last_train_time = tic;
               old_qvalues = [1 1];
            end

            % Guage reward
            if ~gameover
                % reward = 10;
                reward = 10 + 0.01 * (agent_y - GapY(1)) + 0.01 * (GapY(2) - agent_y);
            else
                reward = -1000;
            end


            % Decide next action
            [a, qvalues] = flap(nn, dx, dy, true);
            if (a == 2 && ~gameover)
                FlyKeyStatus = true;
```

```
Loss 56.402
Loss 22.187
Loss 52.294
Loss 41.233
Loss 37.392
---- END OF CURRENT OPTIMIZATION  ------
```

In addition to this reward scheme, I tried 3 learning rates apart from the original lr=0.1: 0.05, 0.09 and 0.005. Increasing the learning rate to 0.05 allowed Flappy to reach this level of performance in fewer deaths with this reward scheme.



```
sx.sy: 12, 21
Q(no-flap): 151.268
Q(flap): -127.029
```

```
Deaths: 29 HighScore: 87
EPISODES: 29
Total Train Time: 359.2 (s)
```

```
clear all;

nn.network = layerGraph([
    featureInputLayer(2, 'Name', 'Input'),
    fullyConnectedLayer(15, 'Name', 'fnn1'),
    reluLayer('Name', 'relu1'),
   % Part 2a Task 1 ############################
   % you can add more layers here
   % you can also update the number
    fullyConnectedLayer(30, 'Name', 'fnn2' ),
    reluLayer('Name', 'relu2'),
    fullyConnectedLayer(2, 'Name', 'fnn4' ),
]);
nn.network = dlnetwork(nn.network);

% Optimization
nn.lr = 0.05;
nn.avgGradientsNetwork = [];
nn.avgGradientsSquaredNetwork = [];
nn.iteration = 1;
nn.save_freq = 20;

trainTime = 0;
EPISODES = 0;
```

Name	Value
deaths	0
EPISODES	0
highScore	0
nn	1x1 struct
trainTime	0

```
Loss 20.703
Loss 5.631
Loss 317.457
Loss 28.791
Loss 3.230
---- END OF CURRENT OPTIMIZATION ------
```

Below is a link to the screen recording to training after increasing the learning rate (nn.lr) to 0.05:

https://drive.google.com/file/d/17QWFTc_BRX6nn-sMlU_6bnxxZPW_hiyG/view?usp=share_link

A learning rate of 0.05 seems to perform best with this reward scheme as a function of relative bird and tub positions. Too high of a learning rate (lr=0.09) did not improve the time or number of deaths to reach 200 pipes, whereas too low of a learning rate (lr-0.005) greatly impeded the bird's ability to reach even 1 pipe. These results suggest that there is a "sweet spot" for the rate at which the weights are updated.

Decreasing gamma to 0.09 (the discount factor in update_records.m is originally 0.9) did not improve the bird's ability to reach 200 pipes. Below is a link to the screen recording to training after decreasing the discount factor (gamma) by a factor of 10:

https://drive.google.com/file/d/1guG9SX1zgW-tEXqnZ-5L4oq2dLTrkEMf/view?usp=share_link