Introduction and Data Analysis

This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective of the dataset is to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset. Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage.

Data Characterization

- Volume: 768 rows of data
- Quality: The dataset is recorded within a supervisor of the National Institute of Diabetes and Digestive and Kidney Diseases
- Bias: The dataset is only collected within Pima Indian heritage females that are at least 21 years of age so the dataset cannot represent all females chances of having diabetes using the same measurements

Input Variables (X)		
Attribute	Range	Mean
1. Number of times pregnant	0-17	3.845
2. Plasma glucose concentration at 2 hours in an oral glucose tolerance test	0-199	120.895
3. Diastolic blood pressure (mm Hg)	0-122	69.105
4. Triceps skin fold thickness (mm)	0-99	20.536
5. 2-hour serum insulin (mu U/ml)	0-846	79.799
6. Body mass index (weight in kg/(height in m)^2)	0-67.1	31.993
7. Diabetes pedigree function	0.078-2.42	0.472
8. Age (years)	21-81	33.241
Output Variables (y)		
Attribute	Range	Mean
9. Class variable	0 or 1	0.349

Algorithm design

Part 1:

Pre-processing

According to the prompt, X is given to be an input with attribute no.1 - no.8, while Y is labeled as an output with attribute no.9. Therefore, we defined X with the first to second last column of the dataset and Y with the last column of the dataset, during our loading dataset process.

Training and Testing

Start by setting train to 0.8 so that the model will use 80% of the dataset for training and use the 20% remaining dataset for validation. train_b is set to get 80% of the dataset rows. X_train and Y_train are variables for training dataset by using the first row of data to row 614th which is 80% of the rows. X_test and Y_test are set from row 614th onwards.

Build a model

While building our model, we used ReLU as an activation function in our hidden layers because it's fast to compute, while also avoiding vanishing gradient problems. However, for the output layer, we used sigmoid function due to its ability to produce numbers between 0 and 1, which we needed for probability.

Compile a model

We choose Binary Cross Entropy for the loss function because the output is whether 0 or 1, Adam for optimizer, and the metrics focus on accuracy.

Train a model

For training, we set epochs to 150 and batch size to 10 since we want a relatively small model.

Part 2:

Pre-processing

We imported more libraries such as activation, dropout and regularizer for further uses. However, the rest are the same as the previous version (Part 1), since our dataset remained the same and we loaded the dataset in the same manner.

Training and Testing

Instead of using the same method as part1, in this part we use the Scikit-learn (sklearn) library since it is simpler and a more straightforward method. After multiple experiments, we found that using 70% to train the model and 30% to test gives more accurate results.

Build a model

The main structure frame of our new model are quite similar with the previous model. We continued to use ReLU as an activation function for the hidden layers and sigmoid for the output layer for the same reasons we did on the first part. Though, the differences between the old and new model are the add-on functions, such as regularizer and dropout, as well as the number of hidden layers and the nodes of each layer. For regularization, we only use L1 since it is a better solution for sparse data and the dataset we are using contains a number of zero.The numbers of layers, dropout, regularizer and nodes for each layer were determined through multiple experimentations. Generally, dropout rate in hidden layers is recommended around 0.5 - 0.8. But for our case, the dataset is quite small so we use a rather small number of 0.2 instead.

Compile a model

The compiling model process of ours remained exactly the same; compile using binary cross entropy as a loss function and adam, stochastic gradient descent, for the optimizer.

Train a model

Through numerous experiments, the new training model procedure remained the same as our previous task. The number of epochs is still 150, but batch size is changed to 15 since from our multiple tests, these numbers give the best accuracy result.

Evaluation

Pre-processing

Strengths:

- Loadtxt uses less memory.

Weaknesses:

- It runs very slowly.

Suggestions:

- Use pd.read_csv instead of loadtxt, in which pd stands for pandas.

Training and Testing

Strengths:

- For part 1, the code can be implemented without having to know other libraries.
- For part 2, using sklearn is easier to use and read.

Weaknesses:

- The result wasn't quite accurate with the 80/20 split training and testing dataset.

Suggestions:

- Splitting data using sklearn is much easier to implement and understand the code.
- Splitting the number of training and testing dataset could change, to give more accurate results. In this case, we change the splitting ratio to 70/30 in part 2.

Build a model

Strengths:

- Part 1 - Used ReLU in hidden layers for faster computation and avoiding vanishing gradient problem.

Weaknesses:

- Part 1 - Overfitting or underfitting problems could occur, due to the lack of regularizer and dropout.
- Part 2 - By increasing the number of hidden layers and nodes, the model takes bigger toll on computing. Thus, it may result in taking longer to finish running.

Suggestions:

- The number of epoch, batch size, regularization, dropout, hidden layer and nodes could be adapted to give more accuracy
- Implement other measures to prevent overfitting or underfitting. Such as using bigger datasets to train, early stopping, and etc.

Compile a model

Strengths:

- Adam, as an stochastic gradient descent, is one of the best optimisers, while also requiring less parameters for tuning.

Weaknesses:

- The dataset we are using is spare, which may cause problems in predicting the accuracy.

Suggestions:

- In case the dataset was not pre-processed in the right manner, it should be redone so the problem of sparse dataset can be lessened.

Train a model

Strengths:

- Part 1 and 2, we kept the same number of epochs since 150 epochs gives quite high accuracy.

Weaknesses:

- It takes a while to train the model, since 150 epochs is quite a large number of repetition.

Suggestions:

- For the 2 part we could try to change up the numbers for epoch and batch size for a possibly higher accuracy

Conclusion

In this project, we aimed to investigate and modify the deep learning model in order to determine how the modification of various model parameters, such as the number of hidden layers, the number of nodes in each layer, or the ratio of data sampling, would affect the model's accuracy.

We used several techniques from multiple sources, including regularization, dropout layer, dataset splitting, adding hidden layers and nodes, to modify our model from the base model in part 1 of the project to be more effective. We observed that after applying those techniques to our model along with trials and errors, the accuracy of the model was increased accordingly. After many modifications, we obtain a more accurate result and the graph has become more stable (Figure 2) compared to Figure 1.

Therefore; in order to enable a more efficient result, we can obtain different permutations and combinations of data, additional data sampling approaches, such as shuffled sampling, stratified sampling, or automatic sampling, could be used in future improvements.

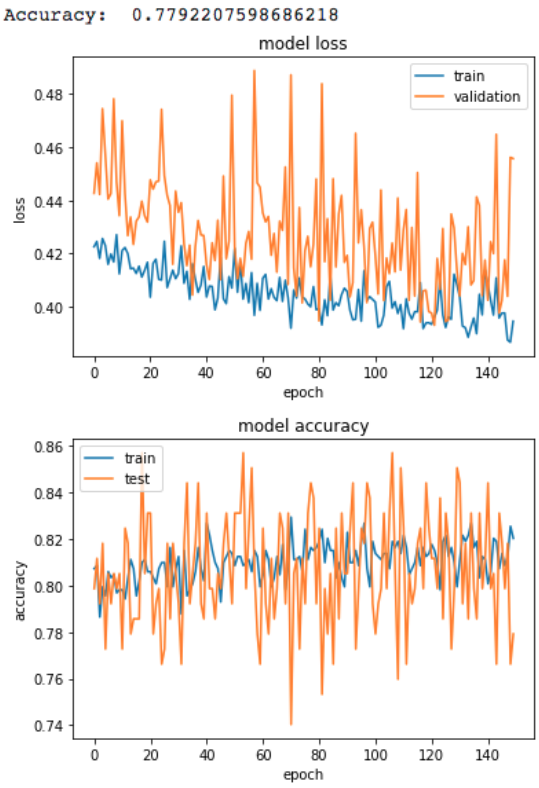

Figure 1: The model's result of Part 1

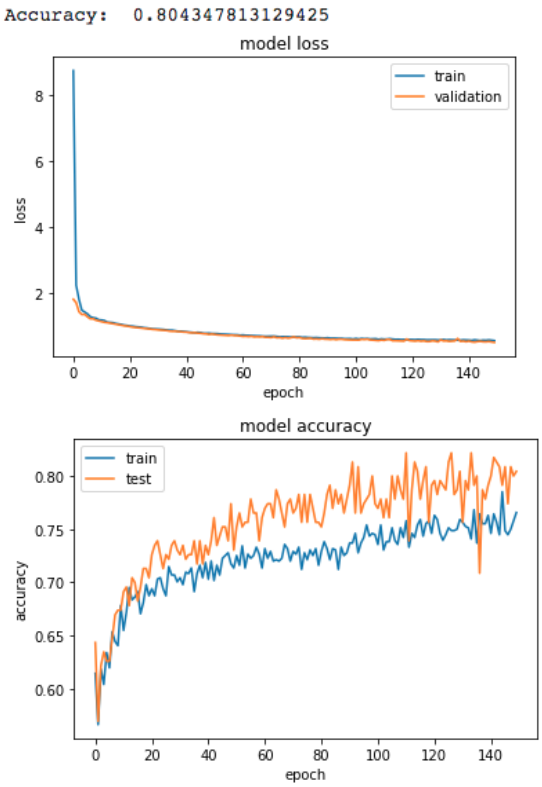

Figure 2: The model's result of Part 2

References

Kumar. (n.d.). *pima-indians-diabetes.csv*. Kaggle. Retrieved September 1, 2022, from https://www.kaggle.com/datasets/kumargh/pimaindiansdiabetescsv

Brownlee, J. (2018, December 3). *A Gentle Introduction to Dropout for Regularizing Deep Neural Networks*. Machine Learning Mastery. Retrieved September 3, 2022, from https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/

Chaudhary, M. (2020, August 27). *Activation Functions: Sigmoid, Tanh, ReLU, Leaky ReLU, Softmax | by Mukesh Chaudhary*. Medium. Retrieved September 3, 2022, from https://medium.com/@cmukesh8688/activation-functions-sigmoid-tanh-relu-leaky-relu-softmax-50d3778dcea5

Myelin Foundry. (2020, January 27). *Deep Learning can tackle the problem of sparse data by augmenting available data*. Myelin Foundry. Retrieved September 4, 2022, from https://www.myelinfoundry.com/deep-learning-can-tackle-the-problem-of-sparse-data-by-augmenting-available-data/

Pykes, K. (2022, July 22). *Fighting Overfitting With L1 or L2 Regularization: Which One Is Better? - neptune.ai*. Neptune.ai. Retrieved September 4, 2022, from https://neptune.ai/blog/fighting-overfitting-with-l1-or-l2-regularization