HOMEWORK 6
NEURAL NETWORKS AND REINFORCEMENT LEARNING[1]

10-301 / 10-601 INTRODUCTION TO MACHINE LEARNING (SPRING 2021)
http://mlcourse.org

OUT: Mar. 30, 2021
DUE: Apr. 7, 2021 11:59 PM
TAs: Clay Yoo, Everett Knag, Varun Natu

Summary In this assignment, you will implement a reinforcement learning algorithm for solving the classic mountain-car environment. As a warmup, the first section is as a review of neural network and regularisation concepts. Later sections will then lead you through an on-paper example of how non-deterministic value iteration and Q-learning work. Then, in Section 2, you will implement Q-learning with function approximation to solve the mountain car environment.

START HERE: Instructions

- **Collaboration Policy**: Please read the collaboration policy here: https://www.cs.cmu.edu/~10601
- **Late Submission Policy:** See the late submission policy here: https://www.cs.cmu.edu/~10601
- **Submitting your work:** You will use Gradescope to submit answers to all questions and code. Please follow instructions at the end of this PDF to correctly submit all your code to Gradescope.
  - **Written:** For written problems such as short answer, multiple choice, derivations, proofs, or plots, we will be using Gradescope (https://gradescope.com/). Please use the provided template. Submissions must be written in LaTeX. Regrade requests can be made, however this gives the staff the opportunity to regrade your entire paper, meaning if additional mistakes are found then points will be deducted. Each derivation/proof should be completed in the boxes provided. If you do not follow the t your assignment may not be graded correctly by our AI assisted grader.
  - **Programming:** You will submit your code for programming questions on the homework to Gradescope (https://gradescope.com). After uploading your code, our grading scripts will autograde your assignment by running your program on a virtual machine (VM). When you are developing, check that the version number of the programming language environment (e.g. Python 3.6.9, OpenJDK 11.0.5, g++ 7.4.0) and versions of permitted libraries (e.g. `numpy` 1.17.0 and `scipy` 1.4.1) match those used on Gradescope. You have 30 Gradescope programming submissions. We recommend debugging your implementation on your local machine (or the Linux servers) and making sure your code is running correctly first before submitting you code to Gradescope.
- **Materials:** The data that you will need in order to complete this assignment is posted along with the writeup and template on Piazza.

[1]Compiled on Sunday 13th June, 2021 at 23:33

Linear Algebra Libraries When implementing machine learning algorithms, it is often convenient to have a linear algebra library at your disposal. In this assignment, Java users may use EJML[a] or ND4J[b] and C++ users Eigen[c]. Details below. (As usual, Python users have NumPy.)

EJML for Java EJML is a pure Java linear algebra package with three interfaces. We strongly recommend using the SimpleMatrix interface. The autograder will use EJML version 0.38. When compiling and running your code, we will add the additional command line argument `-cp "linalg_lib/ejml-v0.38-libs/*:linalg_lib/nd4j-v1.0.0-beta7-libs/*:./"` to ensure that all the EJML jars are on the classpath as well as your code.

ND4J for Java ND4J is a library for multidimensional tensors with an interface akin to Python's NumPy. The autograder will use ND4J version 1.0.0-beta7. When compiling and running your code, we will add the additional command line argument `-cp "linalg_lib/ejml-v0.38-libs/*:linalg_lib/nd4j-v1.0.0-beta7-libs/*:./"` to ensure that all the ND4J jars are on the classpath as well as your code.

Eigen for C++ Eigen is a header-only library, so there is no linking to worry about—just `#include` whatever components you need. The autograder will use Eigen version 3.3.7. The command line arguments above demonstrate how we will call you code. When compiling your code we will include, the argument `-I./linalg_lib` in order to include the `linalg_lib/Eigen` subdirectory, which contains all the headers.

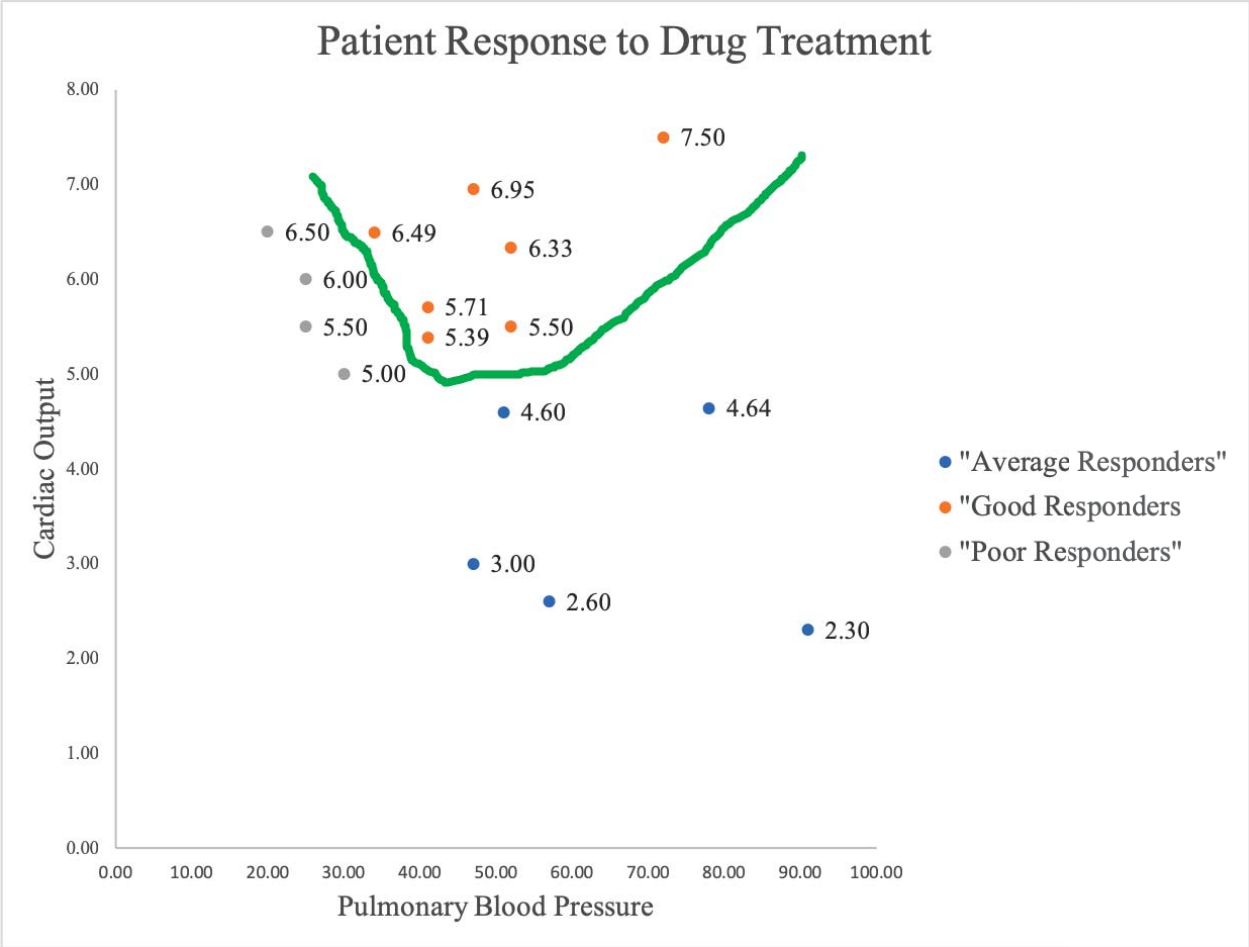
We have included the correct versions of EJML/ND4J/Eigen in the `linalg_lib.zip` posted on the Piazza Resources page for your convenience. It contains the same `linalg_lib/` directory that we will include in the current working directory when running your tests. Do **not** include EJML, ND4J, or Eigen in your homework submission; the autograder will ensure that they are in place.

[a] https://ejml.org
[b] https://deeplearning4j.org/docs/latest/nd4j-overview
[c] http://eigen.tuxfamily.org/

1 Written Questions [51 Points]

1.1 Neural Networks, Logistic Regression, Regularization Revisited

Imagine you work for a pharmaceutical company that is trying to predict whether certain patients will respond well to a new drug. Specifically, these patients have high blood pressure in their lungs, a condition known as pulmonary hypertension. Doctors recommend that the best predictors of a treatment effect is the patient's heart function (measured by cardiac output) and the blood pressure in their lungs (pulmonary blood pressure). You plot the data and visualize the following:



1. (2 points) Draw on the graph above the decision boundaries of a trained neural network that minimizes the training error when classifying Good Responders vs all others (Poor or Average). Assume the neural network has two hidden units and one hidden layer. What is the smallest training error you can achieve?

   Fill in the blank (write answer as a fraction):

   Your Answer

   0

2. (2 points) Using your decision boundaries above, assuming a logistic activation function, which point has the highest probability of being a Good Responder? Provide the point number as shown on the graph (poitnts are labeled by their y value).

   Fill in the blank:

   Your Answer

   7.5

3. (1 point) True or False: Increasing the number of hidden units of a neural network will always guarantee a lower training error.

   √ **True**

   ◯ False

4. (1 point) Convolutional neural networks often consist of convolutional layers, max-pooling layers, and fully-connected layers. Select all the layer types below that have parameters (i.e. weights) which can be learned by gradient descent / backpropagation. **Select all that apply:**

   ■ **convolutional layer**

   □ max-pooling layer

   ■ **fully-connected layer**

5. (2 points) Consider the black-and-white 5 pixel by 5 pixel image shown below. Black pixels are represented by the value 0 and white pixels by 1.

1	1	0	0	0
1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	0	1	1

   Next consider the 3 by 3 convolution with weights shown below.

-1	0	1
-1	0	1
-1	0	1

   Suppose we apply the above convolution (as one would in a CNN convolutional layer) to the above image to produce a new output image. Assume that we do not permit any padding to be used and the stride of the convolution is 1. What is the value of the pixel in the upper-left corner of the output image?

   (Important Note: Convolution is sometimes defined differently in machine learning than in other fields, such as signal processing. So be sure to follow the method of convolution shown in the lectures on CNNs.) **Fill in the blank:**

Your Answer

0

6. (2 points) For the same output image produced in the previous question, what is the value of the pixel in the upper-right corner of the output image?

   Fill in the blank:

Your Answer

-2

7. (1 point) True or False: Long Short Term Memory (LSTM) networks partially address the vanishing gradient problem by incorporating input, output, and forget gates.

   √ **True**

   ◯ False

8. (1 point) True or False: In a recurrent neural network (RNN), the weight vector passed to a hidden unit at the same hidden layer level from a prior unit will have different values.

   ◯ True

   √ **False**

9. (1 point) True or False: Recurrent neural networks (RNNs) can accept sequence data as input, but can only output a single classification decision for that input sequence.

   ◯ True

   √ **False**

10. (2 points) Regularization.

    Which of the following are true about regularization? **Select all that apply:**

    ■ **One of the goals of regularization is combating overfitting.**

    □ A model with regularization fits the training data better than a model without regularization

    ■ **The L-0 norm (number of non-zero parameters) is rarely used in practice in part because it is non-differentiable.**

    ■ **One way to understand regularization is that it attempts to follow Occam's razor and make the learning algorithm prefer "simpler" solutions.**

11. (2 points) Regularization in Linear Regression.

    When performing linear regression, which of the following options will decrease mean-squared training error? **Select all that apply:**

    ■ **Adding higher-order functions of the features as separate features**

- ☐ Increasing the regularization weight
- ☐ For the same weight on the regularizer, using an L1 regularizer instead of an L2
- ■ **For the same weight on the regularizer, using an L1 regularizer instead of an L0**
- ☐ None of the above

1.2 Non-Deterministic Value Iteration

In this question we will explore value iteration with a non-deterministic transition function. Suppose the agent is in a grid world. In any state, there is the following **non-deterministic** transition function: with probability 80% the agent transitions to the intended state. With probability 10% the agent slips left of the intended direction. With probability 10% the agent slips right of the intended direction. If the agent hits the edge of the board, it remains in the same state.

For example, If the agent were in state B and choose action down, there is an 80% chance of moving to state E, a 10% chance of moving to state C, and a 10% chance of moving to state A.

If at any point, the agent transitions into a state labeled P, it is given a reward of -100 and terminates. Similarly, if it transitions to a state G, it is given a reward of +1 and terminates. The agent is never initialized in state P or G. All other rewards are 0 and $\gamma = 1$.

P	P	P	P
A	B	C	G
D	E	F	G
P	P	P	P

Table 1: Depiction of Grid World

Notice that in this problem, the agent's immediate reward R, is a function of its current state s, its action a, and its next state s' (which could be different than the intended next state). Therefore, during value iteration, we need to account for the agent's expected reward given a state-action pair, and use a slightly different formula than the one we saw in lecture: $\forall s \in \mathcal{S}$

$$v_0(s) = 0$$
$$v_{k+1}(s) = \max_a \sum_{s' \in \mathcal{S}} p(s'|s,a) \left[R(s,a,s') + \gamma V_k(s') \right]$$

1. (1 point) How many possible deterministic policies are there in this environment, including both optimal and non-optimal policies?

   Answer

   4^6

2. (1 point) After initializing the board to all zeros, in state B, which of the following actions are optimal.

   Select all that apply:

   ☐ Up

   Down

   ☐ Left

   ☐ Right

3. (1 point) What is the value of state B in the next round (round 1) of synchronous value iteration? Round your answer to three decimal places.

Answer

0

4. (1 point) What is the value of state C in round 1 of synchronous value iteration? Round your answer to three decimal places.

Answer

0.1

5. (1 point) What is the value of state C in round 3 of synchronous value iteration? Round your answer to three decimal places.

Answer

0.245

6. (1 point) What is the final value of state A once value iteration converges.

Answer

1

7. (2 points) What is the optimal action for each state.

P	P	P	P
1	1	1	G
1	1	1	G
P	P	P	P

Table 2: Place the corresponding action in each blank cell

1.3 Q-Learning Multiple Choice

1. (1 point) True or False: Given the value function $V^*(s)$, we can derive the optimal action-value function $Q^*(s, a)$

   Select one choice:

   ☐ True

   ☒ False

2. (1 point) True or False: Q-Learning cannot learn from interacting with the environment because it updates its estimates based on its own previously learned estimates

   Select one choice:

   ☐ True

   ☒ False

3. (2 points) Which of the following is true about Q-Learning?

   Select all that apply:

   ☐ Q-Learning creates an explicit model of the environment and state-transition function to learn state-value estimates

   ☐ Q-Learning must wait for an episode to end before it can update its estimates.

   ☒ Q-Learning can update its estimates after the agent takes a single action

   ☒ Q-Learning can learn without being given an explicit reward function or state-transition function.

   ☐ None of the above.

4. (2 points) Which of the following is true about the convergence properties of Q-Learning?

   Select all that apply:

   ☐ Q-Learning will converge to different estimates of the action-value function $Q(s, a)$ for ϵ-greedy policies with different non-zero ϵ values.

   ☐ Q-Learning necessarily converges to the optimal action-value function $Q^*(s, a)$ for all policies

   ☒ Q-Learning converges to the optimal action-value function $Q^*(s, a)$ under the assumption that all state-action pairs are visited infinitely often by the policy.

   ☐ Q-Learning guarantees convergence only for deterministic environments.

   ☐ All of the above.

5. (2 points) Which of the following is the correct expression for the action-value function Q upon taking action a_t in state s_t in terms of the discount factor γ, Reward function $R(s, a)$, State transition function $T(s'|s, a)$ and the value function $V(s)$?

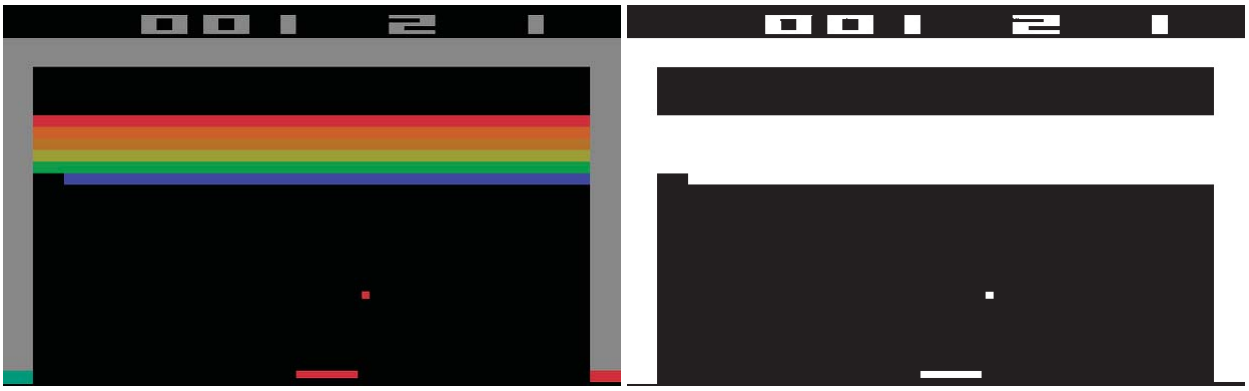
Select one choice:

- ☒ $Q(s_t, a_t) = R(s_t, a_t) + \gamma * [\sum_{s_{t+1}} T(s_{t+1}|s_t, a_t) * V(s_{t+1})]$
- ☐ $Q(s_t, a_t) = R(s_t, a_t) + \gamma * [\frac{1}{|S|} \sum_{s_{t+1}} Q(s_{t+1}, a_t)]$,where $|S|$ is the total number of states
- ☐ $Q(s_t, a_t) = R(s_t, a_t) + \gamma * [\frac{1}{|S|} \sum_{s_{t+1}} V(s_{t+1})]$, where $|S|$ is the total number of states
- ☐ The given quantities and are not enough to express Q

1.4 Function Approximation

In this question we will motivate function approximation for solving Markov Decision Processes by looking at Breakout, a game on the Atari 2600. The Atari 2600 is a gaming system released in the 1980s, but nevertheless is a popular target for reinforcement learning papers and benchmarks. The Atari 2600 has a resolution of 160×192 pixels. In the case of Breakout, we try to move the paddle to hit the ball in order to break as many tiles above as possible. We have the following actions:

- Move the paddle left
- Move the paddle right
- Do nothing



(a) Atari Breakout (b) Black and white Breakout

Figure 1: Atari Breakout. 1a is what Breakout looks like. We have the paddle in the bottom of the screen aiming to hit the ball in order to break the tiles at the top of the screen. 1b is our transformation of Atari Breakout into black and white pixels for the purpose of some of the following problems.

1. (1 point) Suppose we are dealing with the black and white version of Breakout[2] as in Figure 1b. Furthermore, suppose we are representing the state of the game as just a vector of pixel values without considering if a certain pixel is always black or white. Since we are dealing with the black and white version of the game, these pixel values can either be 0 or 1.

   What is the size of the state space?

   Answer

   $2^{160 \times 192}$

2. (1 point) In the same setting as the previous part, suppose we wish to apply Q-learning to this problem. What is the size of the Q-value table we will need?

   Answer

   $2^{160 \times 192} \times 3$

[2] Play a Google-Doodle version here

3. (1 point) Now assume we are dealing with the colored version of Breakout as in Figure 1a. Now each pixel is a tuple of real valued numbers between 0 and 1. For example, black is represented as $(0, 0, 0)$ and white is $(1, 1, 1)$.

   What is the size of the state space and Q-value table we will need?

   Answer

   $\inf$ - It is intractable as state space in continuous.

   By now you should see that we will need a huge table in order to apply Q-learning (and similarly value iteration and policy iteration) to Breakout given this state representation. This table would not even fit in the memory of any reasonable computer! Now this choice of state representation is particularly naïve. If we choose a better state representation, we could drastically reduce the table size needed.

   On the other hand, perhaps we don't want to spend our days feature engineering a state representation for Breakout. Instead we can apply function approximation to our reinforcement algorithms! The whole idea of function approximation is that states nearby to the state of interest should have *similar* values. That is, we should be able to generalize the value of a state to nearby and unseen states.

   Let us define $q_\pi(s, a)$ as the true action value function of the current policy π. Assume $q_\pi(s, a)$ is given to us by some oracle. Also define $q(s, a; \mathbf{w})$ as the action value predicted by the function approximator parameterized by $\mathbf{w}$. Here $\mathbf{w}$ is a matrix of size $\dim(S) \times |\mathcal{A}|$, where $\dim(S)$ denotes the dimension of the state space. Clearly we want to have $q(s, a; \mathbf{w})$ be close to $q_\pi(s, a)$ for all (s, a) pairs we see. This is just our standard regression setting. That is, our objective function is just the Mean Squared Error:

$$J(\mathbf{w}) = \frac{1}{2}\frac{1}{N} \sum_{s \in \mathcal{S}, a \in \mathcal{A}} \left(q_\pi(s, a) - q(s, a; \mathbf{w})\right)^2 \tag{1}$$

   Because we want to update for each example stochastically[3], we get the following update rule:

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \left(q(s, a; \mathbf{w}) - q_\pi(s, a)\right) \nabla_{\mathbf{w}} q(s, a; \mathbf{w}) \tag{2}$$

   However, more often then not[4] we will not have access to the oracle that gives us our target $q_\pi(s, a)$. So how do we get the target to regress $q(s, a; \mathbf{w})$ on? One way is to bootstrap[5] an estimate of the action value under a greedy policy using the function approximator itself. That is to say

$$q_\pi(s, a) \approx r + \gamma \max_{a'} q(s', a'; \mathbf{w}) \tag{3}$$

   Where r is the reward observed from taking action a at state s, γ is the discount factor and s' is the state resulting from taking action a at state s. This target is often called the Temporal Difference (TD) target, and gives rise to the following update for the parameters of our function approximator in lieu of a tabular update:

[3] This isn't really stochastic, you'll be asked in a bit why.
[4] Always in real life.
[5] Metaphorically, the agent is pulling itself up by its own bootstraps.

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \underbrace{\Big(q(s,a;\mathbf{w}) - \underbrace{\big(r + \gamma \max_{a'} q(s',a';\mathbf{w})\big)}_{\text{TD Target}} \Big)}_{\text{TD Error}} \nabla_{\mathbf{w}} q(s,a;\mathbf{w}) \tag{4}$$

4. (2 points) Let us consider the setting where we can represent our state by some vector $\mathbf{s}$, action $a \in \{0,1,2\}$ and we choose a linear approximator. That is:

$$q(\mathbf{s},a;\mathbf{w}) = \mathbf{s}^T \mathbf{w}_a \tag{5}$$

Again, assume we are in the black and white setting of Breakout as in Figure 1b. Show that tabular Q-learning is just a special case of Q-learning with a linear function approximator by describing a construction of $\mathbf{s}$. (**Hint**: Engineer features such that 5 encodes a table lookup)

Answer

When we use a one hot encoding for representing each state-action pair, the linear function approximation reduces to the tabular case.

5. (3 points) Stochastic Gradient Descent works because we can assume that the samples we receive are independent and identically distributed. Is that the case here? If not, why and what are some ways you think you could combat this issue?

Answer

No, the samples generated by interaction are not i.i.d as they depend on the agent's policy. Including a strong exploration mechanism and using a replay buffer are some workarounds.