Homework 3

15-381/681: Artificial Intelligence (Fall 2017)

Out: November 3, 2017
Due: November 15, 2017 at 11:59PM

Homework Policies

- Homework is due on Autolab by the posted deadline. Assignments submitted past the deadline will incur the use of late days.
- You have 6 late days, but cannot use more than 2 late days per homework. No credit will be given for homework submitted more than 2 days after the due date. After your 6 late days have been used you will receive 20% off for each additional day late.
- You can discuss the exercises with your classmates, but you should write up your own solutions. If you find a solution in any source other than the material provided on the course website or the textbook, you must mention the source. All homeworks (programming and theoretical) are always submitted individually.
- Strict honor code with severe punishment for violators. CMUs academic integrity policy can be found here. You may discuss assignments with other students as you work through them, but writeups must be done alone. No downloading / copying of code or other answers is allowed. If you use a string of at least 5 words from some source, you must cite the source.

Submission

For the written portion, please submit your solutions as a pdf to Gradescope.

For the programming portion, please create a tar archive containing `RL.py` and submit it to Autolab. The programming portion will be autograded. When the autograder is released, we may have a submission cap.

1 Part 1: Written [40 Points]

1.1 Neural Networks [20 points]

In this question, you will explore the representation power of neural networks, and how multiple layers can affect it. We will assume the input $x \in \{0,1\}^n$ are binary vectors of length n. We will also use the true binary threshold as the activation function $f(z) = 1$ if $z > 0$ and 0 otherwise. Note that the weights are still allowed to be real numbers. The output will be the result of a single unit and thus be either 0 or 1. We can think of using such a neural network to implement boolean functions.

(a) (4 points) Suppose $n = 2$ i.e. the input is a pair of binary values. Suppose we have a neural network where we don't have any hidden units and just a single output unit i.e. $y = f(w^T x + b)$ is the entire network. What should w, b be if we want to implement boolean AND (i.e. $y = 1$ only when $x = (1, 1)$). What about boolean OR?

(b) (1 point) Under the same conditions as above, what boolean function of two variables cannot be represented? You just need to state one, and provide an explanation.

(c) (4 points) Suppose we now allow a single layer of hidden units i.e. $y = f(w^T z + b), z_j = f(w_j^T x + b_j)$. Construct a neural network that can implement the boolean function you mentioned previously that could not be represented before. The number of hidden units is up to you, but try to keep it as simple as possible.

(d) (8 points) It turns out that for any number of input boolean variables, a single hidden layer is enough to represent any boolean function. Describe a general scheme that one can use to construct such a neural network for any boolean function (HINT: consider conjunctive normal form or disjunctive normal form).

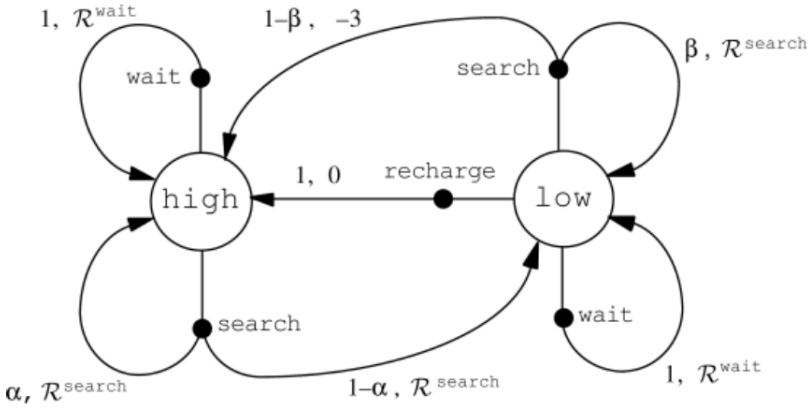
(e) (3 points) If a single layer is enough to represent all boolean functions, why would you ever want to use multiple hidden layers? What does this suggest about designing deep neural network structures in practice?

1.2 Value Iteration and Policy Iteration [20 points]

Suppose we have a recycling robot. At each step, the robot has to decide whether it should either:

- Search for a can
- Wait for someone to bring a can
- Go to home base and recharge

Searching is better but runs down the battery; if the robot runs out of power while searching, it has to be rescued. The two possible states that the robot can be in are just the battery levels, high or low. The reward is the number of cans collected. The diagram below describes the MDP, where each action is annotated with (probability, reward). For this problem, suppose we have the parameters $\mathcal{R}^{\text{wait}} = 1$, $\mathcal{R}^{\text{search}} = 3$, $\beta = .5$, $\alpha = .5$, and discount factor $\gamma = .9$.



(a) (10 points) Perform two iterations of value iteration (i.e. two Bellman updates for each state). Show your work and the resulting values after each iteration.

(b) (10 points) Suppose we have an initial policy that will always wait. Perform one iteration of policy iteration. What is the new policy? Show your work (Note: in the policy evalution step, calculate the value explicitly).

2 Part 2: Flappy Bird RL: Programming [60 Points]

2.1 The Problem [40 Points]

In this part of the assignment, you will be implementing deep reinforcement learning that uses a Deep Q-Network to learn to play the game Flappy Bird. You will use a neural network implemented by `keras` and `tensorflow` to approximate the Q function, the expected utility of an action at each state.

More specifically, at each time step, you should estimate the $Q(s, a)$ value using your Neural Net. Your neural network should have 2 output nodes, one representing the value for flap and another for not flap. You should take action based on which one is greater and the bird moves to the next state s'.

The reward of being in the next state s' is given by $y = R + \gamma * max(Q(s', a'))$, where R is the immediate reward being in state s', calculated according to some rules, γ is the discount factor, and $max(Q(s', a'))$ is the best expected long-term reward over all possible actions (in this case, flap or no-flap in state s') estimated based on the current Neural Net.

To train the Neural Net to learn the Q value for each example, we set the input to be state s, and compute $Q(s, \text{flap})$ and $Q(s, \text{no-flap})$ using the neural network. We choose the action (flap or no-flap) with the higher Q, and then we will compute $y = R + \gamma * max(Q(s', a'))$ for the state the bird will end up to provide a "teaching signal" to train the neural network by minimizing the squared error $\|y - Q(s, a)\|^2$.

However, instead of updating the neural network at each step, you should first gather a mini-batch of samples (states s and predicted reward y), and then draw random samples to perform each round of training. For example, 400 pairs of y and s from the last 400 time steps could be recorded, and then 50 of them will be randomly selected to form a mini-batch for training.

The slides at the end of the 11/2 lecture should provide some more helpful notes, as well as some comments on the batching process that were not covered during class.

2.2 Installation

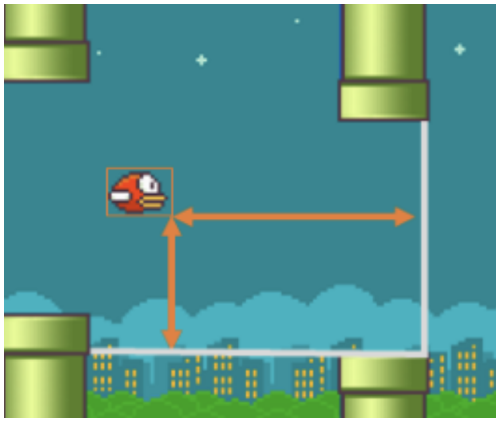
In order to visualize how your code actually performs, you will need to install `pygame, keras,` and `tensorflow`. The version we are using for `pygame` is after 1.9.X, and `keras` and `tensorflow` are the latest version, though it may not make too much of a difference if your version isn't exactly right. You should be able to install all three with `pip`, though if you run into problems you may have to check out the installation instructions in more detail at http://www.pygame.org/, https://www.tensorflow.org/, and https://keras.io/.

To run the flappy bird simulation, run `python flappy.py`.

2.3 Deep Reinforcement Learning

2.3.1 State Space

Each state is represented by the horizontal and vertical distance between the bird and the next pipe. These will be given as parameters in the `flap` function as `distX` (horizontal arrow) and `distY` (vertical arrow).



2.3.2 Minibatching

We have provided one suggestion for mini-batching for this task. You will need to choose a batch size of K to keep track of. At each time stamp, instead of back-propagating one instance a time, you will be fitting your Neural Net with randomly sampled $k \leq K$ instances. Your Neural Net should not be updated until the first K instances are recorded.

2.3.3 Reward Model

There are many possible reward models you can implement. You are access a number of perception measurements: `distX` (horizontal arrow), `distY` (vertical arrow), and the scores. You can use these measurements to derive any other measurements you need and assign reward R to each measurement.

A naive approach would be to add +10 when the bird is alive and within some height relative to the incoming pipes, -10 if the bird is alive but outside some height relative to the incoming pipes, and -10000 when the bird crashes. To calculate the relative heights, you may use the provided constants `PIPEGAPSIZE` and `BIRDHEIGHT`. You might want to reconsider your implementation if the bird cannot reach 20 score after 15 min of training.

2.4 Neural Network

We will provide you a working Neural Network in your starter code. We guarantee that the parameters set for the Neural Net can solve the game, but feel free to tweak it by yourself.

To get the estimated Q value, run `self.model.predict(state, batch_size=1)`, where `state` is the an element of the state space described in the previous section.

To back-propagate, run `self.model.fit(old_states, updates, batch_size=b, epochs=1)`, where `old_states` is a list of states predicted in the past, `updates` is a list of best Q values corresponding to each prediction in `old_state`, `batch_size` is a constant that you choose. Note that you should have `len(old_states) = len(updates) = batch_size`.

2.5 Grading

Your code will be autograded for a score out of 40 points. Your code will be trained in the Autolab for 10 minutes and the maximum score your algorithm can reach will be marked as your score.

The autolab grading system will be as follows:

- >= 2 points on Flappy Bird → 10 autolab points
- >= 5 points on Flappy Bird → 20 autolab points
- >= 15 points on Flappy Bird → 30 autolab points
- >= 40 points on Flappy Bird → 40 autolab points

2.6 Bonus and Competition [10 Bonus Points]

5 bonus points will be awarded to submissions that get >= 100 points on Flappy Bird.

In addition to these bonus points, up to another 5 points will be given as part of a competition. We plan to award some percentage of the top students with up to 5 bonus points. You can change any parameter you want in the entire implementation.

Throughout this entire assignment, parameters that you can vary include but are not limited to the following:

1. Number of hidden layers
2. Number of hidden units
3. Learning rate
4. Mini-batch size K
5. Sub-batch size k
6. Discount factor
7. Activation function
8. Loss function
9. Reward model

We will use your autograded score as your competition score.

2.7 Report [20 Points]

1. Why mini-batching and why random samples from a larger set of samples to form the mini-batch? Run your code without the implementation of mini-batching, i.e. set $K = k = 1$. Report what you observe and briefly explain why. You can also run your code with a mini-batch that does not use random sample to construct the batch. Report what you observe and briefly explain why.
2. Suppose your reward model cannot give any hint on the perception, i.e. you can only build your reward model based on `crashed, scored`. Do you think the algorithm can solve the game? What could be the problem? Briefly explain why.
3. What is the difference between using ReLU and sigmoid as the activation function? How does it apply to this problem?
4. Choose five different learning rates, run your algorithm and record the time for the bird to score 50 for each learning rate (if your solution cannot get to 50, that is OK, use some other number). Plot the graph, list dependent and independent variables, and briefly explain your observations.