



## **CSC3034 Computational Intelligence**

### **Assignment 2**

by

Saurabh Varughese M. Kovoor (20017604)

Chea Wern Yen (18111195)

21st November, 2022

## **Abstract**

This paper presents the comparative study performed on the use of artificial neural networks (ANN) or multilayer perceptron (MLP) model to predict or classify an applicant's suitability to be approved as a credit card-holder based on their input data including past credit history and application history. The NN model was implemented in this project with the help of the MLPClassifier object of the scikit-learn library. This paper presents the tests that were performed and a discussion or comparison of the resultant evaluation metric results to identify the optimal parameter values for this NN model that maximises its performance metric results when applied for this problem domain and to the UCI credit card approval dataset.

Keywords — Artificial Neural Networks (ANN), Multi-layer Perceptron (MLP), Hybrid Intelligence Systems, Machine Learning (ML)

## **Table of Contents**

<b>1</b>	<b>Introduction</b>	<b>4</b>
1.1	Dataset Description	4
<b>2</b>	<b>Construction of Neural Network</b>	<b>8</b>
2.1	Implementation	8
2.2	Parameter Tests	12
<b>3</b>	<b>Modification into an Evolutionary Neural Network</b>	<b>24</b>
3.1	Generating an initial population	24
3.2	Crossover and mutation	26
3.3	Fitness evaluation	26
	<b>Conclusion</b>	<b>27</b>
	<b>References</b>	<b>28</b>

# **1 Introduction**

The main focus of this project is classification and prediction using machine learning methods. A subdivision of the more general artificial intelligence (AI) network that makes use of sophisticated algorithms and deep learning neural networks is what is considered machine learning (ML) [1]. Hence, with the aid of machine learning (ML), a program can automatically and iteratively learn through the input of data, improving its performance based on prior knowledge and experiences (heuristics) [2]. As a result, the database's derived knowledge serves as the basis for this training, which enables the model to make predictions and categorical determinations.

Classification is a technique in which ML algorithms are trained to categorise cases for a specific problem set. Therefore, it is a predictive modelling of any problem domain that requires predictions based on labelled classes and a dataset [2]. As a result, it distributes data among predefined classes in accordance with the given rules, generated by the ML model.

The ML algorithm used for this study is the Artificial Neural Networks (ANN) technique, also known as multilayer perceptrons (MLP). The problem domain concerned in this project is the banking sector, which deals with numerous data that is influential for the purpose of decision-making in determining whether a candidate or applicant is suitable for their credit card program as a borrower. It involves a sequence of rule-based decisions with these input data in order to determine whether a particular credit card application should be approved. On that note, the goal of this project is to use the power of machine learning and neural networks to process the inputs, review each individual credit card application and determine whether or not it will be approved. Thus, the designed NN models utilise applicants' past credit history and application history to accurately predict and classify their suitability as a borrower.

## **1.1 Dataset Description**

The dataset chosen to construct and analyse the performance of the neural network or multilayer perceptron algorithm is the credit card approval dataset from the University of California Irvine (UCI) machine learning repository [4]. This is a partially cleaned dataset, as after analysis of the dataset it is found that it does not contain missing values and have been filled and feature names are provided, allowing for more context and easier use. Hence, the dataset consists of 690

rows/records and 16 attributes or features, discovered using the *df.shape()* function, a Pandas DataFrame method.

Hence, the following table shows the 16 attributes that are part of the final creditcard.csv file that was loaded into the Python program (using the Pandas method, *pd.read\_csv()*) and further ML processing was performed on:

*Table 1: Dataset description*

Attribute name	Feature Values	Description	Data Type
Gender	0: Female 1: Male	Sex/gender of the applicant	Numerical, int
Age	13.8 - 80.3	Age of the applicant in years	Numerical, float
Debt	0-28	Outstanding debt of an applicant	
Married	0: Single/ Divorced/ etc 1: Married	Marital status of applicant	Numerical, int
BankCustomer	0: Does not have a bank account 1: Has a bank account	Denotes whether the applicant currently has an account with the bank.	Numerical, int
Industry	Industrials, Energy, Materials, CommunicationService es, Transport, Financials, Real Estate, Utilities, Education, Healthcare, ConsumerDiscretionar y, ConsumerStaples,	The applicant's current job sector or most recent job	Categorical, object

	InformationTechnology, Research		
Ethnicity	White, Black, Asian, Latino, Other	The applicant's ethnicity or race	Categorical, object
YearsEmployed	0 - 28.5	The number of years the applicant has been employed in their place of work	Numerical, float
PriorDefault	0: no prior defaults 1: prior default	Denotes whether the applicant has prior defaults on any loans	Numerical, int
Employed	0: not employed 1: employed	Denotes whether the applicant is currently employed	Numerical, int
CreditScore	0 - 67	The applicant's credit score	Numerical, int
DriversLicense	0: no licence 1: has licence	Denotes whether the applicant has a drivers licence	Numerical, int
Citizen	ByBirth, ByOtherMeans	Denotes the method in which the applicant obtained citizenship	Categorical, object
ZipCode	0 - 2000	The applicants zip code	Numerical, int
Income	0 - 100.0k	The applicant's monthly income	Numerical, float
Approved (Target)	0: not approved 1: approved	The predicted target attribute. Denotes whether the applicant was approved or not approved for credit card.	Numerical, int

This learning by the NN models is considered to be "supervised" as opposed to "unsupervised" learning, where instances are unlabeled, because the dataset we are using has clear known labels where it has the corresponding correct outputs for the "Approved" attribute, which is the target attribute or the values that are being predicted. When the correct outputs are labelled, this type of ML is very robust, applying to many conditions and scenarios, and very successful for some problem domains.

## **2 Construction of Neural Network**

Neural networks, in general, are a collection of algorithms that imitate the way the human brain thinks in order to find or match hidden relationships in a dataset [2]. Input, output, and hidden layers are the three layers that make up a neural network, and each layer's nodes are connected to the others. This hidden layer typically consists of components that can change the input data into a pattern that the output layer can change. Neurons, which are essentially mathematical functions in neural networks, collect and categorise data according to the particular architecture. NNs can have one or more layers of perceptrons or neurons, hence the name “multilayer perceptron”. These perceptrons transmit the signals from the multiple linear regression to a potentially non-linear activation function. In a multi-layered perceptron, these perceptrons are arranged in a network of interconnected layers, with the input layer gathering input patterns and the output layer holding output signals or classifications that input patterns may map to.

They have become an especially important component of AI since the development of a new technique called backpropagation, which allows networks to modify their hidden layers of neurons in situations where the output does not match the developer's expectations. Backpropagation is utilised in feedforward NNs to optimise the weights and threshold values at the neurons of an NN iterating backward from the last layer, hence using the calculated output, and finding the difference with the expected output to obtain the error values to optimise those two parameters.

### **2.1 Implementation**

In order to implement the ANN or MLP model in this program, the MLPClassifier object of the scikit-learn neural\_network library was utilised. The MLPClassifier object uses a feedforward neural network with three layers—the input, output, and hidden layer—and is made up of nodes that are connected to one another. Akin to a directed graph, each node/perceptron of an NN has only one possible signal path, an MLP connects multiple layers. Every node, with the exception of the input nodes, has a nonlinear activation function.

Data is first received by the input layer, after which it is abstracted in the hidden layer. The output layer is where the predictions, classifications, or outputs are distributed. Each input layer is subjected to an activation function and initial weights in a weighted sum, just like a typical perceptron unit would be. Similar to the coefficients in a regression equation, these weights are



used. A simple mapping of the neuron's output to its summed weighted input is known as an activation function or transfer function. It is referred to as an activation function because it regulates both the intensity of the output signal and the threshold at which the neuron is activated.

However, as mentioned earlier backpropagation is a supervised learning technique that an MLP uses to differentiate itself from other supervised learning approaches. The neural network can iteratively alter its network weights using backpropagation to lower its cost function. An MLP is viewed as a deep learning method because it contains multiple layers. Neural networks' capacity for prediction is a result of this hierarchical or multi-layered structure.

### **Data Preprocessing**

The program starts by preprocessing the dataset to remove faults or errors present in the raw data that might affect the performance and effectiveness of the designed NN models. For this, first, outliers are removed from the numerical attributes of the dataset, using the *outlier()* user-defined function to ensure the values of an attribute are within the interquartile range and the rest are removed. This was seen to have a generally positive improvement on the NN model performance.

Following that, numerical encoding of the categorical attributes of the dataset were performed using the `LabelEncode()` function of the scikit-learn preprocessing library. This converts the string-values of the categorical attributes into numerical values ranging from 0 to the number of classes in that particular feature.

Subsequently, the data is partitioned into 2 sections, the target attribute, *y* which consists of the *num* attribute and the input attribute, *x*, which consists of the other input attributes in the dataset.

In addition, the categorical attributes of the data are one-hot encoded using the pandas method `get_dummies`. It converts the categorical attributes into a number of new columns, based on the number of classes, and assigns a binary value of 1 (True) or 0 (False) to the new column. Therefore, no ordinal relationship exists in between the newly formed attributes or classes within a parent attribute which typically results in the ML models performing better as they can understand and interpret the input attributes more correctly and accurately.

Imputation or replacement of missing values is not performed as it is found that there are no missing values in the dataset.

The Synthetic Minority Oversampling Technique (SMOTE) method of the imblearn oversampling library is utilised for oversampling the dataset which is a data augmentation algorithm, which in short balances the dataset by slightly moving the data point closer to its neighbour, following the k-nearest neighbour method.

Scaling is then performed on the dataset using the StandardScaler method of the scikit-learn preprocessing library. This operation essentially standardised or normalised the numerical attributes of the dataset that each follow a different range, before being used before fitting with the ML models.

### **Data Splitting**

The prepared dataset must be divided into two parts: the training set, which is used to create the predictive models, and the validation/testing set, which is used to judge or test whether the developed model can be applied generally or whether it can correctly categorise inputs. The data split ratio used in this program is 80:20, meaning that 80% of the data is randomly chosen and used to train the machine learning models, and the remaining 20% is used to evaluate the ML models.

This data partitioning technique was implemented using the *train\_test\_split()* function of the scikit-learn library, which will return a randomly assigned training and testing subset for the x and y datasets to be inserted into the ML model or for validating the models (total 4 returned datasets), based on the entered transformed x and y datasets, according to the testing dataset size (total 4 returned datasets). Inputting 0.2 as the *test\_size* will result in an 80:20 split between the training and test datasets.

### **Model Creation**

In the program, the instantiation of this MLPClassifier object, with the parameter and its values is done in the user-defined function, *nn\_pipeline()*. Therefore, this function is called repeatedly by the test case functions to test the different parameter values on the model's performance. Therefore, after instantiation, the *fit()* function is used to train the ML model, following which, the *predict()* function is employed with the testing set to observe the performance of the models.

Since we already know whether a particular applicant has been approved or not, we can use the data that is currently available to train our prediction model. This procedure is also referred to as learning and supervision. Then, the trained model is applied to predict whether the applicant's credit card application is approved or rejected. The *time.time()* method is used to keep track of the length of time required to train and test the model.

### **Model Validation**

After the model is created, its performance is measured using the user defined function, *modelValidation()*, collects the value of the y or target-attribute, the predicted y values or target-attribute (*model.predict()* result), and the tracked time. This function includes a number of ML model evaluation tests from the scikit-learn metrics library, such as `classification_report`, `accuracy_score`, `precision_score`, `recall_score`, `f1_score`, `mean_squared_error`, `mean_absolute_error`, and user-defined methods to determine the root mean square error, misclassification rate, the training, testing, and total time.

The scikit-learn library's `MLPClassifier` object makes it simple to change important parameters or hyperparameters and observe how doing so affects the performance and evaluation metric outcomes of the resulting model. Therefore, the three tests were designed, which altered the values of each of these `MLPClassifier` object parameters. The following sections describes how the three tests were conducted, the possible values of the four parameters that were used, and the results of this testing procedure:

## 2.2 Parameter Tests

### Test 1: Activation Function and Solver

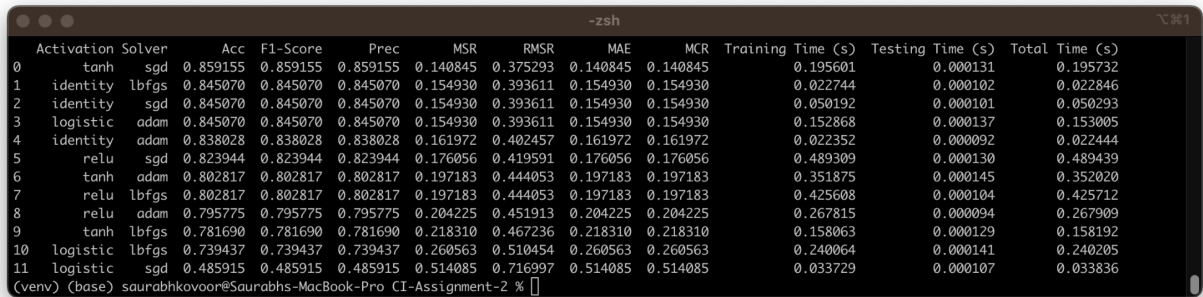
The tested values for the activation function parameter include identity('identity'), logistic sigmoid ('logistic'), hyperbolic tan function ('tanh'), and rectified linear unit ('relu'). This parameter describes how each hidden layer neuron's activation function works. In this case, the "identity" value is a no-op activation technique that successfully implements a linear bottleneck by returning a linear function based on  $f(x) = x$ . The "logistic" value, on the other hand, makes use of a logistic sigmoid function, which is based on a function that returns  $f(x) = 1/(1+\exp(-x))$ . The "tanh" value is a hyperbolic tan function that, as its name implies, returns based on the equation  $f(x) = \tanh(x)$ . The "relu" value, in addition, specifies a rectified linear unit function that is based on the function  $f(x) = \max(0, x)$  in order to return results.

The tested values for the solver parameter include Limited-memory BFGS ('lbfgs'), stochastic gradient descent ('sgd'), and stochastic gradient-based optimizer ('adam'). This solver parameter designates the weight optimization solver across the nodes, where "adam" performs well in terms of training time and validation score, particularly on relatively large datasets. In contrast, "lbfgs" performs marginally better for smaller datasets because it converges more quickly.

To begin with, the parameters list for the activation function and solvers is initialised for the first test to specify the values for these parameters. The maximum iterations and hidden layer size are then initialised with default values of 1000 and (32, 16), respectively. With the MLP having 2 layers, 32 neurons in the first layer, and 16 in the second, and training for a maximum of 1000 iterations, there will be a total of 12 combinations to test since there are 4 activations and 3 solvers. By doing this, it is ensured that every combination leads to a solution. Every test run starts with a call to the user-defined method *nn\_pipeline*, which basically starts the construction of the MLPClassifier object along with the parameter values for that specific test run and calls the aforementioned *modelValidation* function to evaluate it. Following the test run, the activation and solver used, as well as their computed evaluation metric results, are stored in an array as key information about each test run. To easily see the parameters and their corresponding performance and evaluation metric results, this multidimensional array is sorted in descending order of those evaluation metric scores, stored in a Pandas DataFrame, and printed.

## Results and Discussion

The best performing combination of hidden layers and neurons, along with the performance and evaluation metrics that correspond to it, are shown in the following Pandas DataFrame visualisation:



```
-zsh
Activation Solver  Acc  F1-Score  Prec  MSR  RMSR  MAE  MCR  Training Time (s)  Testing Time (s)  Total Time (s)
0  tanh  sgd  0.859155  0.859155  0.859155  0.140845  0.375293  0.140845  0.140845  0.195601  0.000131  0.195732
1  identity  lbfgs  0.845070  0.845070  0.845070  0.154930  0.393611  0.154930  0.154930  0.022744  0.000102  0.022846
2  identity  sgd  0.845070  0.845070  0.845070  0.154930  0.393611  0.154930  0.154930  0.050192  0.000101  0.050293
3  logistic  adam  0.845070  0.845070  0.845070  0.154930  0.393611  0.154930  0.154930  0.152868  0.000137  0.153005
4  identity  adam  0.838028  0.838028  0.838028  0.161972  0.402457  0.161972  0.161972  0.022352  0.000092  0.022444
5  relu  sgd  0.823944  0.823944  0.823944  0.176056  0.419591  0.176056  0.176056  0.489309  0.000130  0.489439
6  tanh  adam  0.802817  0.802817  0.802817  0.197183  0.444053  0.197183  0.197183  0.351875  0.000145  0.352020
7  relu  lbfgs  0.802817  0.802817  0.802817  0.197183  0.444053  0.197183  0.197183  0.425608  0.000104  0.425712
8  relu  adam  0.795775  0.795775  0.795775  0.204225  0.451913  0.204225  0.204225  0.267815  0.000094  0.267909
9  tanh  lbfgs  0.781690  0.781690  0.781690  0.218310  0.467236  0.218310  0.218310  0.158063  0.000129  0.158192
10 logistic  lbfgs  0.739437  0.739437  0.739437  0.260563  0.510454  0.260563  0.260563  0.240064  0.000141  0.240205
11 logistic  sgd  0.485915  0.485915  0.485915  0.514085  0.716997  0.514085  0.514085  0.033729  0.000107  0.033836
(venv) (base) saurabhkvoor@Saurabhs-MacBook-Pro CI-Assignment-2 %
```

Figure 1: Visualisation of the evaluation metric results from the testing of different activation function and solver combination with the MLPClassifier.

In the results, it is found that the “tanh” activation function and “sgd” solver combination performed the best, returning the highest accuracy result of 0.859155, an f1-score of 0.859155, and precision score of 0.859155, with the least mean square error (MSE) of 0.140845, root mean square error (RMSE) of 0.375293, mean absolute error (MAE) of 0.140845, and misclassification rate (MCR) of 0.140845. The training time and testing time are also relatively low, indicating a low complexity of the model. Thus it is the combination chosen for the subsequent tests and the final MLPClassifier object that is created, as it is the most stable and best-performing activation function and solver combination in the parameter list, outperforming every other combination. The sgd solver is seen to perform the worst when paired with the logistic activation function, thus it needs to be paired with the tanh activation to achieve the best performance.

## **Test 2: Number of Hidden Layers and Neurons**

First, the default values for the activation function and solver—the "tanh" activation function and "sgd" solver—are instantiated for the second test using the hyperparameter values that performed the best in the first test. After that, as previously mentioned, the maximum iteration is initially set to 1000 to allow the models to converge to a solution. Then MLPClassifier objects with hidden layer sizes ranging from (1) to (30, 30, 30) will be created and tested. So, the maximum number of hidden layers and neurons that can be instantiated is 3 and 30, respectively.

As testing a different number of neurons for each layer would result in a more complex test sequence, it was decided to keep the maximum number of neurons per layer constant for consistency, test simplicity, and reduced computational time. For the MLPClassifier object, this parameter essentially describes the number of layers and neurons in each layer. It is a tuple data type, and each element's index and length refer to the number of hidden layers in the MLPClassifier. Each element's value indicates the number of neurons in that layer.

As a result, there will be a total of 90 test runs using this method. In order to achieve this sequence, a nested for loop is used for the initial loop through the maximum number of hidden layers and a child for loop for the initial loop through the maximum number of neurons. Every test run includes a call to the nn\_pipeline method, which is similar to the previous test and makes it simpler to initialise the MLPClassifier with parameters based on the values that are passed in that test run.

The parameters (no. of hidden layers, no. of neurons per layer, hidden layer tuple) and their evaluation metric scores are stored in an array and arranged in descending order of those scores. In order to easily visualise the parameters and the results of their corresponding performance and evaluation metrics, the array is then stored in a Pandas DataFrame and printed. Additionally, the evaluation metrics for the number of neurons for each hidden layer's iteration are saved at each layer's iteration and can then be plotted using the Matplotlib method. This is done to illustrate how the performance of the model would change if the number of neurons were increased at each hidden layer stage.

## Results and Discussion

The line graph plots that are shown after each hidden layer for loop are as follows:

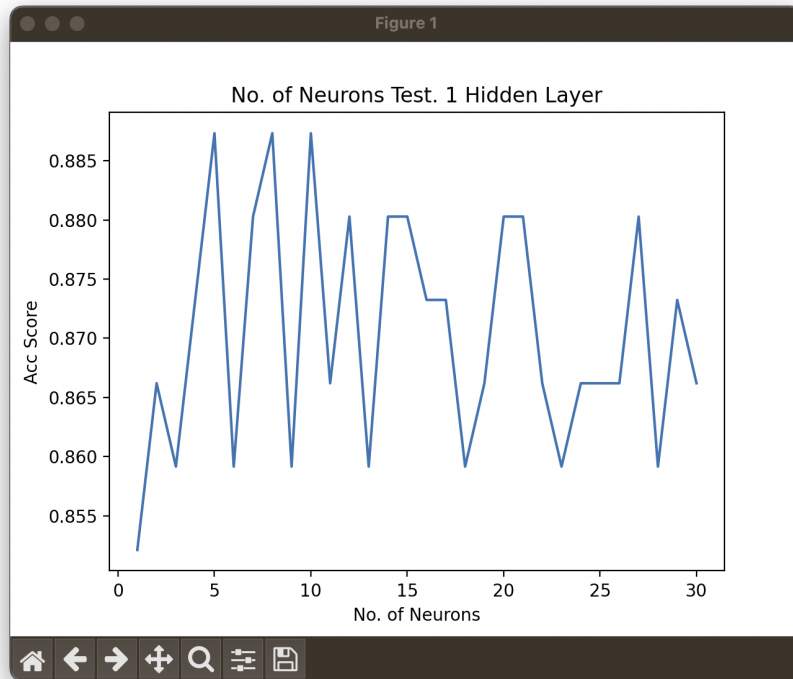


Figure 2: Accuracy score for MLPClassifier model with 1 hidden layer and different number of neurons.

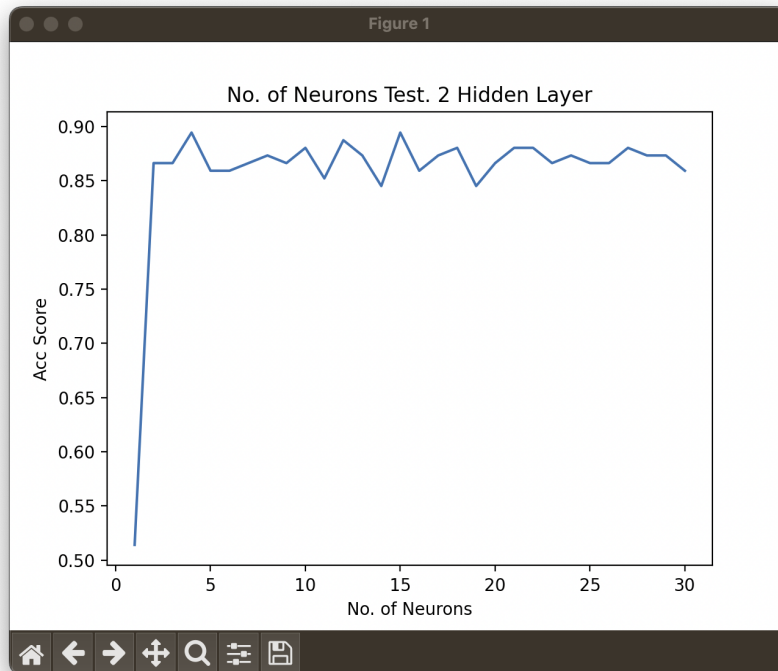


Figure 3: Accuracy score for MLPClassifier model with 1 hidden layer and different number of neurons.

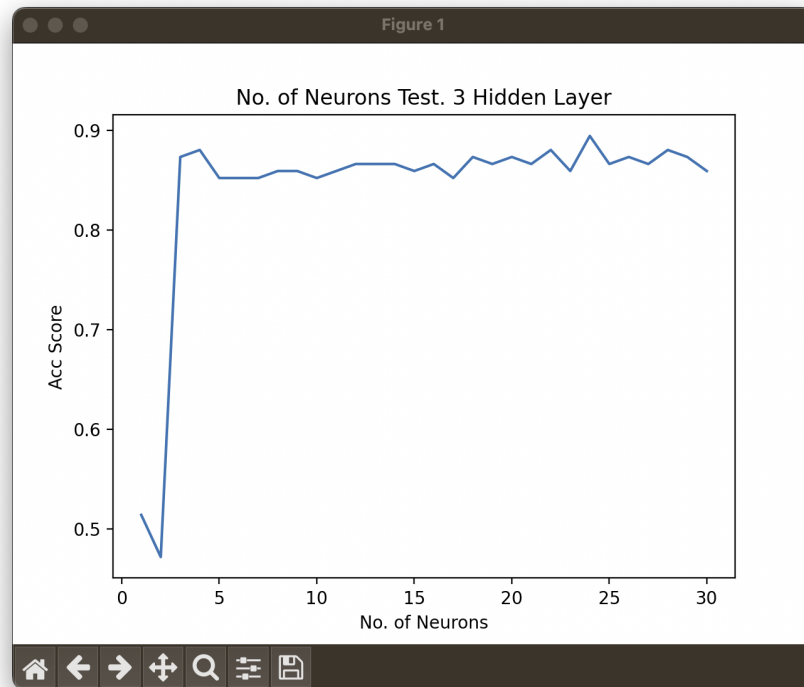


Figure 4: Accuracy score for MLPClassifier model with 1 hidden layer and different number of neurons.

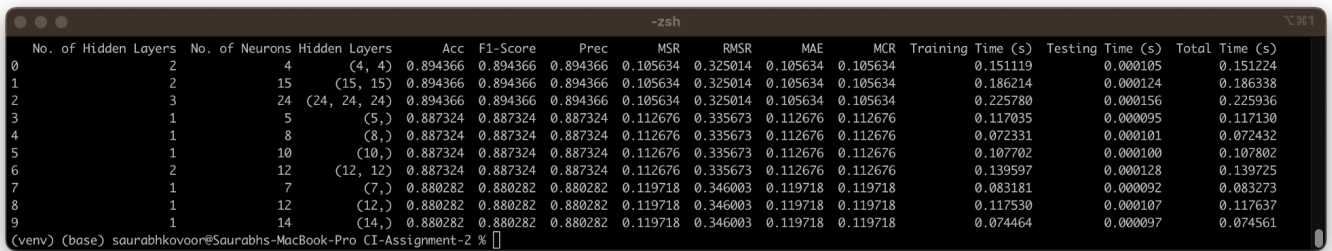
The aforementioned diagrams show how, at each hidden layer, adding more neurons increases the model's accuracy until it stabilises at a mean score of 0.894366, as values above 4 are approached. This shows that the MLP performs better as the number of neurons increases up to the 4 neuron mark. Beyond that, increasing the number of neurons does not improve or deteriorate the model. Therefore, for lesser complexity, 4 neurons are chosen for the number of neurons per layer, as there are lesser neurons or perceptrons to be trained and tested.

Additionally, the accuracy values stabilise at a lower number of neurons as the hidden layer count rises. So, when 1 hidden layer is employed, this exhibits the greatest fluctuations and potentially volatile behaviour in accuracy values when increasing the number of neurons. The accuracy score distribution of when 2 and 3 layers are used are similarly stable, where after the 4 neuron mark the accuracy score stabilises and there is little to no increase or decrease. When the number of neurons is low, the accuracy scores are low as well, and this grows until the 4 neuron mark after which there is no significant fluctuation. Thus, looking at this, 2 hidden layers are preferable for the final chosen hidden layer combination, as it will have lesser complexity



compared to when 3 hidden layers are used, as there will be more neurons and connections, thus more computation with the regression function, hence extending the training and testing time. Therefore, the chosen combination is 2 hidden layers with 4 neurons each, owing to its maximised performance achieved in terms of the evaluation metric results and the low complexity due to the relatively low number of neurons and layers.

The following is the visualisation of the Pandas DataFrame to identify the best performing number of hidden layers and neurons combination and their corresponding performance and evaluation metric results.



	No. of Hidden Layers	No. of Neurons Hidden Layers	Acc	F1-Score	Prec	MSR	RMSR	MAE	MCR	Training Time (s)	Testing Time (s)	Total Time (s)
0	2	4 (4, 4)	0.894366	0.894366	0.894366	0.105634	0.325014	0.105634	0.105634	0.151119	0.000105	0.151224
1	2	15 (15, 15)	0.894366	0.894366	0.894366	0.105634	0.325014	0.105634	0.105634	0.186214	0.000124	0.186338
2	3	24 (24, 24, 24)	0.894366	0.894366	0.894366	0.105634	0.325014	0.105634	0.105634	0.225780	0.000156	0.225936
3	1	5 (5,)	0.887324	0.887324	0.887324	0.112676	0.335673	0.112676	0.112676	0.117035	0.000095	0.117130
4	1	8 (8,)	0.887324	0.887324	0.887324	0.112676	0.335673	0.112676	0.112676	0.072331	0.000101	0.072432
5	1	10 (10,)	0.887324	0.887324	0.887324	0.112676	0.335673	0.112676	0.112676	0.107702	0.000100	0.107802
6	2	12 (12, 12)	0.887324	0.887324	0.887324	0.112676	0.335673	0.112676	0.112676	0.139597	0.000128	0.139725
7	1	7 (7,)	0.880282	0.880282	0.880282	0.119718	0.346003	0.119718	0.119718	0.083181	0.000092	0.083273
8	1	12 (12,)	0.880282	0.880282	0.880282	0.119718	0.346003	0.119718	0.119718	0.117530	0.000107	0.117637
9	1	14 (14,)	0.880282	0.880282	0.880282	0.119718	0.346003	0.119718	0.119718	0.074464	0.000097	0.074561

Figure 5: Visualisation of the evaluation metric results from the testing of different hidden layer combinations with the MLPClassifier

From here, it is found that the (4, 4) hidden layer combination performed the best, returning the highest accuracy result of 0.894366, an f1-score of 0.894366, and precision score of 0.894366, the lowest MSR of 0.105634, RMSR of 0.105634, MAE of 0.105634, MCR of 0.105634, and relatively low training time of 0.151119s, testing time of 0.000105s and a total time of 0.151224s. Thus it is the combination chosen for the subsequent tests and the final MLPClassifier object that is created, as it is the most stable and best-performing hidden layer combination in the parameter list, outperforming every other combination.

### **Test 3: Maximum Number of Iterations (Epochs)**

First, the default values for the activation function, solver, and hidden layer —the "tanh" activation function, "sgd" solver, and (4, 4) hidden layer size—are instantiated for the third test using the hyperparameter values that performed the best in the first and second tests.

The maximum number of iterations is then instantiated to 1000 to test the highest number of training runs, or epochs, which range from 1 to 1000. Consequently, there will be 1000 test runs in total using this method.

To loop through the maximum number of iterations, a for loop is used. At the end of each loop, the `nn_pipeline` is called, and it is used to create an `MLPClassifier` object with the current `max_iteration` parameter.

The parameters (maximum number of iterations) and their evaluation metric scores are stored in an array and arranged in descending order of those scores. In order to easily visualise the parameters and the results of their corresponding performance and evaluation metrics, the array is then stored in a Pandas DataFrame and printed.

## Results and Discussion

In addition, a graph is drawn to show how raising the maximum iteration affects the MLPClassifier model's accuracy. This graph is shown below:

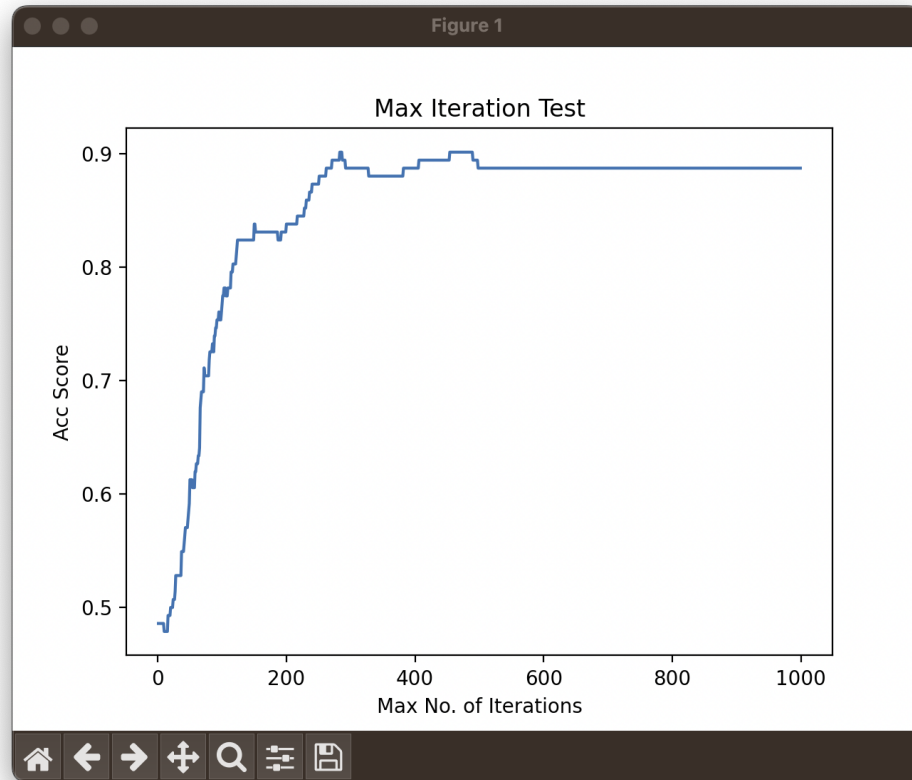


Figure 6: Accuracy score for MLPClassifier model with maximum iterations from 1-1000.

As seen in the aforementioned figure, the accuracy of the NN model increases as the maximum number of iterations is increased, but only until a certain point or the 283 maximum number of iterations mark, after which the accuracy score flattens out or stabilises at 0.901408. This is due to the fact that the NN model is convergent and does not significantly improve beyond an accuracy value of 0.901408, at which point it has had enough training time. Therefore, increasing the maximum iterations will only make the NN model more complex as more time is spent training the model and fitting the NN model over more runs/epochs. However, at too low values of maximum number of iterations, the NN model is not trained or fitted with the dataset sufficiently, hence the accuracy score at these early stages are low. As a result, the lowest maximum iteration value that maximises the accuracy score and other evaluation metric results is found.

For this, the following is the visualisation of the Pandas DataFrame to identify the best-performing number of maximum number of iterations and their corresponding performance and evaluation metric results.



	Maximum No. of Iterations	Acc	F1-Score	Prec	MSR	RMSR	MAE	MCR	Training Time (s)	Testing Time (s)	Total Time (s)
0	283	0.901408	0.901408	0.901408	0.098592	0.313993	0.098592	0.098592	0.077212	0.000063	0.077275
1	284	0.901408	0.901408	0.901408	0.098592	0.313993	0.098592	0.098592	0.077583	0.000064	0.077647
2	285	0.901408	0.901408	0.901408	0.098592	0.313993	0.098592	0.098592	0.078314	0.000094	0.078408
3	286	0.901408	0.901408	0.901408	0.098592	0.313993	0.098592	0.098592	0.080106	0.000098	0.080204
4	454	0.901408	0.901408	0.901408	0.098592	0.313993	0.098592	0.098592	0.124096	0.000087	0.124183
5	455	0.901408	0.901408	0.901408	0.098592	0.313993	0.098592	0.098592	0.124867	0.000075	0.124942
6	456	0.901408	0.901408	0.901408	0.098592	0.313993	0.098592	0.098592	0.127328	0.000169	0.127497
7	457	0.901408	0.901408	0.901408	0.098592	0.313993	0.098592	0.098592	0.129266	0.000100	0.129366
8	458	0.901408	0.901408	0.901408	0.098592	0.313993	0.098592	0.098592	0.126654	0.000096	0.126750
9	459	0.901408	0.901408	0.901408	0.098592	0.313993	0.098592	0.098592	0.130749	0.000120	0.130869

Figure 7: Visualisation of the evaluation metric results from the testing of different number of maximum iterations with the MLPClassifier.

From here, it is found that the 283 maximum number of iterations performed the best, returning the highest accuracy result of 0.901408, an f1-score of 0.815085, and precision score of 0.815085, the lowest MSR of 0.098592, RMSR of 0.313993, MAE of 0.098592, MCR of 0.098592, and the lowest training time of 0.077212s, testing time of 0.000063s and total time of 0.077275s. Thus it is the value chosen for the final MLPClassifier object that is created, as it is the most stable and best-performing maximum iteration value in the parameter list, outperforming every other parameter value, while being the lowest value. Therefore at a low complexity it is able to maximise the performance metric values.

### **Final Test With the Best-Performing Parameter Values**

Conclusively, after obtaining all the optimal values for the aforementioned hyperparameters, it is utilised to build and configure the best performing scikit-learn `MLPClassifier()` object, and observe its performance metric scores. Therefore, as per the results obtained the applied hyperparameters are the following:

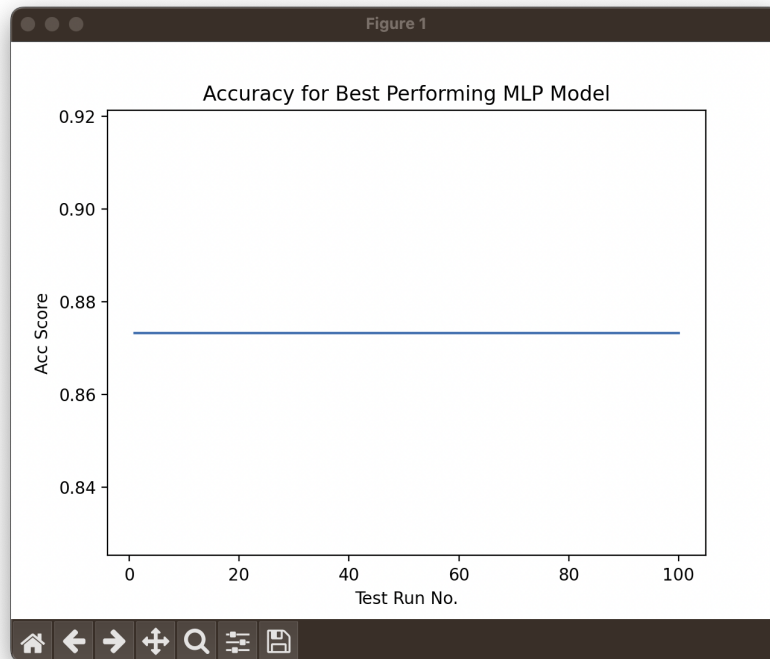
1. Activation function: tanh
2. Solver: sgd
3. Hidden Layer Size (`hidden_layer_sizes`): (4, 4)
4. Maximum Iterations (`max_iteration`): 1000

The purpose of this final test is to iteratively instantiate the `MLPClassifier()` object with the found best and optimal parameter values (as listed above), train/fit it with the dataset, use it to make predictions, and evaluate the performance of the model and observe for any performance fluctuations across the multiple runs or iterations of the designed and implemented neural network model. Thus, the maximum number of test runs is set to 100 to get a good sample of evaluation metric results to compare and observe trends in the distribution of the evaluation metric scores.

Like before, first, the default values for the activation function, solver, and hidden layer sizes are initiated, and maximum iterations using the best-performing hyperparameter values found from the previous test. Then, using a for loop the process of fitting, predicting, and evaluating is performed repeatedly for 100 test runs. During this, their evaluation metric scores are stored in an array and sorted in descending order of their evaluation metric scores (accuracy, f1-score and precision). After that, the array is stored in a Pandas DataFrame to visualise the parameters and their corresponding performance and evaluation metric results easily, which is then printed.

## Results and Discussion

The mean accuracy score is calculated by summing the total accuracies and averaging it with the number of successful runs. This mean accuracy score is found to be 0.873239. After that, a line graph is plotted of the accuracy scores for each test run as seen below:



*Figure 8: Accuracy score for MLPClassifier model with best performing parameter values across multiple test runs.*

From the aforementioned figure, the accuracy of the NN model is seen to remain constant or stabilise at the accuracy value of 0.873239 over all 100 test runs with no fluctuation at all. This indicates that the model performs consistently with the passed in parameter values with no indication of bias.

The following is the visualisation of the Pandas DataFrame to have a more detailed observation of the specific values for the evaluation metric results to observe any deviations or fluctuations in the results:

Test Run	Acc	F1-Score	Prec	MSR	RMSR	MAE	MCR	Training Time (s)	Testing Time (s)	Total Time (s)
0	0.873239	0.873239	0.873239	0.126761	0.356034	0.126761	0.126761	0.098517	0.000139	0.098656
1	0.873239	0.873239	0.873239	0.126761	0.356034	0.126761	0.126761	0.078344	0.000079	0.078423
2	0.873239	0.873239	0.873239	0.126761	0.356034	0.126761	0.126761	0.077660	0.000077	0.077737
3	0.873239	0.873239	0.873239	0.126761	0.356034	0.126761	0.126761	0.077721	0.000079	0.077800
4	0.873239	0.873239	0.873239	0.126761	0.356034	0.126761	0.126761	0.077556	0.000084	0.077640
5	0.873239	0.873239	0.873239	0.126761	0.356034	0.126761	0.126761	0.077832	0.000099	0.077931
6	0.873239	0.873239	0.873239	0.126761	0.356034	0.126761	0.126761	0.077731	0.000084	0.077815
7	0.873239	0.873239	0.873239	0.126761	0.356034	0.126761	0.126761	0.077596	0.000091	0.077687
8	0.873239	0.873239	0.873239	0.126761	0.356034	0.126761	0.126761	0.077602	0.000085	0.077687
9	0.873239	0.873239	0.873239	0.126761	0.356034	0.126761	0.126761	0.077455	0.000084	0.077539

Figure 9: Visualisation of the evaluation metric results from the testing of the MLPClassifier model with best performing parameter values across multiple test runs.

From here, it is found that these chosen parameter values are found to be optimal as these combination of values maximise the performance metric results achieving accuracy result of 0.873239, an fl-score of 0.873239, and precision score of 0.873239, and the lowest MSR of 0.126761, RMSR of 0.356034, MAE of 0.126761, MCR of 0.126761. The training, testing and total time fluctuate in values as this might be due fluctuations in the machine performance of running the NN model that is inconsistent, however these values are similar and alike nonetheless and only fluctuate with minor  $\pm 0.02s$  deviations for the training time,  $\pm 0.0001s$  for the testing time, and  $\pm 0.02s$  for the total time.

All in all, the aforementioned chosen parameter values are maintained as the best performing parameter values for this problem domain and dataset for credit card approval classification.

### **3 Modification into an Evolutionary Neural Network**

Analogous to brute-force techniques, multiple attempts were performed to identify the effect of different hidden layers, neurons, maximum iterations, and pairs of activation functions and solvers on the accuracy of classification. As such, efforts in implementing the neural network above heavily revolve around trial-and-error, which proves inefficient as it does not guarantee an optimal solution. This is further cemented as our efforts were a general attempt in finding a solution which could produce a better output, without knowing if it was the best solution.

Therefore, the neural network applied to the credit card dataset above can be further modified as a hybrid intelligent system, particularly, an evolutionary neural network. This hybrid system is able to overcome the suboptimal results produced by the initial neural network as it employs a genetic algorithm (GA) to identify the optimal weights and topology, in lieu of learning through back propagation. In GAs, new solutions, also known as offsprings, are created from existing solutions or generations, where their performance is estimated by a fitness rating.

The implementation of an evolutionary neural network on our dataset is discussed in the following sections.

#### **3.1 Generating an initial population**

Essentially, the GA in an evolutionary neural network aims to identify the optimal weights and network architecture, and as such, would require chromosomes that can be represented as multiple neural networks of different parameters. This can be achieved through direct encoding, where the weights and connections between neurons are directly encoded into a chromosome, albeit with a fixed number of neurons. This is to ensure that all chromosomes have the same architecture and thus, will be compatible for sexual crossover in the subsequent step.

Generally, each neuron connection in direct encoding corresponds to 1 bit on the chromosome, where ‘1’ signifies that there exists a connection between two neurons, while ‘0’ shows that there is no connection, as illustrated in Figure 10. However, this example only depicts the direct encoding of connections which excludes the weights.



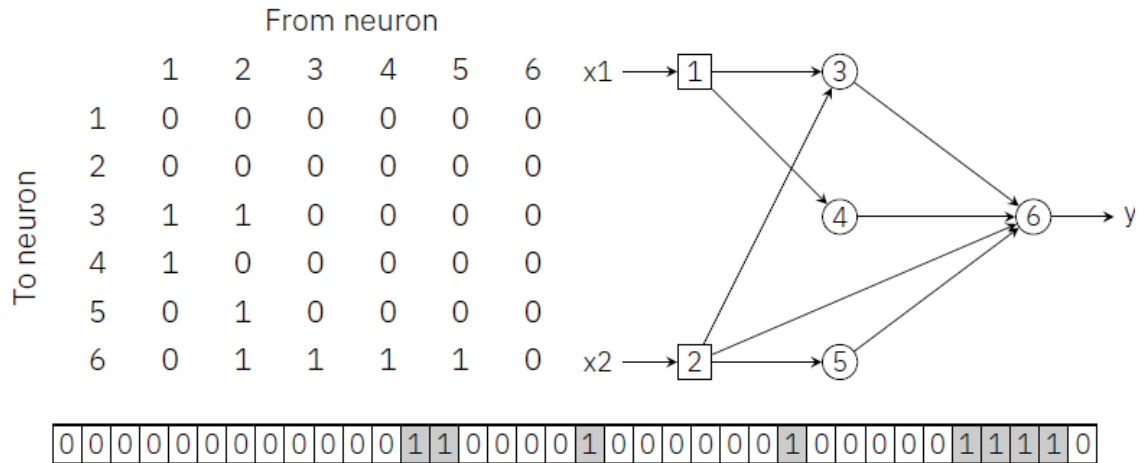


Figure 10: Direct encoding demonstrating connection topology

The relation between a connection and its weight can be represented through a pair of Index Bit and Weight Encoding Bits, which indicates the existence of a connection and its binary-encoded weight [3]. An example is shown in Figure 11, where both the connection and its weight is encoded as genes in the chromosome.

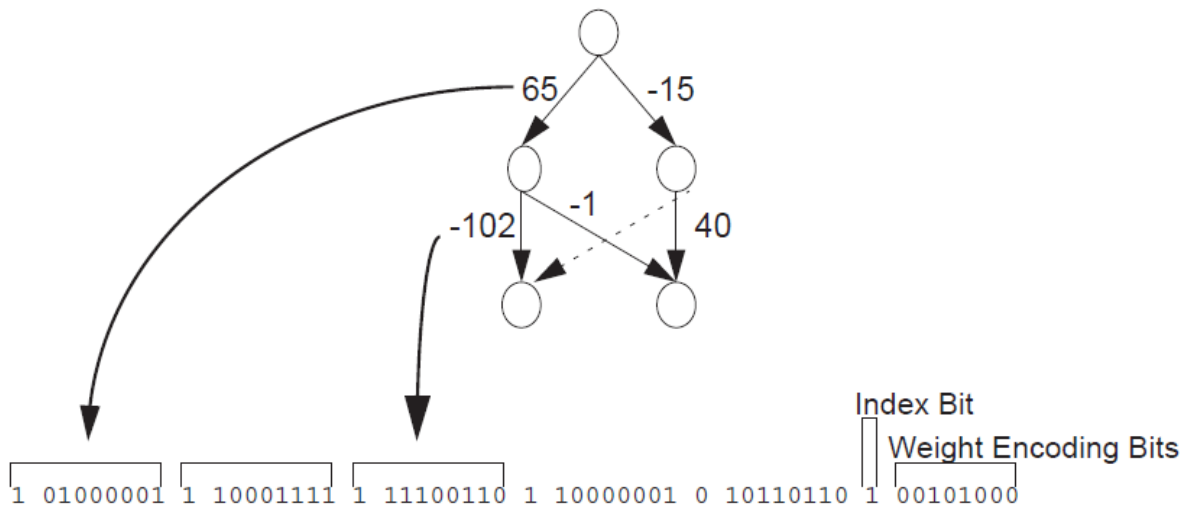


Figure 11: Encoded chromosome with weights and connection [3]

Once the chromosomes of the initial population have been generated, crossover and mutation is performed.

### **3.2 Crossover and mutation**

Pairs of chromosomes are selected as parents, where crossover is performed on every pair to produce offspring. This is followed by mutation, where a particular gene on every offspring is mutated by adding a small random value in the range of -1 and 1. As a result, potentially better neural networks are produced as offsprings.

### **3.3 Fitness evaluation**

A fitness function is then applied to estimate the performance of each offspring with the aim of identifying the weights and topology that produces minimal errors. Processes 3.2 and fitness evaluation are repeated until a target fitness is achieved, which we can define as a value closer to 1. As a result, an 'evolved' neural network with optimal weights and topology is produced, which in turn, has greater accuracy in identifying the *Approved* target class.

## **Conclusion**

In summary, we have successfully employed ANN to assist the banking domain, particularly, in determining the approval of a credit card application. To illustrate, the algorithm processes the input credit card approval dataset and utilises an applicant's past credit and application history to determine the outcome of approval. Several tests were run to identify better parameters for the model, namely, tests concerning the activation function and solver, the number of hidden layers and neurons, and the maximum number of iterations. Although a final test was run with the best outcome from each parameter test, our efforts in determining these parameters were based on trial-and-error, and as such, we have proposed the modification of the algorithm into an Evolutionary Neural Network.

## References

- [1] Lucci, S. & Kopec, D., 2016. Artificial Intelligence in the 21st Century. 2nd ed. Mercury Learning and Information
- [2] Negnivitsky, M., 2005. Artificial Intelligence: A Guide to Intelligent Systems 2nd ed. Addison Wesley.
- [3] D. Whitley, T. Starkweather, and C. Bogart, "Genetic algorithms and neural networks: optimizing connections and connectivity," *Parallel Computing*, vol. 14, no. 3, pp. 347-361, Aug. 1990. Accessed: Nov. 9, 2022. doi: 10.1016/0167-8191(90)90086-O. [Online]. Available: <https://www.sciencedirect.com/science/article/abs/pii/016781919090086O>
- [4] UCI Center for Machine Learning and Intelligent Systems, "Credit Approval Data Set," archive.ics.uci.edu. <https://archive.ics.uci.edu/ml/datasets/Credit+Approval> (accessed: 27 June, 2022).