Deep Learning for Predicting Credit Card Defaults

Henry Forrest

March, 2024

Abstract

This report details the development and evaluation of an Artificial Neural Network (ANN) model for predicting credit card defaults utilizing a given dataset. The model architecture features a feed-forward neural network with two dense layers and dropout layers. Early stopping was implemented to prevent over-fitting, optimizing the model's generalization performance. Experimentation was conducted to fine-tune hyper-parameters such as activation functions, batch size, and optimizer selection. The findings highlight the effectiveness of ReLU activation, a batch size of 32, and the Adam optimizer in achieving an impressive accuracy of over 82%. Overall, this study demonstrates the efficacy of ANN models in real-world prediction tasks, emphasizing their versatility and applicability.

1 Introduction

The primary objective of this report is to explore the creation process of a deep learning model for predicting credit card defaults from a given dataset. The task involves predicting the target variable 'Y' based on a set of input features including gender, age, spending patterns of individuals and more. I have decided to use an Artificial Neural Network for this project as they are best for solving complex problems on large datasets. The report outlines the pre-processing steps, model architecture, hyper-parameter tuning, and evaluates the model's performance on a testing set. The findings of this study contribute insights into the effectiveness of the proposed deep learning method for the given task.

2 Proposed Method

Following is the final model that I found to give the best results based on loss score, test accuracy graphs representing model training patterns and more. Additionally a confusion matrix was used to breakdown to model accuracy to true positives and true negatives. The hyper-parameters that have been used were carefully selected through a meticulous testing phase.

2.1 Data Loading and Preprocessing:

The dataset is loaded from an Excel file using the Pandas library[Pan24]. The first bit of pre-processing that was necessary for this model was to drop the first row of data as there seemed to be two titles for each column. I also decided to change the values of the gender column from 1 for a male and 2 for a female to 0 for a male and 1 for a female. This will help the AI model as now the field looks like a binary value instead of 2 possibly being more important or better than 1. This was done using the map function from the pandas library, which allows you to easily change (or map) the 2 values to 1 and 1 values to 0. The columns from X12 to X23 have been summed and replaced with one column which is the total money paid minus spent in the last six months. This should simplify the data for the model. Data for the columns Education and Marriage have been one hot encoded using the pandas function get dummies. This involves turning one column with many possible values into many columns all with only two possible values true or false. As well as this only one of the new columns can be true for any one individual. Which will assist the model as it ensures numerical compatibility, prevents bias, and contributes to the overall performance and interpretability of the model[Bro17]. Finally features ('X') and the target variable ('Y') are extracted from the dataset. Rows with index 0 are removed from both features and the target variable. Data types are converted to 'float32' for compatibility with the deep learning model. This is done using a pandas library routine.

2.2 Data Splitting and Standardization:

The dataset is split into training and testing sets using the train_test_split function from scikit-learn[sl19]. A random seed of 42 ensures reproducibility. Standardization is applied to the features using the StandardScaler to ensure consistent scaling across the dataset.

2.3 Model Architecture:

A feed forward neural network (FNN) has been used for this model. This is an artificial neural network (ANN) in which nodes do not form loops and each node is connected to every node in the next layer. It is also one of the more simple neural networks as information only flows in one direction from the input layer to the output layer[Tur]. This structure design makes a FNN ideal for this project as there is no need for a more complex model such as a recurrent neural network (RNN) which uses loops for information to pass through layers multiple times[Kal22]. A Sequential model is constructed using the Keras library. The model consists of two dense layers, each followed by a dropout layer. Dropout layers are used to prevent over-fitting and allow the model to perform better on unseen data. A dropout rate of 0.5 has been used [SHKS14]. The input layer has 256 units and the Rectified Linear Unit (ReLU) activation function, while the output layer has 1 unit and sigmoid activation for binary classification.

2.4 Early Stopping Implementation:

Early stopping is employed using the EarlyStopping callback from Keras[Ten19]. Monitoring 'val_loss', the training process halts if no improvement in the model is observed for 5 consecutive epochs. The model's best weights are then restored to prevent over-fitting. This means that there is no need to worry about setting the correct number of epochs as the model will stop at its best version.

2.5 Model Compilation and Training:

The model is compiled using the Adam optimizer and binary cross-entropy loss function. Due to the Adam optimizer having an adaptive learning rate I have decided not to change the default starting learning rate of 0.001 as it is shown that the Adam optimizer often performs best with this learning rate[Vis23]. Training is performed on the training set with 200 epochs, a batch size of 32, and a validation split of 0.2. I have set the epochs at 200 because of the early stopping, this ensures that the model runs for as many epochs as it needs until it starts to deteriorate and then the best weightings are restored.

3 Experimental Results

During the creation process of this model a lot of experimentation took place to find the correct hyper-parameters for the model. These were the ones which gave the model the best accuracy while still having minimal loss. True positive predication rate was also taken into account when deciding on the best hyper-parameters for the model as it is very important that the model predicts positive cases more often than not. The use of graphs was essential in determining which models had better training and validation curves and the best loss score.

3.1 Activation function

The first thing that was experimented with was the activation function of the hidden layer. I tested the model with the activation functions ReLu, Tanh and Leaky ReLu. I found that the best results came when using the ReLu activation function. Although the results where not too different ReLu still performed better then the other activation functions with higher accuracy and lower loss score.

3.2 Batch size

I also found through research that a batch size of 32 was is a good starting point for most models. This article shows how the batch size of 32 is ideal as it has the lowest test error rate and also does

not take a very long time like a batch size of 8 does[Tha20]. As well as this many different batch sizes were tried and tested and a batch size of 32 proved to be the most reliant.

3.3 Test Size

While experimenting with the test size I found that a lower value was better for this model as the model easily gets over trained otherwise. This can be shown in figure 1 and figure 2. It can be seen that the validation curve is a lot closer to the training curve in figure 1 where the test size is lower (test size = 0.2 as used in final model). Even though the test accuracy is only marginally better in figure 1 the graph shows a far better model and learning curve. Additionally in figure 2 the validation accuracy is a lot less consistent than in figure 1 as it jumps up and down between each epoch, this is not desirable for a good model.

3.4 Optimizer

The Adam optimizer is know to be better with handling larger datasets as opposed to the Stochastic gradient descent(SGD) optimizer. This means it takes less epochs to learn train the model as it learns much faster. This can be seen in figures 1 and 3. Figure 1 shows the training and validation accuracy for a model using the Adam optimizer and 3 using the SGD optimizer. Please note the different scales on the y-axis in both figures. This shows how the Adam optimizer reaches a slightly higher final test accuracy than the SGD optimizer and does so in less epochs. This is desirable as it takes less time to train the model.

3.5 Oversampling

Due to the dataset having a class imbalance because a strong majority of people do not default on their payments I thought it might be a good idea to implement Synthetic Minority Oversampling Technique (SMOTE) oversampling. SMOTE is a technique specifically designed to tackle this issue by oversampling the minority class, creating synthetic samples to balance the class distribution. This helps as the model can not get an increased accuracy anymore by just guessing the majority class as it could before. Although applying the SMOTE oversampling reduces the models test accuracy by a bit I think it can be looked over because it increases the percentage of true positives (correctly predicted defaults). This can be seen in figures 4 and 5 which are two confusion matrices for the model with and without the SMOTE over-sampler. What can be seen is a far higher true positive rate when SMOTE is used. I think that this is very important as it matters more if someone is going to default rather than if they aren't. Therefore getting a few more false negatives is not an issue when you increase the percentage of true positives by this much. This is because with the real world applications of this project default instances would be focused on more than non-default ones.

4 Summary

In conclusion, an Artificial Neural Network model was successfully developed for credit card default prediction, achieving an accuracy exceeding 82%. Key findings include the significance of hyper-parameter tuning, with notable parameters including the ReLU activation function, a batch size of 32, and the Adam optimizer. Additionally, early stopping proved instrumental in enhancing model efficiency. This report underscores the utility of ANN models in real-world prediction tasks, highlighting their efficacy and versatility.

References

[Bro17] Jason Brownlee. Why one-hot encode data in machine learning?, Jul 2017.

[Kal22] Debasish Kalita. A brief overview of recurrent neural networks (rnn), Mar 2022.

[Pan24] Feb 2024.

[SHKS14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[sl19] scikit learn. scikit-learn: machine learning in python, 2019.

[Ten19] TensorFlow. Tensorflow, 2019.

[Tha20] Ayush Thakur. What's the optimal batch size to train a neural network?, Aug 2020.

[Tur] Turing. Understanding feed forward neural networks in deep learning.

[Vis23] Neha Vishwakarma. What is adam optimizer?, Sep 2023.

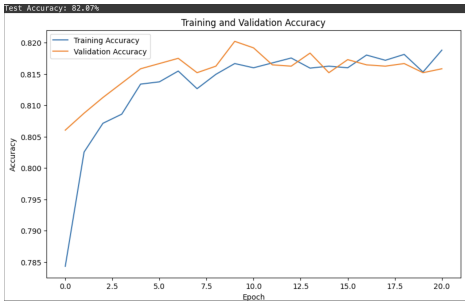

Figure 1: Final Model

Figure 2: Test Size = 0.5

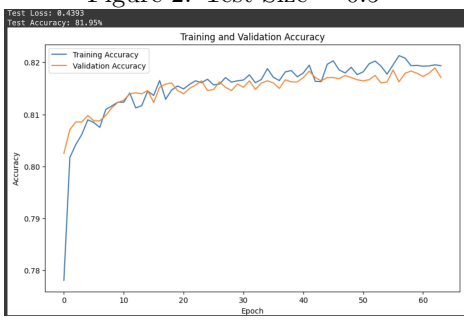

Figure 3: SGD optimizer

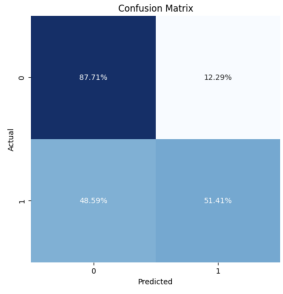

Figure 4: Using SMOTE oversampling

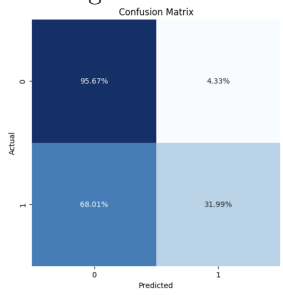

Figure 5: Without SMOTE oversampling