Singapore University of Technology and Design

50.039 Deep Learning Project

Authors:	*Student ID:*
Kim Mi Ran	1005612
Lee Chien Shyong	1005903
Leong Keng Hoy	1005164

April 14, 2024

Contributions

- **Mi Ran:** 1- and 6-Layer Neural Networks, Comparison with state-of-the-art models
- **Chien Shyong:** 3- and 4-Layer Neural Networks, Dataset analysis, Feature engineering, Hyperparameter tuning
- **Keng Hoy:** 2- and 5-Layer Neural Networks, Hyperparameter Grid Search, Tuning of final model

1. Introduction of the dataset

Overview of the problem

Type-2 diabetes is typically diagnosed through a combination of medical history, physical examination, and laboratory tests. To assist with the diagnosis of diabetes, we have elected to develop a Deep Learning model that classifies patients according to their likelihood of having diabetes based on various features. The selected dataset was sourced from the United States National Institute of Diabetes and Digestive and Kidney Diseases, which can be found online via this link:
https://www.kaggle.com/datasets/akshaydattatraykhare/diabetes-dataset

This dataset only includes females at least 21 years old of Pima Indian heritage (a Native American group, which was selected due to their high susceptibility to type-2 diabetes).

Features of the dataset

The 8 features in this dataset are the patient's number of pregnancies, blood glucose level, blood pressure, skin thickness, blood insulin level, body mass index, diabetes pedigree function and age. The format of the data is a Comma Separated Values (csv) file. There are 768 data points in the dataset.

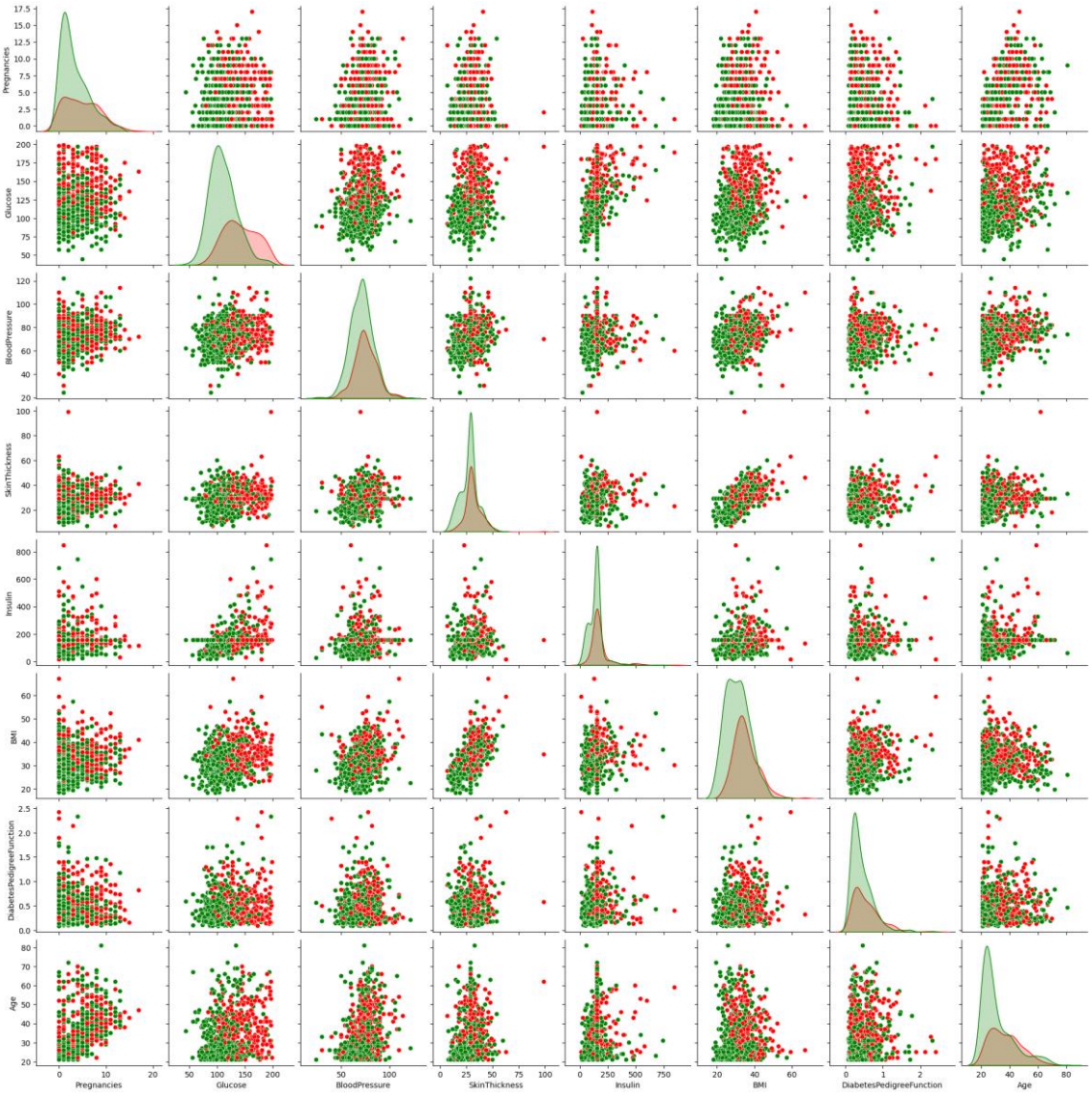
However, the dataset has some missing data points, where if data is missing, a zero is indicated. Hence, we will need to apply some feature engineering to deal with this dirty data.

2. Initial Approaches Taken

Data Visualization & Feature Engineering

As mentioned previously, there were some missing data to deal with. Our approach is to replace all instances of '0' in a feature column with the average of the rest of the column. Hopefully, this would cause each missing datapoint to have as little influence as possible on the model's decision.

Below is a pair-plot of all 8 features after data cleaning (Green = Healthy, Red = Diabetes)



We also calculated the correlation between each feature and the output to see which feature has the most influence on the likelihood of diabetes.

```
Glucose                     0.492928
BMI                         0.311924
Age                         0.238356
Pregnancies                 0.221898
SkinThickness               0.215299
Insulin                     0.214411
DiabetesPedigreeFunction    0.173844
BloodPressure               0.166074
Name: Outcome, dtype: float64
```

Exploring different model architectures

We wish to model neural networks of differing layers and hyperparameters to find one which can predict if a patient has diabetes with reasonable accuracy. As we have three members, our approach was to individually build slightly different models to explore what model works best, and then select the best model out of what we have to further tune.

1-Layer and 6-Layer (Mi Ran)

The dataset was split 80-20 by train-test split. 1-layer and 6-layer neural networks were built then trained with differing hyperparameters listed below.

Learning rate: 1e-3, 1e-2, 1e-1

Activation function: ReLU (6-layer), Sigmoid (1-layer)

Optimizer: Adam, Adagrad

Number of epochs: 120, 260, 500, 1000, 2000

For the 6-layer neural network, a dropout layer was applied between every FC layer, ranging from 0.1 to 0.5.

2-Layer and 5-Layer (Keng Hoy)

Hyperparameter grid search was used to find a starting point for building the neural networks with 2 and 5 layers. The hyperparameters of interest were the number of neurons in each hidden layer, the activation function, the optimizer, and the learning rate.

For the 2-layer neural network, searches were run for 200 epochs, and the search space was as follows:
Number of neurons in the hidden layer: 16, 32, 64, 128

Activation function: ReLU, CELU
Optimizer: Adam, Adadelta, RMSprop, RAdam, Adagrad
Learning Rate: 0.005, 0.01, 0.02

For the 5-layer neural network, searches were run for 100 epochs, and the search space was as follows:
Number of neurons in the hidden layer: (256,128,64,32), (160,80,40,20), (128, 64, 32, 16)
Activation function: ReLU, CELU
Optimizer: Adam, Adadelta, RAdam, Adagrad
Learning Rate: 0.005, 0.01, 0.02

The dataset was normalized, then split using an 80-10-10 training-validation-test split, although early stopping was ultimately not implemented as validation accuracy never went higher than 0.8. A batch size of 128 was used for training, 64 for the other sets.

Once the best performing set of hyperparameters for each neural network were found, the model was tested using those hyperparameters, with 2000 epochs for the 2-layer network and 1000 epochs for the 5-layer network. As the models were found to overfit to the training set, the addition of dropout was experimented with.

3-Layer and 4-Layer (Chien Shyong)

I attempted to build two neural networks, experimenting with three layers (two hidden layers) and four layers (three hidden layers). Each hidden layer used ReLU activation function, with a sigmoid as the final activation function. BCE Loss was used with Adam Optimizer (learning rate = 1e-2, betas = (0.9, 0.999), eps = 1e-08).

The dataset was split 80-20 into training and test sets, which were normalized before training. A fixed batch size of 64 was used. Variables such as number of epochs and neurons in each layer were left to be experimented with.

3. First Results

1-Layer and 6-Layer (Mi Ran)

Here are some of the results for 1-layer neural network with differing learning rates and optimiser Adam.

Epochs /Learning rates	1-Layer Test Accuracy		
	1e-3	1e-2	1e-1
120	0.7370	0.7721	0.7695
260	0.7396	0.7799	0.7786

500	0.7292	0.7773	0.7760
1000	0.7604	0.7734	0.7695
2000	0.7708	0.7773	0.7721

It seems that all variations converge between accuracies 0.7 and 0.8. Even after training for 5000 epochs with learning rate of 1e-3, accuracy has shown to be 0.7747

The results of 6-layer neural network with learning rate of 1e-3, optimizer Adagrad, and differing dropout rates are depicted in the table below. Number of neurons in hidden layers goes as [32, 64, 128, 128, 32]

Dropout /Epochs	6-Layer Test Accuracy	
	1000	2000
0.1	0.7734	0.7539
0.2	0.7083	0.6953
0.3	0.6615	0.7031
0.4	0.6510	0.6501
0.5	0.6510	0.6539

Loss for 6-layer neural network while training remained between 0.6 and 0.7 with little variance, suggesting that it is overfitting. Dropout rate of 0.1 seems to show the best accuracy.

2-Layer and 5-Layer (Keng Hoy)

The results of the neural networks are as follows:

2-Layer Neural Network: 32 neurons in hidden layer, ReLU activation, RMSProp optimizer, 0.01 learning rate

Dropout	2-Layer Test Accuracy
0	0.7532
0.2	0.7922
0.4	0.7532
0.5	0.7532
0.75	0.7662

It was observed through accuracy graphs that the training and validation accuracies were very volatile, suggesting that the learning rate might be too high, and that there might not be enough neurons in the hidden layer. It is possible that this set of hyperparameters performed the best out of sheer luck, hence more experimentation should be done in the direction of reducing the learning rate and increasing the size of the hidden layer, as well as adjusting the optimizer if necessary.

5- Layer Neural Network: (256, 128, 64, 32) neurons in hidden layers, ReLU activation, Adagrad optimizer, 0.01 learning rate

Dropouts	5-Layer Test Accuracy
(0,0,0,0)	0.7532
(0.2,0.2,0.2,0.2)	0.7662
(0.25, 0.5, 0.5, 0.25)	0.7662
(0.4, 0.4, 0.4, 0.4)	0.7922
(0.5, 0.5, 0.5, 0.5)	0.7922
(0.75, 0.75, 0.75, 0.75)	0.7013

It is observed that the training accuracy quickly reaches 100% when no dropout is used, and still reaches 100% within 1000 epochs with dropouts under 0.5, indicating a strong overfitting and thus strongly suggesting that 5 layers might be too much.

3-Layer and 4-Layer (Chien Shyong)

Below, I tabulated some results from adjusting the parameters for my 3- and 4- layer Neural Networks.

3-layer test accuracy			
Neurons in hidden layers	32 Epoch	64 Epoch	128 Epoch
8, 8	.6948	.7078	.7273
16, 16	.6558	.6861	.6558
32, 32	.6688	.6558	.6623
4-layer test accuracy			
Neurons in hidden layers	32 Epoch	64 Epoch	128 Epoch
8, 8, 8	.6948	.6883	.7143
16, 16, 8	.6883	.7143	.6948
32, 32, 16	.6818	.6948	.7078

For all these examples, training accuracy exceeded 90%. Especially with 128 epochs of training, the training accuracy was close to 100%. The difference between training and test accuracy clearly shows a large amount of overfitting to the training data, which will need to be rectified.

Additionally, all the parameters that I experimented with for 3 or 4 layers underperformed the model with only 2 layers, which usually reached around 75% test accuracy. Hence, I concluded that three layers introduced too much complexity for this classification task and was responsible for some of the observed overfitting.

4. Hyperparameter Tuning

Based on the initial findings, we chose to use the 2-layer model as a base for our model, then worked on adjusting the hyperparameters to make the training more stable to confirm the reliability of our results.

Hyperparameters were manually tested over 2000 epochs, with hidden layer sizes of 32,64,128,256,512,1024 and learning rates of 0.0001 and 0.00001 being tested over the optimizers of RMSProp, RAdam and Adam. Dropout was not tested as training accuracy never went above 0.9 even at 0 dropout, likely due to the significantly lower learning rates.

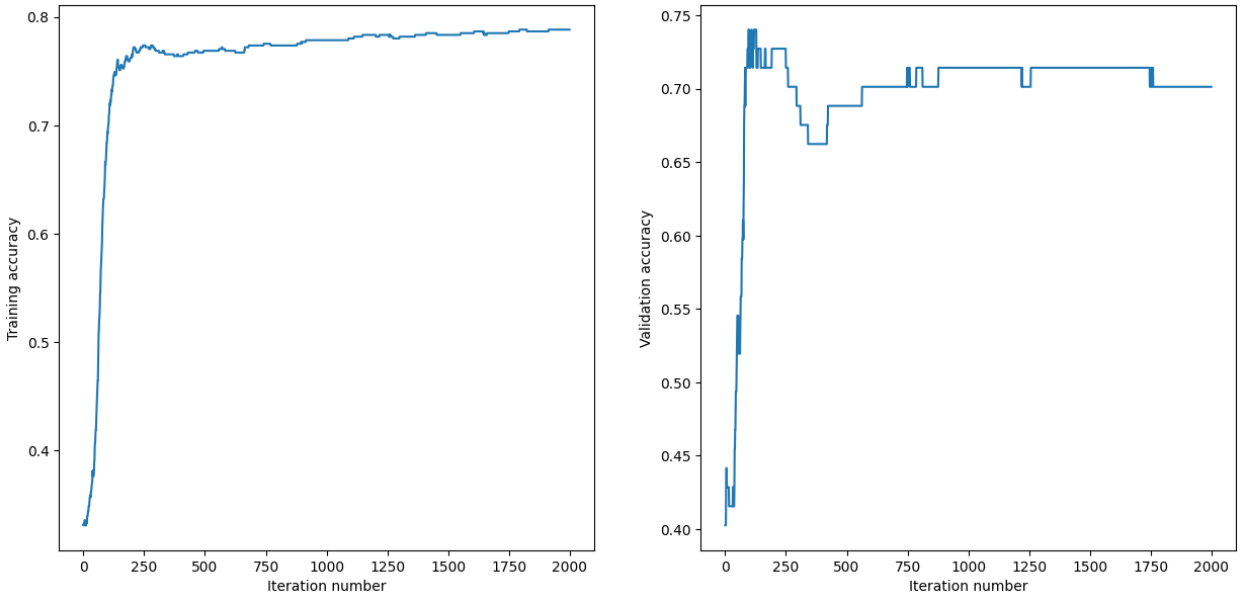
Not every combination was tested, but the graphs showed that reducing the learning rates to 0.0001 and 0.00001 made the training accuracy significantly more stable. Furthermore, tests using the hidden layer size of 512 and learning rate of 0.00001 seemed to have the best overall performance.

5. Final Results

Our final model of choice has the following hyperparameters and results:

Hidden Layer Nodes: 512, Activation: ReLU, Optimizer: Adam
Learning Rate: 0.00001, Dropout: 0, Test Accuracy: 0.8052

The following figure shows the training and validation accuracy over the training epochs and can be recreated by following the steps described later in Part 6: Code Implementation regarding training the model.



Comparison with State-of-the-art models

Using built-in models in Sklearn, our final model was compared with Random Forest Classification (RFC) model, Gaussian Naive-Bayes (NB), and Support Vector Classification (SVC) model. The results of training the dataset with the above-mentioned models are tabulated below.

Model	Accuracy
RFC	0.7597
Gaussian NB	0.7467
SVC	0.7467
Final Model	0.8052

6. Code Implementation

In the GitHub repository, the subdirectories contain various notebooks with the parts done by each member. The notebook in the "chien" folder contains the work done for Chien Shyong's 3-layer and 4-layer neural networks. The notebook in the "kh" folder contains the work done for Keng Hoy's 2-layer and 5-layer neural networks. For the "miran" folder, layer1.ipynb contains the work done for Mi Ran's 1-layer neural network, layer5.ipynb for her 6-layer neural network, and SOTA_models.ipynb contains the training and testing of the state-of-the-art models.

Outside the folders, the most important notebook is final_model.ipynb, which contains the final round of hyperparameter testing on the 2-layer neural network as well as the training and testing of the final model itself.

To train the model, first ensure that the "diabetes_clean.csv" file is in the same directory as the "final_model.ipynb" notebook, then run the cells in the "Imports", "Load Dataset", "Setting Up Models", "Setting Up Trainer" and "Training Final Model" section.

To load and test the trained model, make sure the cells in the "Imports", "Load Dataset" and "Setting Up Models" sections have been run, then run the cells in the "Loading the model" and "Testing the loaded model" sections.

The final notebook has the following dependencies: matplotlib, numpy, pandas, torch, torchmetrics. Specifically, it was tested with the following versions: matplotlib v3.8.2, numpy v1.23.1, pandas v1.4.3, torch v2.2.0, torchmetrics v1.3.1 and Python 3.10.0.