# CS 472 Final Project: Neural Network Multi-Class Prediction of Obesity Risk

Dax Lynch, 951895850

Google Colab Link, Run this and upload the train.csv and test.csv in the zip

## 1 Abstract

Obesity is a growing public health concern linked to various chronic diseases, including cardiovascular disease, diabetes, and certain types of cancer. This project aims to develop a multi-class classification model for predicting the risk of obesity in individuals based on a dataset from a Kaggle competition. The dataset includes 17 features, such as demographic information and healthcare-related factors. My approach involves preprocessing the data, implementing a multi-layer perceptron (MLP) model with ReLU activations, and evaluating the model on its accuracy. Experimental results demonstrate the effectiveness of my model in predicting obesity risk, suggesting potential applications in healthcare for early intervention and prevention strategies.

## 2 Introduction

This project focuses on developing a multi-class classification model to predict the risk of obesity based on various factors, which could aid in early intervention and prevention efforts.

The dataset used for this project is sourced from a Kaggle competition and includes 17 features encompassing demographic information and healthcare-related factors. The primary methodology involves preprocessing the data to convert categorical features into numerical representations, followed by implementing a multi-layer perceptron (MLP) model using the PyTorch library. Through exploration, hypothesizing, and testing, I was able to fine-tune the accuracy of the MLP.

## 3 Methodology

### 3.1 Data Preprocessing

The initial step involved data cleaning, which involved converting the .csv into a learnable format. Categorical features were converted into numerical values using one-hot encoding provide through the pandas *get_dummies* function. The dataset was split into training and test sets to facilitate model evaluation. Care was taken to ensure both test and train splits had equal proportion of classes. By utilizing the sklearn's $StratifiedShuffleSplit$, I could attain a split that respect class proportions. Then I converted the data into pytorch tensors for processing.

### 3.2 Model Architecture

I selected a multi-layer perceptron (MLP) due to its simplicity and effectiveness in handling multi-class classification tasks. The MLP architecture consisted of an input layer, hidden layers with ReLU activations, and an output layer to handle multiple classes. Hyperparameters such as the weight regularization, number of epochs, number of hidden layers, and the size of those layers were tuned in a series of experiments to optimize model performance. By keeping other factors the same and then varying hyperparameters in a grid or range search allowed me to gain multiple percentage points of accuracy on my test data.

The MLP was implemented using the PyTorch library. The training procedure involved defining a training loop, selecting an appropriate cross-entropy loss as the loss function and ADAM as the optimizer. Regularization techniques such as dropout and weight decay were both tested to prevent overfitting but both lead to a greatly increased time per epoch and no appreciable accuracy improvement.

## 4 Experiments

The tuning process involved performing a grid and range searches for each hyperparameter, fixing the values of the other hyperparameters to their best found values so far. This approach assumes independence between the hyperparameters.

### 4.1 Dropout and weight regularization

A dropout layer was added after the first hidden layer, and tested with values of .2, .1, and .05. All three values lead to much longer training times, later crossover epochs, and lowered accuracy. Hence in the final model I chose to not use dropout.

Weight regularization was added as a parameter to the ADAM optimizer. At first I tried a value of 1, which lead to an accuracy of 19%. I then tried the value .01, and .0001. Accuracy of the model was improved at none of these values.

### 4.2 Architecture

Arguably the most important parameters, I spent a majority of my time optimizing the architecture. I performed a grid search, varying the number of hidden layers and the size of the layers. Because I knew that increasing model complexity could increase overfitting, I recorded the difference between the train and the test accuracy at the end of the test. If this delta is larger, it implies the model is more apt to overfit in the 50 epochs they were trained over. The results are graphed below

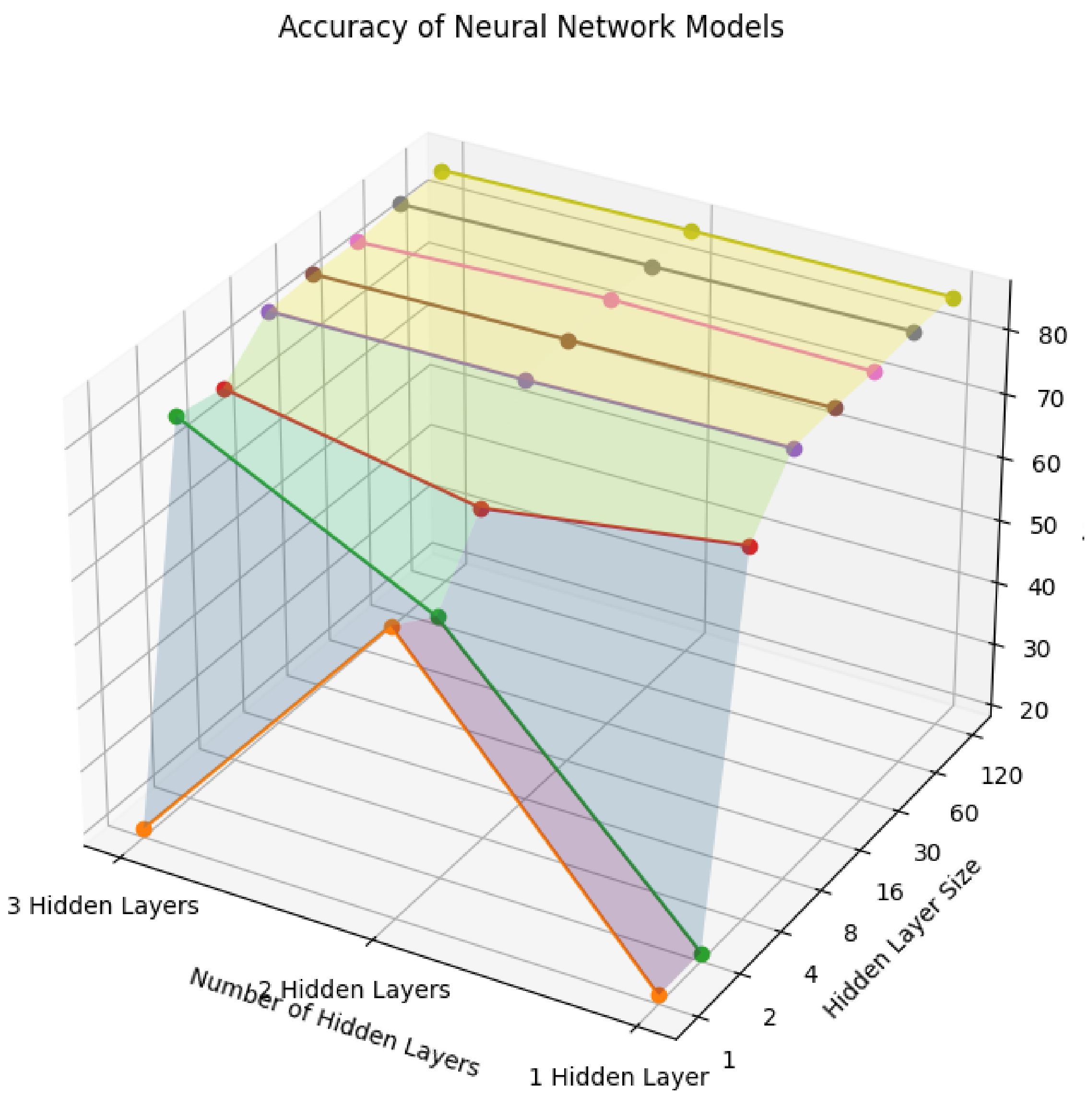


Figure 1: Accuracy vs Hidden Layers vs Layer Size

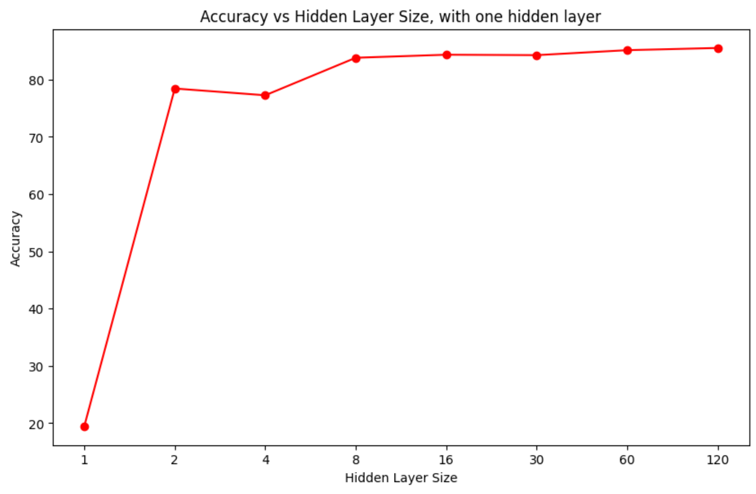


Figure 2: Accuracy vs Layer Size

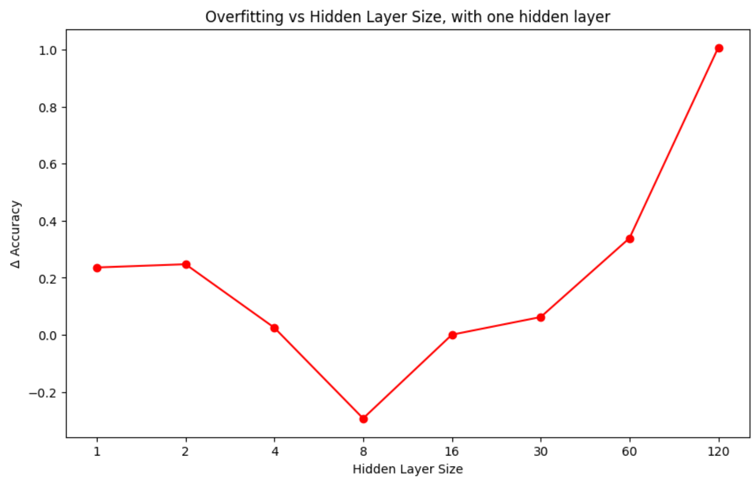


Figure 3: Overfitting vs Layer Size

After looking at these results, I decided to use 2 hidden layers of 60 neurons each. I decided on 60 as it seemed like the best middle ground for accuracy and overfitting. I decided on 2 layers as it had little deficit compared to 3, but increased efficiency. Below is the accuracy and loss graphs for this architecture. The red line represents the training data, while the blue represents the test data.

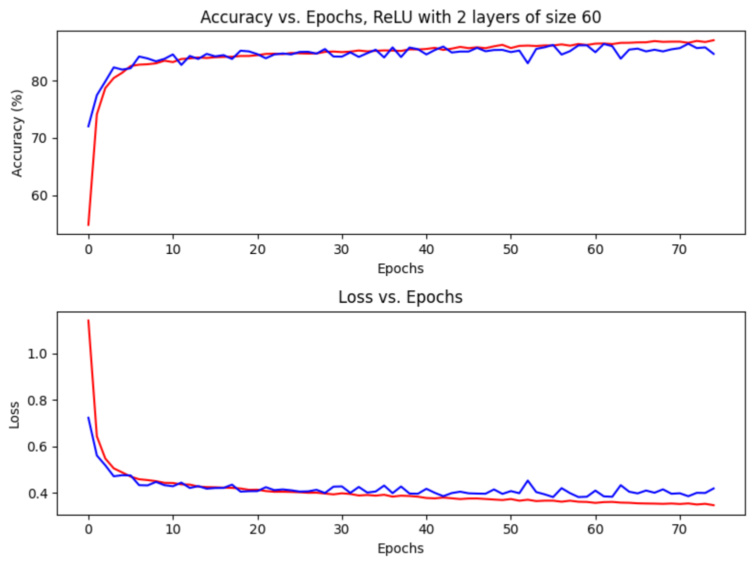


Figure 4: Accuracy and Loss vs Epochs

### 4.3 Class proportion

My class sizes were originally unequal, ranging from 2427 to 4046 items in the classes. My thought process was two similar classes, such as Overweight Type III and Overweight Type II might have very similar data, but different proportions. If the model sees data that looks like either of those it will choose the one with the higher proportion as it leads to higher accuracy rather than learning discriminating features. Therefore, I reduced the dataset so every class was equally represented and retrained the model on the equal data while testing it on the unequal data. Once I had trained on the equal data it became clear that the model had not converged, unlike previously.

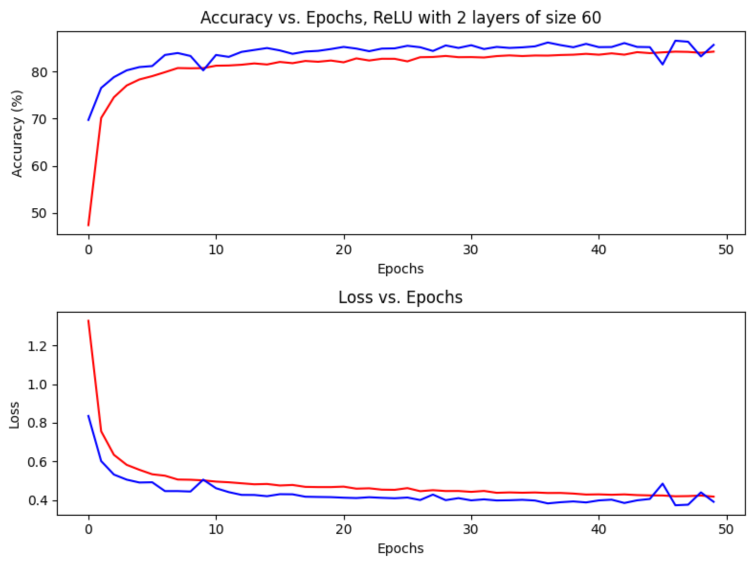


Figure 5: Accuracy and Loss vs Epochs, Equal Class Proportions

This implied that the model was still learning on the data. I decided to increase model size to 3 layers of 90 neurons trained over 200 epochs since the data was now less ”overfittable”.

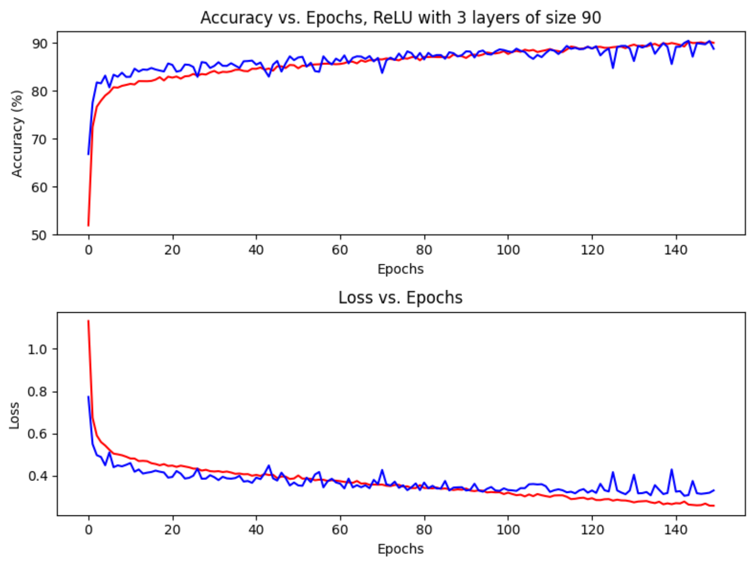
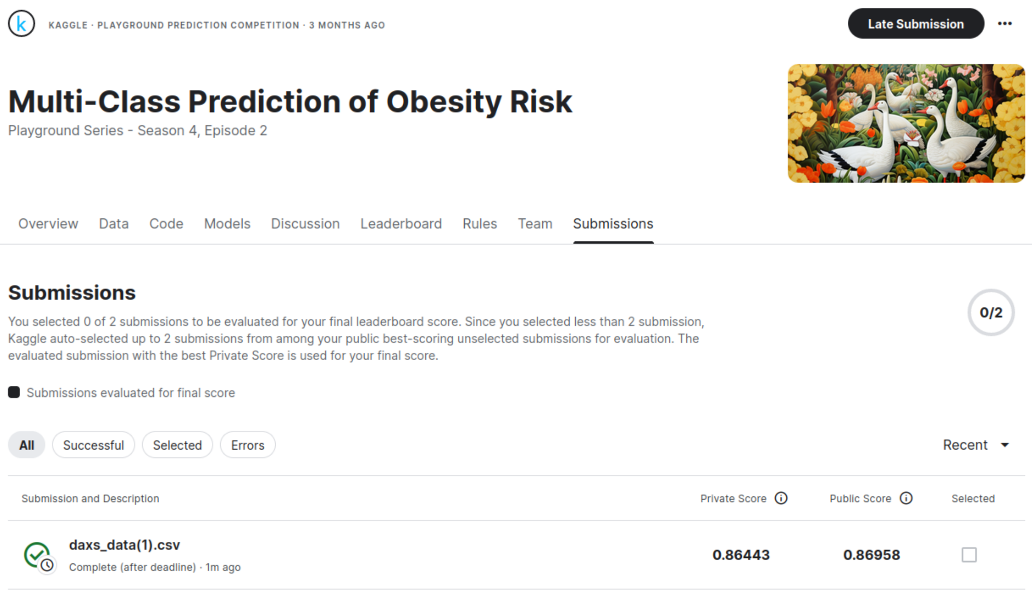


Figure 6: Accuracy and Loss vs Epochs, Equal Class Proportions, Larger Model

This architecture yielded a best accuracy on the test data of 90.41%.

### 4.4 Validation

I then used the model to perform classification on unlabeled data provided by Kaggle, I then uploaded those labels to Kaggle and it reported an accuracy of 86.95%



KAGGLE · PLAYGROUND PREDICTION COMPETITION · 3 MONTHS AGO

Late Submission

**Multi-Class Prediction of Obesity Risk**

Playground Series - Season 4, Episode 2

Overview Data Code Models Discussion Leaderboard Rules Team Submissions

**Submissions**

You selected 0 of 2 submissions to be evaluated for your final leaderboard score. Since you selected less than 2 submission, Kaggle auto-selected up to 2 submissions from among your public best-scoring unselected submissions for evaluation. The evaluated submission with the best Private Score is used for your final score.

0/2

■ Submissions evaluated for final score

All Successful Selected Errors Recent

| Submission and Description | Private Score | Public Score | Selected |
|---|---|---|---|
| daxs_data(1).csv<br>Complete (after deadline) · 1m ago | 0.86443 | 0.86958 | ☐ |

Figure 7: Kaggle Validation Accuracy

## 5 Conclusion

In this project, I developed a multi-class classification model to predict obesity risk using a Kaggle dataset. The final architecture consisted of three hidden layers with 90 neurons each, trained over 200 epochs. This configuration achieved a test accuracy of 90.41%. When applied to Kaggle's unlabeled data, the model attained an accuracy of 86.95

The results demonstrate the model's effectiveness in predicting obesity risk, highlighting its potential application in healthcare for early intervention and prevention strategies.

## 6 Collaborator Credit

No collaborators were consulted for this assignment.