

Summary of Neural Network Models

Project of machine learning

By Wang Jian(Steven)

Course: Data science for economics (classe lm-data) (code: B79, class: LM-DATA - Data science for economics) Year 2

Matriculation number: 14326A

Comparative Analysis of Neural Network Models for Credit Default Prediction

Contents

Abstract	2
Introduction	2
Methodology	2
Data Preparation	2
Experimental Setup	2
Neural network model execution	4
Results	6
Discussion	6
Models comparison	7
Other models	7
Conclusion	8
Appendices	9
Variable list	9
Code	10
Runtime details	10

Abstract

This report presents a comparative analysis between different neural network architectures aimed at predicting credit defaults. Utilizing a real-world dataset, various neural network models are implemented and evaluated based on accuracy, computational efficiency, and reproducibility.

Introduction

The prediction of credit defaults is a critical task for financial institutions. Neural networks have been recognized as an effective method for this purpose. This project examines various neural network architectures to identify the most efficient model in terms of predictive performance and computational requirements.

The dataset is obtained from Kaggle and the existing codes on the platform are not referenced.

Methodology

Data Preparation

The dataset¹ was processed using Pandas and preprocessed for neural network application. Categorical features are transformed by one hot encoding method². Standardization of scalar features was performed using StandardScaler from Scikit-Learn to maintain fairness in model comparisons. The dataset was subsequently divided into training and testing sets, adhering to an 80-20 split.

Neural Network Models

Several neural network models were implemented utilizing TensorFlow. A base model comprising two dense layers with ReLU activation and a sigmoid activation output layer was established to set a benchmark for subsequent comparisons.

Experimental Setup

Training of the models occurred on the designated training set, incorporating a 20% validation split. The Adam optimizer and binary cross-entropy loss were uniformly applied across all models to maintain consistency in the experimental conditions. Hyperparameters were selected based on prior experiments and literature reviews to mitigate overfitting and underfitting.

1. Model Compilation³:

¹ <https://www.kaggle.com/datasets/uciml/default-of-credit-card-clients-dataset/data>. This dataset contains information on default payments, demographic factors, credit data, history of payment, and bill statements of credit card clients in Taiwan from April 2005 to September 2005. Variable list is in appendix.

² 'SEX', 'EDUCATION', 'MARRIAGE', 'PAY_0', 'PAY_2', 'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6'. An interesting finding is that, without transformation of categorical features, the accuracy level does not vary significantly.

³ Base model:

```
# Compile the model
```

```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

```
# Train the model
```

```
original_history = model.fit(X_train, y_train, epochs=20, batch_size=32, validation_split=0.2)
```

- **Input Layer:** defined by the input shape (`X_train.shape[1]`), which corresponds to the number of features in the input data.
- **Hidden Layers:** Two hidden layers are defined with 128 and 64 neurons, respectively, both using the ReLU (Rectified Linear Unit) activation function. ReLU is a commonly used activation function that introduces non-linearity into the model.
- **Output Layer:** The output layer consists of a single neuron with a sigmoid activation function, which is suitable for binary classification tasks. Sigmoid function squashes the output to a range between 0 and 1, representing the probability of the input belonging to the positive class (class 1).
- **Optimizer:** 'adam'. Adam is an adaptive learning rate optimization algorithm that's well-suited for training deep neural networks. It computes adaptive learning rates for each parameter.
- **Loss Function:** 'binary_crossentropy'. This is a common choice for binary classification problems. It measures the difference between the true labels and the predicted probabilities.
- **Metrics:** ['accuracy']. During training, accuracy will be monitored to evaluate the performance of the model.

2. Model Training:

- **Epochs:** Number of times the entire dataset is passed forward and backward through the neural network.
- **Batch Size:** Number of samples propagated through the network before the weights are updated. Smaller batch sizes tend to provide a regularizing effect and can lead to faster convergence.

3. Model Evaluation:

- The model is evaluated on the test set using the previously specified loss function and accuracy metric.

Theory Background:

In binary classification problems, the output layer typically consists of a single neuron with a sigmoid activation function, which produces values between 0 and 1, representing the probability of the input belonging to one of the two classes.

Math Equations:

1. ReLU Activation Function:

$$f(x) = \max(0, x)$$

- ReLU is defined as the positive part of its argument x . It introduces non-linearity by outputting the input directly if it is positive, otherwise, it outputs zero.

2. Sigmoid Activation Function: $\sigma(z) = \frac{1}{1+e^{-z}}$

- Used in the output layer to squash the output between 0 and 1, representing the probability of the input belonging to class 1.

3. Binary Crossentropy Loss Function:

$$L(y, \hat{y}) = -\frac{1}{N} \sum_{i=1}^N [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)]$$

- Where y represents the true labels (0 or 1), \hat{y} represents the predicted probabilities, and N is the number of samples.

4. Adam Optimization Algorithm:

- Adam updates the weights using the gradient of the loss function with respect to the weights. It adapts the learning rates of each weight parameter by estimating the first and second moments of the gradients.

Structure-related Info vs. Hyperparameters:

- **Structure-related Info:**
 - Input Layer: Determined implicitly by the shape of the input data.
 - Output Layer: Single neuron with sigmoid activation for binary classification.
 - Hidden Layers: Number of hidden layers and neurons per layer are not specified in this code snippet.
- **Hyperparameters:**
 - Activation Functions: ReLU for hidden layers, sigmoid for the output layer.
 - Optimizer: adam
 - Loss Function: binary_crossentropy
 - Metrics: accuracy
 - Epochs
 - Batch Size

Neural network model execution

Steps are followed as training, repeat(try additional three times to see the change of metrics), adjustment, fine-tuning(with different hyperparameter groups).

1. model 0 and re-running(they are the same model runned for 4 times):

Model	Training Accuracy (%)	Validation Accuracy (%)	Training Loss	Validation Loss	Test Accuracy (%)
0	83.06	80.04	0.3982	0.4642	81.45
1	83.42	80.62	0.3939	0.4571	81.02
2	83.23	80.75	0.3956	0.4523	81.40
3	83.11	81.31	0.3982	0.4495	81.17

Table 1 Four independent executions of training same neural networks

Best Model is Model 3 with Test Loss: 0.4502 and Test Accuracy: 0.8117

2. Then, fintuning the hyperparameters⁴

Model Variation	Training Accuracy (%)	Validation Accuracy (%)	Training Loss	Validation Loss	Test Accuracy (%)
Original	83.06	80.04	0.3982	0.4642	80.18
1	83.42	80.62	0.3939	0.4571	81.02
2	83.23	80.75	0.3956	0.4523	81.40
3	83.11	81.31	0.3982	0.4495	81.17
4	83.38	81.04	0.3933	0.4569	81.43
5	83.58	81.18	0.3935	0.4529	81.23
6⁵	83.22	80.62	0.3980	0.4573	81.95
7	83.20	80.92	0.3956	0.4545	80.83
8	83.11	81.08	0.3963	0.4537	81.57
9	83.28	80.21	0.3949	0.4628	81.33
10	83.31	81.10	0.3941	0.4548	81.75

Table 2 fintuning hyperparameters

The best test accuracy comes from 6th variation model⁶.

3. Last, fin-tuening⁷ the structure/other hyperparameters⁸

Model Configuration	Test Accuracy (%)	Training Accuracy (%)	Validation Accuracy (%)	Training Loss	Validation Loss
64 neurons first layer, 32 neurons second	81.95	82.78	81.02	0.4044	0.4518
64 neurons first layer, 48 neurons second	81.30	82.86	81.10	0.4058	0.4487
64 neurons first layer, 64 neurons second	81.53	82.79	81.48	0.4028	0.4420
64 neurons first layer, 80 neurons second	81.32	83.05	81.12	0.3996	0.4526
64 neurons first layer, 96 neurons second	81.28	83.17	81.31	0.4009	0.4497
64 neurons first layer, 112 neurons second	80.78	83.21	80.52	0.3984	0.4638
96 neurons first layer, 32 neurons second	81.52	82.82	81.08	0.4026	0.4478
96 neurons first layer, 48 neurons second	81.32	82.89	80.87	0.4028	0.4534
96 neurons first layer, 64 neurons second	81.30	83.20	80.81	0.3969	0.4536
96 neurons first layer, 80 neurons second	81.35	83.26	81.12	0.3964	0.4510
96 neurons first layer, 96 neurons second	81.25	83.22	81.00	0.3946	0.4497

⁴ creat more models to fine tune : epochs +- 10 holding others constant. Batch +-10 holding others constant. Validation split+- 0,1 holding others constant.

⁵ Decrease validation split by 0.1 holding other hyperparameters constant.

⁶ Variation Parameters: Epochs Variation = 0, Batch Size Variation = 0, Validation Split Variation = -0.1.

⁷ Additional sample data finetuning is not possible. All fintuning steps are focusing on adjustments of hyperparameters/structure of the model.

⁸ All results and metrics for the models change everytime the codes are run.

Model Configuration	Test Accuracy (%)	Training Accuracy (%)	Validation Accuracy (%)	Training Loss	Validation Loss
96 neurons first layer, 112 neurons second	81.45	83.05	81.10	0.3951	0.4542
128 neurons first layer, 32 neurons second	81.68	83.02	81.46	0.4024	0.4482
128 neurons first layer, 48 neurons second	81.63	83.11	81.04	0.3993	0.4550
128 neurons first layer, 64 neurons second	81.23	83.22	81.27	0.3964	0.4569
128 neurons first layer, 80 neurons second	81.17	83.21	81.02	0.3961	0.4504
128 neurons first layer, 96 neurons second	81.10	83.32	81.38	0.3950	0.4542
128 neurons first layer, 112 neurons second	81.68	83.46	81.00	0.3907	0.4563
160 neurons first layer, 32 neurons second	81.45	82.85	81.10	0.3978	0.4595
160 neurons first layer, 48 neurons second	81.32	83.39	81.17	0.3946	0.4450
160 neurons first layer, 64 neurons second	81.20	83.22	80.69	0.3954	0.4527
160 neurons first layer, 80 neurons second	81.35	83.37	81.25	0.3942	0.4532
160 neurons first layer, 96 neurons second	81.37	83.21	80.96	0.3922	0.4592
160 neurons first layer, 112 neurons second	80.85	83.56	80.65	0.3850	0.4589
192 neurons first layer, 32 neurons second	81.40	83.01	81.21	0.3995	0.4496
192 neurons first layer, 48 neurons second	80.48	83.36	80.52	0.3960	0.4652
192 neurons first layer, 64 neurons second	81.43	83.33	81.35	0.3926	0.4602
192 neurons first layer, 80 neurons second	81.47	83.38	81.10	0.3895	0.4591
192 neurons first layer, 96 neurons second	81.52	83.51	80.87	0.3890	0.4548
192 neurons first layer, 112 neurons second	80.45	83.67	80.40	0.3878	0.4631
224 neurons first layer, 32 neurons second	81.50	83.23	81.17	0.3965	0.4552
224 neurons first layer, 48 neurons second	80.98	83.14	80.69	0.3954	0.4674
224 neurons first layer, 64 neurons second	81.58	83.27	81.23	0.3946	0.4533
224 neurons first layer, 80 neurons second	81.60	83.18	81.19	0.3918	0.4566
224 neurons first layer, 96 neurons second	81.25	83.49	80.81	0.3882	0.4544
224 neurons first layer, 112 neurons second	80.97	83.52	80.63	0.3866	0.4565

Table 3 fintuning hyperparameters on neuron numbers in each layer

Results

Model performance was primarily evaluated using accuracy metrics on the testing dataset. The models' test accuracy provides a benchmark for subsequent model comparisons. Since there are not significant accuracy(especially test accuracy) variations and runtime differences among the chosen NN models, choice of the model should be mainly based on dimation and complexity of the dataset, computational resources, further finetuned results/models.

Discussion

The analysis covers the performance of each model against its computational/fintuning complexity and requirements. Considerations include the balance between model intricacy and

predictive performance, although most metrics do not vary a lot. The reproducibility of results is also examined and maintained, by emphasizing the transparent and precise documentation of experimental setups and parameters⁹.

Models comparison

Other models

Previously trained ML models, including logistic, random forest, KNN, are listed below with metrics:

Model ¹⁰	Accuracy	Precision	Recall	F1-Score	ROC AUC
Logistic Regression	0.784	1	0.000771	0.001541	0.652552
Random Forest	0.821667	0.647594	0.383963	0.482091	0.765032
K-Nearest Neighbors	0.789333	0.519031	0.346955	0.415896	0.709843
Logistic Ridge	0.809	0.680119	0.232143	0.346139	0.723051

Table 4 Model metrics comparisons

Below are the runtime records¹¹:

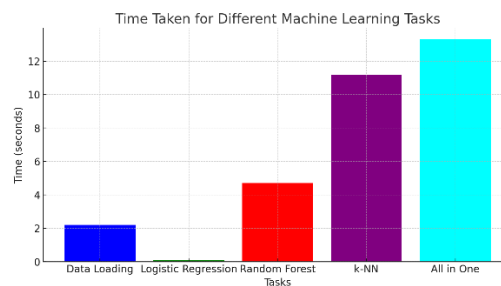


Figure 1 Runtime statistics of logistic(similar to ridge logistic regression), random forest, KNN models

Introducing Neural Network algorithms, the single step run time record and collective comparisons with all models(only on training session) are as follows¹²:

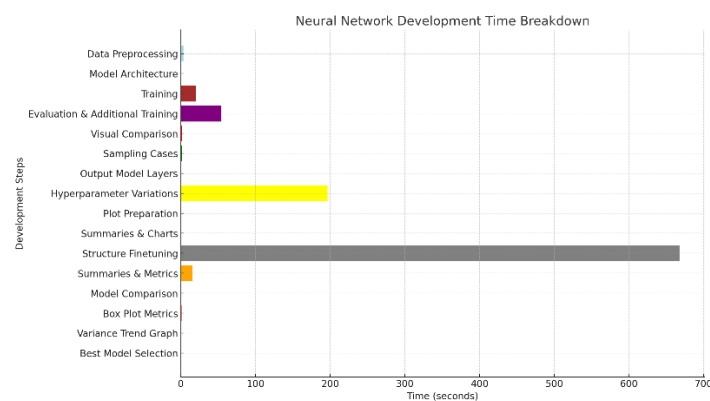


Figure 2 Runtime detail bar chart of Neural Network codes

⁹ Details in the code documents.

¹⁰ All models' codes are attached in github.

¹¹ Details in appendix.

¹² After each execution of the codes, the results vary. Small variance can be ignored.

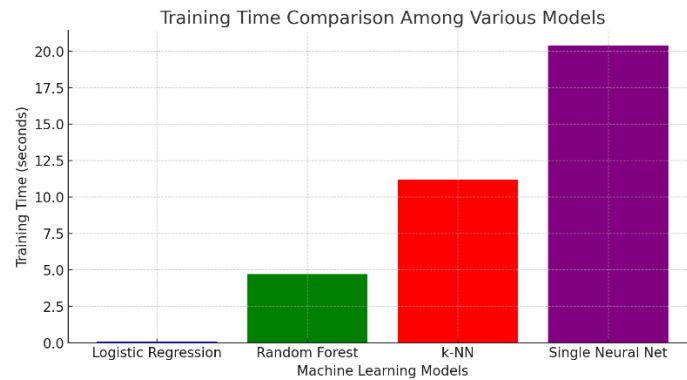


Figure 3 Comparison of all models regarding single training session runtime

Logistic Regression(and Ridge logistic regression¹³): Fastest training time at only 0.1 seconds, suitable for simpler datasets and problems.

Random Forest: More time-consuming at 4.7 seconds due to its complex nature of training multiple decision trees.

k-Nearest Neighbors (k-NN): Slower still at 11.2 seconds, particularly for large datasets due to the need for calculating distances between instances.

Single Neural Network Model: Takes the longest time to train at 20.4 seconds, reflecting the complexity and depth involved in neural network training.

Training neural network models has the highest computational resource usage and the finetuning results may not be higher than random forest models, but the above data has shown the result that any neural network model has outperformed logistic regression model¹⁴ and ridge logistic regression model¹⁵.

Conclusion

The investigation underscores the capability of neural networks in the context of credit default prediction. The findings indicate that the model accuracy level is higher than other models previously tested. Although the interpretation of neural network model is not straight forward like other simpler models, and the runtime level costs computational resources, its performance is superior. Further research could investigate more sophisticated architectures and feature engineering strategies to enhance predictive accuracy.

There are multiple improvements which could be done in the future analysis.

1. Structural finetuning by adding additional hidden layers in the model, or more aggressive adjustment of the hyperparameters which are already adjusted in the previous training sessions.
2. Additional sample data could be validated and used as input to further verify or finetune the model.

¹³ Ridge logistic regression runtime level is similar, close to 0.

¹⁴ This does not exclude the possibility where, logistic models can be improved.

¹⁵ Accuracy: 0.809. But the penalty variation does not improve the model. Code is attached in github.

3. It could be useful to find the measurements to gauge the suitable complexity of models when confronting large data sets. Computational resources, time cost¹⁶, etc., should be properly considered.

Appendices

Supplementary materials encompass the project's code, detailing data preprocessing, model construction, training protocols, and evaluation metrics, ensuring the reproducibility of the experiments.

Variable list

Variable	Description
ID	ID of each client
LIMIT_BAL	Amount of given credit in NT dollars (includes individual and family/supplementary credit)
SEX	Gender (1=male, 2=female)
EDUCATION	Education level (1=graduate school, 2=university, 3=high school, 4=others, 5=unknown, 6=unknown)
MARRIAGE	Marital status (1=married, 2=single, 3=others)
AGE	Age in years
PAY_0	Repayment status in September, 2005 (-1=pay duly, 1=payment delay for one month, ..., 9=payment delay for nine months and above)
PAY_2	Repayment status in August, 2005 (scale same as above)
PAY_3	Repayment status in July, 2005 (scale same as above)
PAY_4	Repayment status in June, 2005 (scale same as above)
PAY_5	Repayment status in May, 2005 (scale same as above)
PAY_6	Repayment status in April, 2005 (scale same as above)
BILL_AMT1	Amount of bill statement in September, 2005 (NT dollar)
BILL_AMT2	Amount of bill statement in August, 2005 (NT dollar)
BILL_AMT3	Amount of bill statement in July, 2005 (NT dollar)
BILL_AMT4	Amount of bill statement in June, 2005 (NT dollar)
BILL_AMT5	Amount of bill statement in May, 2005 (NT dollar)
BILL_AMT6	Amount of bill statement in April, 2005 (NT dollar)
PAY_AMT1	Amount of previous payment in September, 2005 (NT dollar)
PAY_AMT2	Amount of previous payment in August, 2005 (NT dollar)
PAY_AMT3	Amount of previous payment in July, 2005 (NT dollar)
PAY_AMT4	Amount of previous payment in June, 2005 (NT dollar)

¹⁶ The whole project takes me around 2 months time excluding non working days. Where simple models took around 2-3 weeks and neural network model took more than one month. The time spent consists of model choosing, design, coding, fintune, result interpretations.

Variable	Description
PAY_AMT5	Amount of previous payment in May, 2005 (NT dollar)
PAY_AMT6	Amount of previous payment in April, 2005 (NT dollar)
default.payment.next.month	Default payment (1=yes, 0=no) for this dataset

Table 5 variable list of the dataset

Code¹⁷

As shown in github¹⁸ and attached in the email.

Runtime details

Three single algorithms time breakdown:

Data Loading: Took 2.2 seconds. This process involves reading the file from disk, which is typically limited by the I/O speed of the system rather than the CPU.

Logistic Regression Training: Took only 0.1 seconds. Logistic Regression is faster compared to other models as it involves solving a convex optimization problem that usually converges quickly.

Random Forest Training: Took 4.7 seconds. This is more time-consuming due to the need to train multiple decision trees, which increases computational and memory requirements.

k-Nearest Neighbors (k-NN) Training: Took the longest at 11.2 seconds. k-NN can be slow, especially with large datasets, as it requires computing the distance to every instance in the training set.

All in One Code + Graph: Took 13.3 seconds. This includes the cumulative time of training all models, making predictions, and generating ROC curves for each.

Neural Network Development Time Breakdown:

Section 1: Single Model

Data Preprocessing: 3.5s

Model Architecture: 0s

Training: 20.4s

Evaluation & Additional Training: 54.5s

Visual Comparison: 1.1s

Sampling Cases: 1.8s

Output Model Layers: 0s

Section 2: Hyperparameter Finetuning

¹⁷ Results in this report and in the markdown area of the code are not finite. Each execution of the code produces different results.

¹⁸ <https://github.com/StevenFromUnimiMIEDSE/Unimi-DSE-Machine-Learning-Project>

Variations Setting (10 models): 3m 16.4s

Plot Function Preparation: 0s

Print Summaries and Charts: 1s

Section 3: Model Structure Finetuning

Changing Number of Neurons (36 models): 11m 7.8s

Print Summaries, Metrics, Graphs: 16.1s

Compare Models Collectively: 0.7s

Box Plot for Metric Variations: 1.5s

Shaded Variance Trend Graph: 0.6s

Best Model Selection: 0s