EECS 492 Homework 3

Due Friday, November 4

Fall 2022

Introduction

Deadline

This homework is due by 11:59 PM EST on Friday, November 4. Late homework will be penalized by 10% per day (where each day starts at 12:00 AM on the due day). Homework turned in after three days will not be accepted.

Gradescope

Please familiarize yourself with Gradescope if you have never used it before. In particular, when you submit an assignment, Gradescope asks you to select pages of your solution that correspond to each problem. This is to make it easy for graders to find where in your work they should grade.
For the written questions, put your answers in a PDF and submit to the corresponding assignment on Gradescope. You can submit this PDF as many times as you would like before the deadline. You can use the provided LaTeX template or Word template and fill in your answer. You may also submit your written response homework in a separate document.

For the coding questions, submit x_train submission.npz, y_train submission.npz, dnn model.h5, dnn_improved_model.h5, and your Google Colab notebook (as a .ipynb) to the Gradescope autograder. A template starter code is provided for you. Do not modify any of the imports in these files. You will be graded on both visible and hidden test cases. Thus, you should make sure your solution is always correct (possibly by testing it on other cases that you write) and not just whether it succeeds on the visible test cases. This is an important skill for a practicing computer scientist! You can submit to the autograder as many times as you would like before the deadline. While you will not be graded on style, we encourage you to uphold best-practice style guidelines. For more information about the coding refer to the coding portion of the homework.

Setup

For more information on how to use Google Colab and instructions, refer to the coding portion of the homework. Feel free to visit us in office hours if you need help on the above or help on setting up tools for debugging.

Written Section [50 points]

1 Intro to Learning [10 points]

1. Provide a brief answer (1-2 sentences) to the following questions: [10 points]

   (a) What is a disadvantage of initializing all weights in a deep neural network to 0? [2 points]

   Solution: Each weight would get updated to the exact same weight, so during training the nodes would just learn the same features.

   (b) State one advantage of rectified linear unit (ReLu) activation compared to logistic sigmoid activation. For more information on both of these activation functions, click here [2 points]

   Solution: As ReLU just needs to calculate max(0,x) while sigmoid needs to calculate exponential operations, ReLU is more computationally efficient.

   (c) List 1 advantage and 1 disadvantage of using decision trees for classification. [2 points]

Solution: Advantage: interpretability (intuitive and easy to explain) Disadvantage: Because of greedy search, they may not be optimal.

(d) Why do we use regularization in machine learning? [2 points]

Solution: It penalizes complex models, making it less likely to overfit.

(e) What is a kernel function and why is it useful? [2 points]

Solution: Kernel functions are used in SVM to help project the data to a different dimension in order to find better classifiers. The functions are then parameters in SVM functions. It's helpful because data that is not linearly separable in its original dimension is often linearly separable, or at least more easily separable in higher dimensions.

2 Cross Validation and KNN [15 points]

1. Given the dataset below, we will use k-nearest neighbors to predict the label "Red" or "Blue" for each datapoint. Note that to find the nearest neighbor, the distance between two datapoints is the absolute value of their difference (ex., |0–1| = 1). In other words, the distance between two samples is the sum of difference between attributes.[12 points]

	A_1	A_2	A_3	A_4	A_5	Label
S_1	1	0	0	1	1	Red
S_2	1	1	0	1	0	Blue
S_3	0	1	1	0	0	Blue
S_4	0	0	0	1	1	Red
S_5	0	1	0	1	0	Blue
S_6	1	0	1	0	1	Red

(a) Using KNN and Leave-One-Out Cross-Validation, where we test/classify each datapoint using the remaining datapoints (ex. classify S_1 using training dataset S_2 to S_6, then classify S_2 using training dataset S_1 and S_3 to S_6, etc), find the best k value between k=1,2,3 with the highest classification accuracy. Note that when breaking ties, choose the datapoint's label with the lowest instance number (if there is a tie between S_1 and S_2, assign the label of S_1). [7 points]

Solution:

Left out Pt	**S1**	**S2**	**S3**	**S4**	**S5**	**S6**
S1	X	2	5	1	3	2
S2	2	X	3	3	1	4
S3	5	3	X	4	2	3
S4	1	3	4	X	2	3
S5	3	1	2	2	X	5
S6	2	4	3	3	5	X

K = 1

TTTTTT 6/6 100% Accuracy, Highest classification accuracy

K= 2

FFTTTT 4/6 66.67% Accuracy

K= 3

TTTFTT 5/6 83.33 % Accuracy

(b) Now, test your classifier on the following data point below using KNN and find what label is assigned to S_7 for k=1,2,3. Note that when breaking ties, choose the datapoint's label with the lowest instance number (if there is a tie between S_1 and S_2, assign the label of S_1) [3 points]

	A_1	A_2	A_3	A_4	A_5	Label
S_7	0	1	1	1	0	?

Solution: Distances b/w S7 and S1-6:

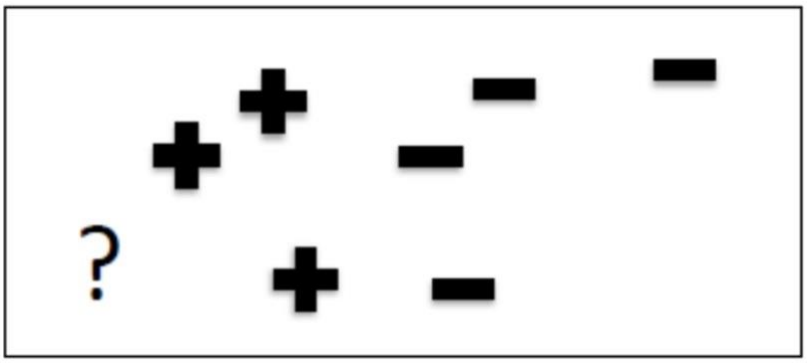
S1	S2	S3	S4	S5	S6
4	2	1	3	1	4

K = 1, S7 label = Blue

K = 2, S7 label = Blue

K = 3 S7 label = Blue

2. Given the dataset in the figure below where + illustrates a positive example and - illustrates a negative example, what is the minimal value of k in the k nearest neighbours classification algorithm such that the "?" query point is classified as negative, without considering ties? Please provide 1-2 sentences of justification for your answer. [3 points]



Solution: k = 7 as the nearest 3 points are positive, and in order for '?' to be classified as negative, the plurality needs to be negative, which means you need to include another 4 negative points, totaling 7 neighbors.

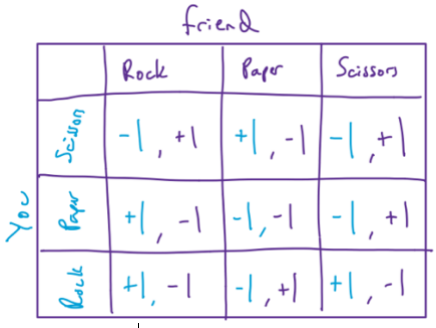
3 Game Theory [10 Points]

1. You decide to play rock-paper-scissors with a friend, but with a twist. You make the following changes to the rules -

   - If you throw rock and your opponent throws rock, you win and they lose
   - If you throw scissors and your opponent throws scissors, you lose and they win
   - If both of you throw paper, you both lose.

   (a) Construct the payoff matrix for this game. Pick suitable values for each players' payoff and justify your answer. [3 Points]

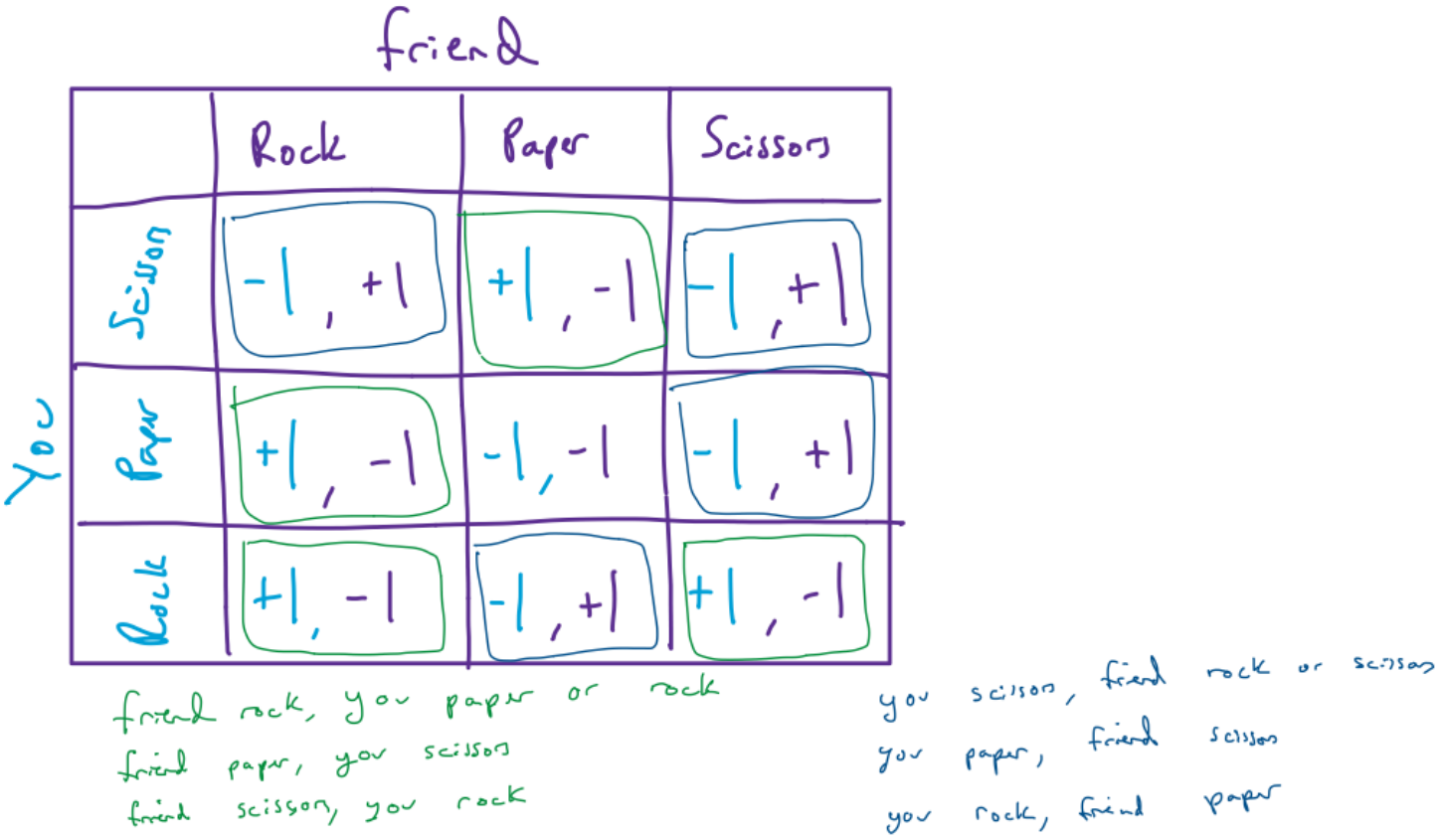
Solution:

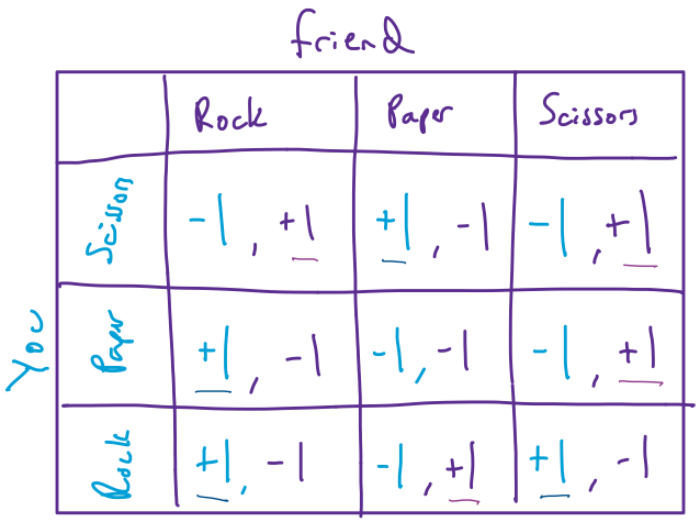


+1 for a win and -1 for a loss, so neither is weighted more than the other. This allows the game to be repeated in for best-of-three or more iterations, as is common practice when playing the game.

(b) Find the pure strategy nash equilibrium for this game, if it exists. If there are multiple, list them all. If there are no equilibria, justify why. [7 points]

Solution:



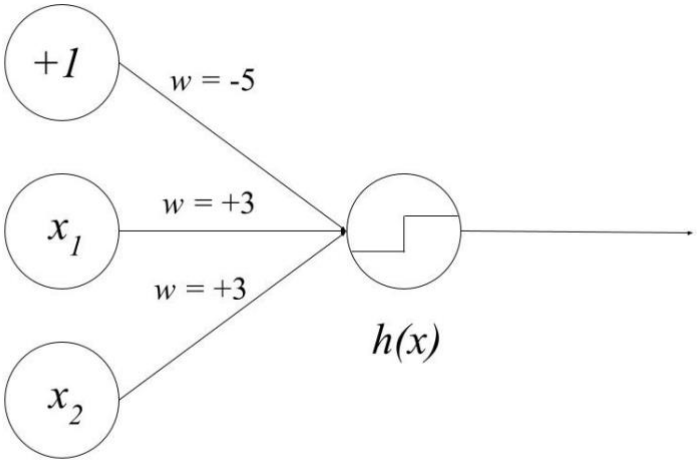


Underlines are the best replies for the agent given the strategy of the opponent.

There are no equilibria as there is no combination of actions where neither agent has an incentive to change.

4 Neural Networks [15 points]

1. The neural network below takes as input two binary valued inputs x_1 and $x_2 \in [0,1]$ and the activation function h(x) is the threshold function defined as h(x) = 1 if x > 0, and 0 otherwise. Which logical function does the neural network compute? Please justify your answer and include your computation. (Hint: You may find it useful to construct a truth table.) [3 points]



Solution: 1(-5) + 0(3) + 1(3) = -2, h(x) = 0 . FT → F

1(-5) + 1(3) + 0(3) = -2, h(x) = 0 . TF → F

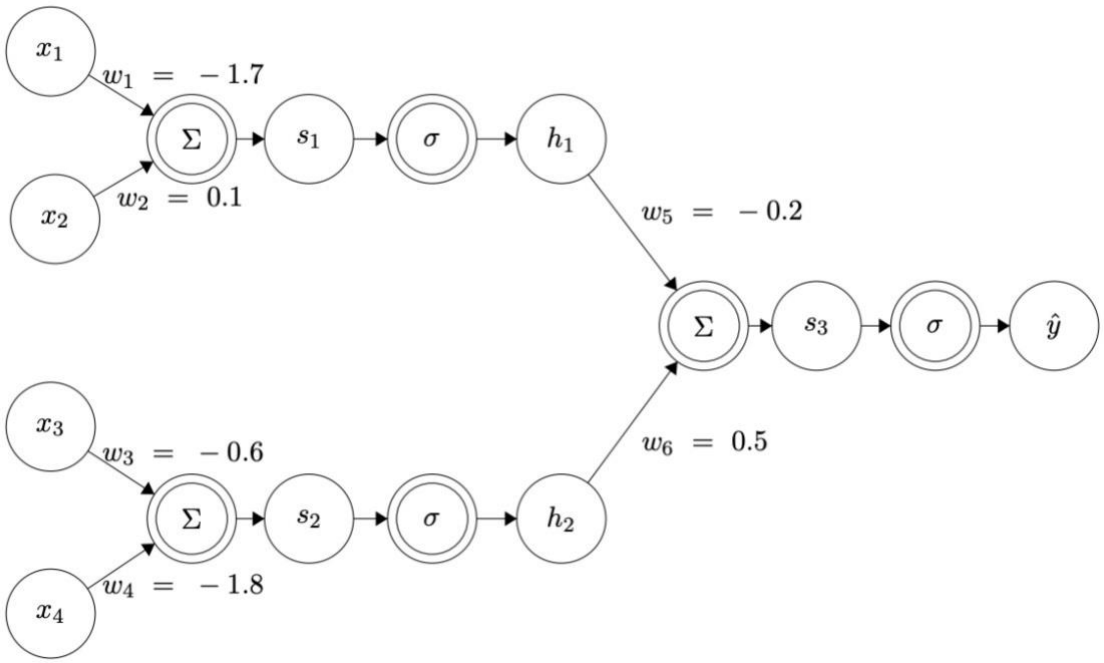
1(-5) + 1(3) + 1(3) = 1, h(x) = 1 . TT → T

1(-5) + 0(3) + 0(3) = -5, h(x) = 0. FF → F

The network computes AND as it only returns true when both x1 and x2 are 1s (true).

2. Consider the neural network illustrated in the figure below. Single-circled nodes denote variables (e.g. x_1 is an input variable, h_1 is an intermediate variable, and $\hat{y}$ is an output variable), and double-circled nodes denote functions (e.g. Σ takes the sum of its inputs, and σ denotes the logistic function $\sigma(x) = \frac{1}{1+e^{-x}}$,

Therefore in the network below the value of h_1 Will be computed as $h_1(x) = \frac{1}{1+e^{-x_1 w_1 - x_2 w_2}}$).



Suppose we have an L_2 loss $L(y, h_y) = ||h_y - y||_2^2$. We are given a data point $(x_1,x_2,x_3,x_4) = (-0.7,1.2,1.1,-2)$ with the ground-truth label of 0.5 [12 points]

(a) Write a function that represents the neural network. [3 points]

Solution:

sigmoid(sigmoid(w1x1 + w2x2)w5 + sigmoid(w3x3 + w4x4)w6))

(b) Please compute the forward pass for the given example input. [3 points]

Solution: y^ = 0.5787

b) $s_1 = -0.7(-1.7) + 1.2(0.1) = 1.31$

$s_2 = 1.1(-0.6) + (-2)(-1.8) = 2.94$

$h_1 = \sigma(s_1) = \frac{1}{1+e^{-1.31}} = 0.7875$

$h_2 = \sigma(s_2) = \frac{1}{1+e^{-2.94}} = 0.9498$

$s_3 = 0.7875(-0.2) + 0.9498(0.5) = 0.3174$

$\hat{y} = \sigma(s_3) = \frac{1}{1+e^{-0.3174}} = 0.5787$

(c) Calculate the loss function for the given example input. [3 points]

Solution: L(y, hy) = ||0.5787-0.5||^2 = 0.006194

(d) Use the equation below to compute the partial derivative $\partial L/w_1$. Use **3 significant figures** to report your answer. [3 points]
Hint: The gradient of an L_2 loss function $||h_y - y||_2^2$ is $2||h_y - y||_2$. Also, the gradient of a sigmoid function is $\sigma'(x) = \sigma(x)(1-\sigma(x))$.

Solution: $\partial L/w_1 = 0.000899$

d) $\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial s_3} \cdot \frac{\partial s_3}{\partial h_1} \cdot \frac{\partial h_1}{\partial s_1} \cdot \frac{\partial s_1}{\partial w_1}$

$= 2 \| \hat{y} - y \| \cdot \sigma'(s_3) \cdot w_5 \cdot \sigma'(s_1) \cdot x_1$

$= 2 \| 0.5787 - 0.5 \| \cdot 0.5787(1-5787) \cdot (-0.2) \cdot 0.7874(1 - 0.7874) \cdot -0.7$

$= 0.000899$

(1 – 5787) is actually (1-0.5787) I miswrote it.

Programming Section [50 Points]

Introduction and Objective

In this class, our goal is going to be to iteratively work towards building a fairly competent chess AI. In this assignment, we'll be creating and training a simple Deep Neural Network (DNN) Model using TensorFlow to determine which chess moves are good for Version 2 of our Chess AI. Before creating this model, we will generate our own Chess training data using utility functions provided. Finally, once we finalized the layers of our DNN Model, we will tune the model's hyperparameters so that we have an upgraded version of our previous model. To summarize, you will implement the generate lr training data(), build _model() functions, and the "Improving our DNN Model" section in the Google Colab notebook. After this, you will submit x train submission.npz, y train submission.npz, dnn _model.h5, dnn _improved model.h5, and your Google Colab notebook (as a .ipynb).

There are two qualitative questions that are also present in this section! Please attempt them after you complete the programming task and include your answers along with the written section!

Google Colab Notebook

For this programming section, because we're using TensorFlow, normally some OS and computers are unable to run this assignment. Additionally, setting up an environment for Tensorflow is difficult and time-intensive. Because of these two reasons, ML engineers use Google Colab so that they can automatically host their notebooks in virtual environments that can handle TensorFlow and can store huge databases for training.

To start this programming assignment, make a copy of 492hw3.ipynb Here. To successfully make a copy of the Google Colab notebook, go to "File" >> "Save a Copy in Drive" on the upper left-hand corner. With this new copy, you can change and compile your code similar to the Jupyter Notebooks from Homework 2! Note that there are additional instructions/hints in the notebook itself, so make sure you're reading all the text as you're running your code blocks. Additionally, if your runtime disconnects because of inactivity or with running runtime.unassign(), **you will have to run the necessary code blocks again**. Also note that we're not using GPU in our Google Colab, so do not manually enable GPU.

Training Data

For this part of the assignment, we'll be generating a training dataset for our DNN model by creating the generate lr _training data() function. Training data is valuable because it helps guide our DNN model to accurately decide which moves are good and bad. There are two parts of our training dataset that we need to consider: The chess boards (x train) and their associated scores based on an accurate evaluator (y _train). **Using our Model Util functions**, we will generate chess boards, evaluate their scores, and process the chess board into a 3D matrix so that our DNN model can accurately assess the board. Note that this 3D representation is one of many representations we can give the board, but because this representation is common in Chess ML models, we'll be converting our chess board into a 3D matrix. After we generated 100 boards of training data, we will automatically upload our arrays x train submission.npz and y train _submission.npz into our specified Google Drive folder and submit it to Gradescope for evaluation, specifically the Gradescope labelled "Homework 3 Coding - Training Data".

Model Utils

The provided methods for generating Training Data and running the DNN model are

1. random _board(max _depth) : this creates a random board (chess.Board) that had a random number between 0 - max depth previous moves generated. Note that these previous moves are all legal moves and the output will reflect the board after these random moves.

2. board score(board) : this returns an int score, where the score represents how good our inputted board is based on the SimplifiedEvaluator class in HW2.

3. split dims(board) : this splits our inputted chess.Board into a Binary 3D Matrix so that our DNN model can accurately assess moves and boards. Specifically, for our (14, 8, 8) np.array:

- The first 6 layers (0-5) represent white pieces (king, rook, bishop, queen, knight, and pawn) and their locations on the 8x8 board.
- The next 6 layers (6-11) represent black pieces (king, rook, bishop, queen, knight, and pawn) and their locations on the 8x8 board.
- The last 2 layers (12-13) represent white and blacks' legal moves respectively. This layer is so that the model can understand which pieces are being attacked when training.

DNN Model

For this portion, after we've created our training data, we'll develop our DNN Model using Tensorflow to have specific layers for accurate choices in the build_model function. Our implemented layers will be:

1. 1 Flatten Layer, which would flatten the input from 3D to 1D
2. 1 Dense Layer, which is a densely-connected Neural Network layer that takes in 64 units and uses ReLu activation
3. 1 Dense Layer, which is a densely-connected Neural Network layer that takes in 1 units and uses Sigmoid activation

For more information on TensorFlow.keras.layers and what layers are available, click here. We **highly, highly** recommend looking at this link for more information.
Once we create this layer structure, we can train our model successfully! The last code blocks in the "Deep Neural Networks" section will compile the layer model and define hyperparameters that guide the model. After the model is successfully trained, the model will save as dnn_model.h5 to your Google Drive, which should then be submitted to Gradescope, specifically the Gradescope labelled "Homework 3 Coding - Original DNN Model". As a sanity check, if the model's summary prints out the output below, then that means you've initialized the layers' output sizes correctly (Note that this doesn't guarantee that the layers' activation functions are initialized correctly)

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 14, 8, 8)]        0

 flatten (Flatten)           (None, 896)               0

 dense (Dense)               (None, 64)                57408

 dense_1 (Dense)             (None, 1)                 65

=================================================================
Total params: 57,473
Trainable params: 57,473
Non-trainable params: 0
```

Important Note: Because we saved the model on Google Drive, we can easily load our model for testing and submission.

Only attempt this written problem when you complete this programming section and pass on Gradescope! Looking at the training and validation loss, they seem to be around the same. What would the model indicate if the training loss is significantly smaller than the validation loss? [1 points]

Solution: That the model is underfiiting, model needs more complexity to better generalize.

Improving our DNN Model

After implementing our base model, if we run our model in the "Testing our DNN Model" section, we can see that this model is not only consistently losing to Stockfish AI, but also has a fairly high loss. Most of the time in the ML industry, the first model doesn't work effectively because the model's hyperparameters aren't tuned optimally. In this section (the last section of the Google Colab notebook), you will modify the hyperparameters of the DNN Model so that we can achieve a lower test loss. In Gradescope, when submitting dnn improved model.h5, we will return a test loss with our private testing dataset for comparing whether the

hyperparameter changes improved the model. **In order to receive full credit for this section, your test loss must be less than 0.035**.

2. In your written solution, please discuss the changes you made, your initial and final changes as well as their corresponding training and test losses, and finally why you decided to change these hyperparameters/ what effect did you hypothesize these changes will have below. [5 points]

Solution: I changed the loss function from mean_absolute_error to mean_squared error. I picked MSE because MAE doesn't punish large errors as it's the absolute average distance between real and predicted data, while MSE does as it's the squared average distance between real and predicted data, which I predicted would decrease the loss.

Old Model's Error (MAE):

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 14, 8, 8)]        0

 flatten (Flatten)           (None, 896)               0

 dense (Dense)               (None, 64)                57408

 dense_1 (Dense)             (None, 1)                 65

=================================================================
Total params: 57,473
Trainable params: 57,473
Non-trainable params: 0
_________________________________________________________________
Epoch 1/3
3/3 [==============================] - 1s 220ms/step - loss: 0.0653 - val_loss: 0.0631
Epoch 2/3
3/3 [==============================] - 1s 172ms/step - loss: 0.0622 - val_loss: 0.0611
Epoch 3/3
3/3 [==============================] - 0s 157ms/step - loss: 0.0604 - val_loss: 0.0597
```

New Model's Error (MSE):

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 14, 8, 8)]        0

 flatten_2 (Flatten)         (None, 896)               0

 dense_4 (Dense)             (None, 64)                57408

 dense_5 (Dense)             (None, 1)                 65

=================================================================
Total params: 57,473
Trainable params: 57,473
Non-trainable params: 0
_________________________________________________________________
Epoch 1/3
3/3 [==============================] - 1s 153ms/step - loss: 0.0070 - val_loss: 0.0069
Epoch 2/3
3/3 [==============================] - 0s 92ms/step - loss: 0.0068 - val_loss: 0.0068
Epoch 3/3
3/3 [==============================] - 0s 85ms/step - loss: 0.0067 - val_loss: 0.0067
```

3. For our 30,000 point training dataset, because our values are generated using Alpha-Beta pruning from HW2, generating this many points took 70+ hours between multiple core processors. Because training data took this long, why would we want to use Machine Learning instead of the original HW2 method? [2 points]

Solution: Because Machine Learning was faster at training the model on a large dataset than the Alpha-Beta pruning from HW2.