A Deep Learning Approach to Predicting Credit Card Defaults

Abstract

When predicting credit card defaults, a Deep Feedforward Neural Network implemented with a simple model architecture proved to be the most consistent and thorough. A model architecture of one hidden layer with an applied dropout and the use of regularisation techniques outperformed other approaches. This approach and a systematic hypertuning process played a vital role in creating an optimal model for predicting credit card defaults.

1 Introduction

The task was to create a neural network that would accurately and efficiently predict credit card defaults. The given dataset consisted of personal information and transaction information belonging to credit card holders.

When choosing which type of neural network to employ, I factored in the characteristics of the given dataset and the features of said dataset. The dataset was tabular, static and the features were independent of one another and were of fixed size. Therefore I decided to employ Deep Feedforward Neural Networks (D-FFNNs) over other models such as Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs).

RNNs are effective when processing sequences and time-series data due to their recursive nature. However these characteristics were not adequately present in the dataset. Furthermore, D-FFNNs are advantageous for their ability to model complex mappings between inputs and outputs without the need for considering temporal information. Additionally, CNNs are not distinguishably better than D-FFNNs or RNNs when dealing with numerical data. They are more suited when there is a need to capture spatial relationships.[1]

The simpler architecture of a D-FFNN when compared to RNNs and CNNs often lead to faster training times. D-FFNNs typically use fewer hyperparameters than RNNs and CNNs which allows for a quicker process of obtaining an optimal model. In the financial field where a model such as my proposed model would be used, the efficieny interpretability of the model would be vital when making financial decisions. This supports the use of a D-FFNN.

Regarding the structure of this report, Section 2 details the proposed methodology and data preprocessing used to create my final model. Section 3 is an evaluation of the experimental results of the different models that I implemented when performing this task.

The hyperparameter settings, graphs and resultant values of each model are provided along with the process of creating said model. Section 4 is a summary in which I discuss and highlight my main findings.

2 Proposed Methodology and Preprocessing

2.0.1 Proposed Methodology

I first decided to create a basic D-FFNN that would serve as the foundation for future iterations of the model. These iterations would be subsequently altered to address the shortfalls of the previous iteration. The changes made to each iteration were made based on the results and graphs produced by a model. Hyperparamters, model architecture, different regularisation techniques and dropout would be introduced and altered in order to obtain an optimal model.

An iterative approach to creating the models and hyperparameter tuning proved to be extremely beneficial when seeking to create an optimal D-FFNN, as evidenced by the work of Syahira Ibrahim and Norhaliza Abdul Wahab[2]. This iterative approach allowed me to effectively record and compare the efficacy of the changes I made to different hyperparameters. This process resulted in a stable and effective D-FFNN that minimised overfitting and underfitting. This model is denoted as 'Model C' and will be expanded upon in Section 3.01.

2.0.2 Data Preprocessing

The dataset consisted of a blend of personal details and transactional details pertaining to credit card holders, featuring both categorical and continuous attributes. There were 24 features including the feature 'default payment next month' which was selected as the target variable. The feature's value was either a one or a zero. One indicated a default and Zero indicated that there was no default. The other features were all used as input features.

The dataset was loaded into a pandas Dataframe which allows for flexible and versatile data manipulation.

The target variable was then separated from the input features. This separation allowed for the data to be normalised. The input features were scaled with the use of the StandardScaler from the Sklearn python library. Scaling allowed for a consistent performance of the optimisation algorithm. Scaling input features also enhanced the model's ability to generalise better to unseen data. [3]

Subsequently, I split the data into distinct training, validation and testing sets. To achieve these splits, the dataset underwent an initial division where 60% of the data was allocated to the training set, with the remaining part temporarily held back. This reserved portion was subsequently halved, evenly distributing the data into validation and test sets. This arrangement ensures that the model is exposed to a variety of data points during training while retaining a sufficient quantity of unseen observations for validation and final evaluation.

The process incorporates a mechanism to maintain consistency across experimental runs, achieved through the setting of a predetermined seed in the random number generator. This approach, referred to as fixing the "random state," ensures that the splits are reproducible, allowing for the validation of experimental results and the comparison of model performances under identical data partitioning conditions.

There was a class imbalance in the given dataset so I employed a stratification technique to ensure that each split reflects the original dataset's class distribution. This preserved the statistical properties of the data across all subsets.

3 Experimental Results

The evaluation process was an iterative approach at creating an optimal model based on a model's results. I will evaluate the performance of three of my models. The three different models are denoted as Model A, Model B and Model C.

3.0.1 Evaluation of Model A

Model A was my first attempt at building a D-FFNN. I created a straightforward model that would serve as a baseline for further iterations. These further iterations are Model B and Model C which seek to improve upon Model A's results as shown in Table 2.

The designed architecture included an input layer with 128 neurons to process features, and two hidden layers with 64 neurons to learn non-linear relationships, and a singular neuron in the output layer for the binary classification task.

The Rectified Linear Unit (ReLU) function used for activation was chosen to circumvent vanishing gradients that can occur with the use of other activation functions such as sigmoid functions as mentioned by Tina Jacob in a recent article. [4]

I selected the Adam optimizer for training due to its well-known adaptability and efficiency. Starting with a learning rate of 0.01. This learning rate was employed with the desire of balancing convergence speed and stability. [5]

Model A's obtained results as shown in Figure 1 and Table 2 were promising due to a sustained decrease in test and validation loss and increase in test and validation accuracy but I believed that Model A could be refined.

Model A's initial loss was abnormally large and the model's resultant training loss was greater than its validation accuracy. Model A's validation accuracy shows a high variance which was disconcerting. These results lead me to believe that the model was exhibiting signs of overfitting due to a high learning rate of 0.01 which was overly aggressive. The complexity of the model could also be reduced to help prevent overfitting.

3.0.2 Evaluation of Model B

When building Model B, I refined my approach by simplifying the architecture. I opted for an input layer with 64 neurons and a single hidden layer with 32 neurons. I also adjusted the learning rate to a more conservative 0.001, as recommended by best practices and empirical research.[6]

The choice to use the Adam optimizer remained, given its proven efficiency and robustness in various scenarios.

This learning rate adjustment aimed to curb the abnormally high initial loss and overfitting observed in Model A by facilitating more gradual and stable learning progress. These alterations in the model's complexity and training regimen were crucial in enhancing the generalisation ability of the neural network.

As seen in Figure 5, the training and validation loss improved slightly and the training and validation accuracy improved. Figure 2 shows a decrease in training loss over time, which suggests that the model is learning and improving its prediction on the training data.

However, the validation loss does not show a significant improvement, which may suggest that the model is not generalising well to unseen data. Similarly, the training accuracy increases, indicating better performance on the training set, but the validation accuracy remains relatively flat. This discrepancy shows that I did not successfully alleviate Model A's problem of overfitting.

3.0.3 Evaluation of Model C

When finally building Model C, my main goal was to prevent overfitting therefore my refined approach was the incorporation of L2 regularisation, encouraging the model to learn more robust features that generalise better to new data.

The lambda value was set to 0.01 based on that value's common use as a benchmark across many neural networks used in various contexts. Further aiding my attempt to prevent overfitting, dropout layers were introduced, imposing a rate of 0.3 to randomly omit a subset of neurons during training. [7]

3.1 Evaluation of All Three Models

As seen in Table 2, Model C outperformed Model A in terms of test loss and validation loss in terms of test accuracy and validation accuracy.Model C was slightly outperformed by Model B in those same metrics, however Figure 3 shows Model C's smoother loss and accuracy curves, reflecting a model that generalises well to validation data. Conversely, Model B and Model A, which lack these regularisation mechanisms, exhibit more fluctuation in validation metrics, suggesting potential overfitting to the training data. The stabilisation in Model A's metrics underscores the efficacy of regularisation in enhancing model robustness and reliability.

This result of the addition of dropout layers results in a slight increase in training loss but allowed the model to generalise to unseen data better than Model B. Hence the validation loss and accuracy tend to follow the training loss and accuracy more closely when compared to Model B's graphs.

The implementation of dropout layers in Model C's architecture cures the wild swings shown by Model A's resultant graphs as seen in Figure 1 and the disparity between validation and training loss and accuracy shown in Model B's graphs as seen in Figure 2.

3.2 Experimental Results

Table 1: Table Expressing Different Hyperparameter Settings Used For Each Model

Model	**Epochs**	**Batch Size**	**Number of Hidden Layers**	**Optimiser**	**Optimiser Learning Rate**	**Dropout Used (Value If Used)**	**L2 Regularisation Used**	**Early Stopping**
Model A	50	64	2	Adam	0.01	No	No	No
Model B	50	32	1	Adam	0.001	No	No	Yes
Model C	50	32	1	Adam	0.001	Yes (0.01)	Yes (L2=0.01)	Yes

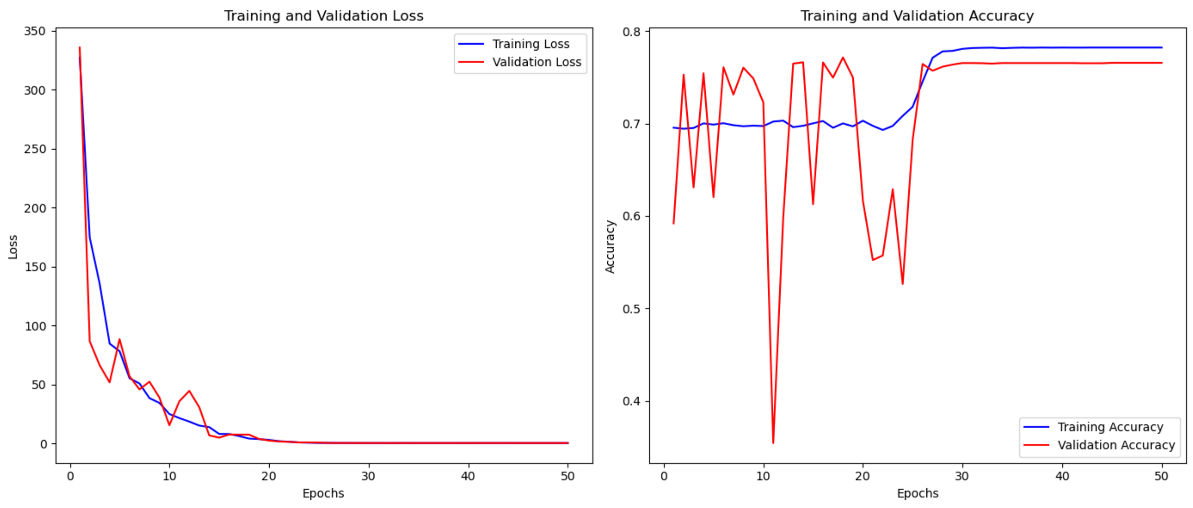

Figure 1: Model A's Loss and Accuracy Graphs

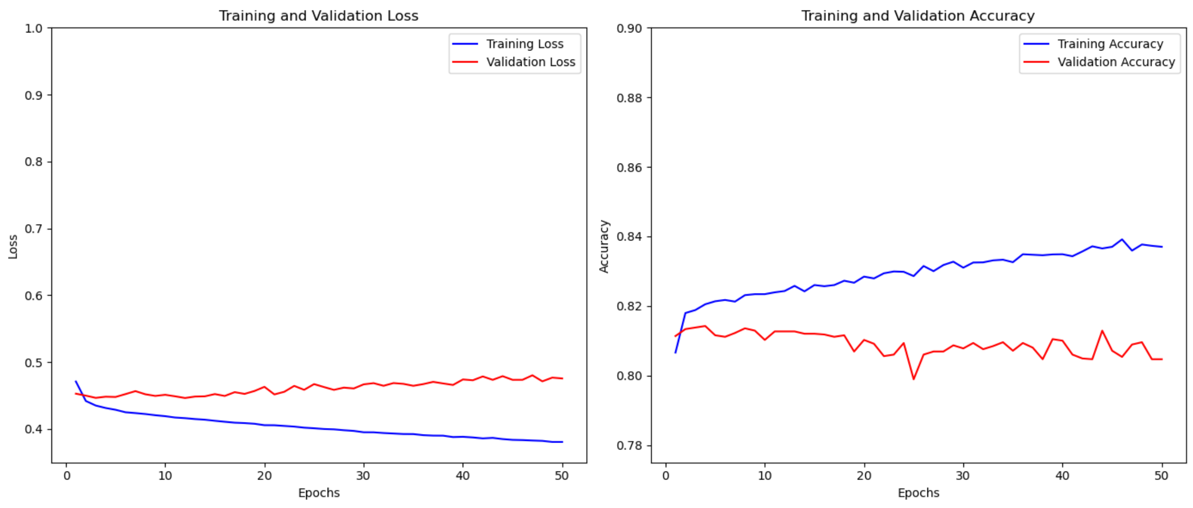

Figure 2: Model B's Loss and Accuracy Graphs

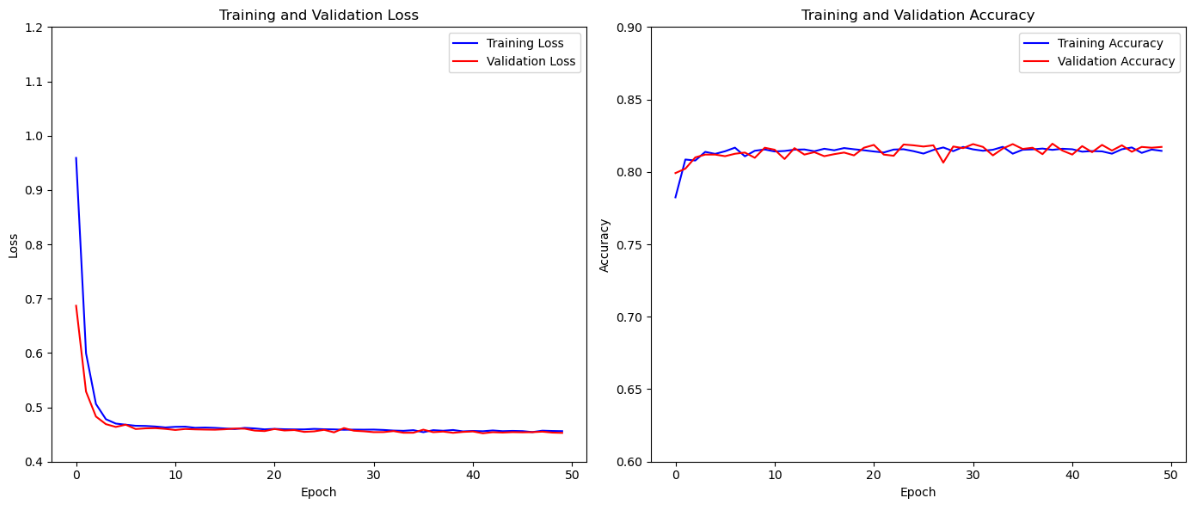

Figure 3: Model C's Loss and Accuracy Graphs

Table 2: Final Loss and Accuracy Results of Each Model

Model	**Test Loss**	**Validation Loss**	**Test Accuracy**	**Validation Accuracy**
Model A	0.4573	0.4752	0.7812	0.8091
Model B	0.3861	0.4861	0.8358	0.8389
Model C	0.4485	0.4483	0.8213	0.8220

4 Summary

When attempting to accurately and efficiently predict credit card defaults, a Deep Feed-forward Neural Network (D-FFNN) with a simpler architecture proved to outperform more complex models. The selected model featured a singular hidden layer that uses dropout and other regularisation techniques. This architectural simplicity in tandem with an iterative hyperparameter tuning process allowed me to create an optimal model. This model is 'Model C' which outperformed my other models in terms of accuracy, consistency and generalisation to unseen data. I believe that Model C is a D-FFNN that effectively predicts credit card defaults.

5 References

References

[1] Frank Emmert-Streib, Zhen Yang, Han Feng, Shailesh Tripathi, and Matthias Dehmer. An introductory review of deep learning for prediction models with big data. *Frontiers in Artificial Intelligence*, 3:4, 2020.

[2] Syahira Ibrahim and Norhaliza Abdul Wahab. Improved artificial neural network training based on response surface methodology for membrane flux prediction. *Membranes*, 12(8):726, 2022.

[3] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, Inc., 2019.

[4] Tina Jacob. Vanishing gradient problem: Causes, consequences, and solutions. *KDnuggets*, June 2023. Available online at: https://www.kdnuggets.com/2022/02/vanishing-gradient-problem.html.

[5] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[6] Yanzhao Wu and Ling Liu. Selecting and composing learning rate policies for deep neural networks. *arXiv: Learning*, 2022.

[7] Francois Chollet et al. Keras 3 api documentation: Layer weight regularizers. `https://keras.io/api/layers/regularizers/`, 2021.