Rotterdam School of Management
Erasmus University

Individual Assignment 2

Develop an Artificial Neural Network (ANN)

Khoi Gia Pham - 523755kg

Course: Introduction to Deep Learning CMME139

MSc Business Information Management

Rotterdam School of Management, Erasmus University

Table of contents

1. Introduction

In this report, we delve into the process of developing and fine-tuning deep learning models for a binary classification task on the cleaned "17K Mobile Strategy Games" dataset (comprising 11,878 entries with 18 features). Initially, a base neural network model is developed and assessed. Subsequently, we explore strategies for model optimization, including adjusting the network architecture, manipulating neuron counts, and regulating dropout rates. Our models are then evaluated and compared, revealing Model 5 as the top performer. The deep learning models notably outperform traditional models like SVM, NB, and KNN in terms of accuracy, but the benefits need to be weighed against potential computational costs and development efforts.

2. Data Preparation

1. **Conversion to Numeric:** The first step is converting all columns into numeric values, except for the objective column *Categorical Rating Count*.
2. **One-hot Encoding:** The column *Categorical Rating Count* then undergoes a one-hot encoding procedure, which is a process of converting categorical data into a binary (0 or 1) matrix. Each unique value in the original column becomes a new column in the transformed data. After this process, the original *Categorical Rating Count* column is removed from the dataset.
3. **Formatting:** The data is then converted to a matrix format and the column names are removed. This is done to meet the input requirements of the Neural Network model.
4. **Partitioning:** The data is randomly partitioned into a training set and a test set, with approximately 70% of the data going to the training set and 30% to the test set. The seed for the random number generator is set to ensure that the random split of the data can be reproduced.
5. **Feature Scaling:** The features in the training and test sets are standardized (scaled) to have a mean of 0 and standard deviation of 1. The mean and standard deviation from the training set are used to standardize the test set to prevent information leakage.

3. Model Development

3.1 Model Creation

This model serves as the baseline model for further parameter comparison. The model is a neural network with 3 layers:

1. The first layer is a dense (fully connected) layer with 32 units (neurons) and uses the ReLU (Rectified Linear Unit) activation function. This is also the input layer, and it expects input vectors of 18 elements (**input_shape = c(18)**).
2. The second layer is another dense layer with 16 units and also uses the ReLU activation function.
3. The third layer is the output layer, which is also a dense layer. It has 2 units and uses the softmax activation function, making it suitable for a binary classification problem.

3.2 Model Compilation

The model is compiled with the RMSprop optimizer and uses categorical cross-entropy as its loss function, which is suitable for multi-class classification problems. The metric that will be used to measure model performance during training and testing is accuracy.

3.3 Model Training

The model is trained on the **training** data with corresponding labels **trainingtarget** for 50 epochs. An epoch is one complete pass through the entire training dataset. A batch size of 32 is used, meaning the model weights are updated after each 32 samples.

The **validation_split** argument is set to 0.2, which means 20% of the training data will be set aside and used as validation data. The model will not train on this data, instead, it will use this data to evaluate how well it's learning during training.

3.4 Model Evaluation:

After the training phase, the model's performance is evaluated on the **test** data and corresponding **testtarget** labels. Finally, the **plot** function is used to plot the training history of the model, showing the progression of loss and accuracy on both the training and validation data over the epochs (Figure 1). This can help identify if the model is underfitting or overfitting the training data.

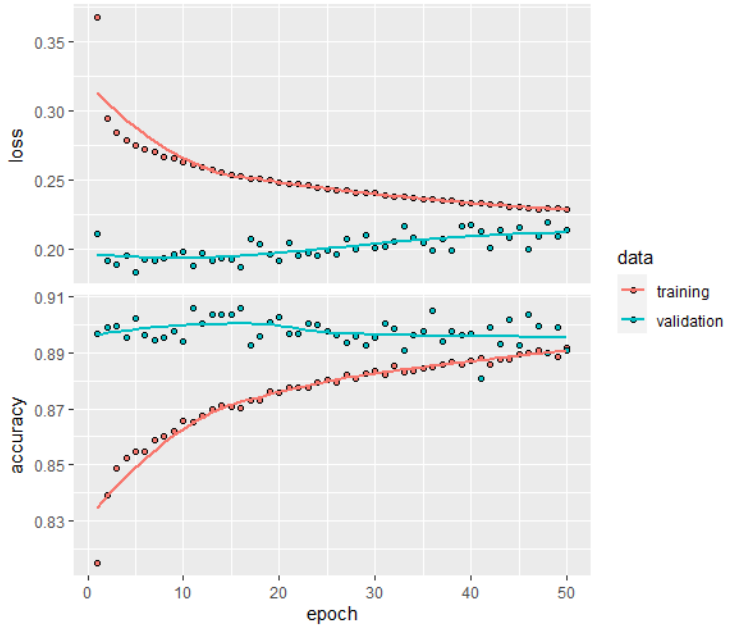

Figure 1. Training history of Model 1 (baseline)

From the plot**,** we can observe that **training loss keeps decreasing while validation loss starts increasing**: This is a classic sign of overfitting. The model continues to improve on the training data, but its performance on the validation set starts to degrade. This means the model is learning the specific examples in the training data too well and is unable to generalize to unseen data (Carremans, 2020).

4. Model Tuning

4.1 Parameters Testing

Model 2: Addition of a layer and Dropout

The first tuned model (**model2**) includes the addition of another Dense layer and the application of Dropout method after each Dense layer.

1. This model has 4 Dense layers with 32, 16, 8, and 2 units, respectively. The activation function used is ReLU (Rectified Linear Unit) for the first three layers and softmax for the last layer.
2. Dropout layers have been added after each Dense layer. Dropout randomly sets input units to 0 with a frequency of 'rate' at each step during training time to help prevent overfitting. The rates used are 0.3, 0.2, and 0.1 respectively.
3. The model is compiled using the RMSprop optimizer, categorical cross-entropy loss function, and accuracy as the metric.

4. The model is trained on the training data for 50 epochs with a batch size of 32 and validated on 20% of the training data.

Model 3: Increasing number of neurons

The second model (**model3**) keeps the structure of **model2** (including Dropout) but increases the number of neurons in each Dense layer.

1. This model has 4 Dense layers with 64, 32, 16, and 2 units, respectively. The activation function used is ReLU for the first three layers and softmax for the last layer.
2. Dropout layers have been added after each Dense layer. The rates used are 0.3, 0.2, and 0.1 respectively, the same as in **model2**.
3. The model is compiled using the RMSprop optimizer, categorical cross-entropy loss function, and accuracy as the metric.
4. The model is trained on the training data for 50 epochs with a batch size of 32 and validated on 20% of the training data.

Model 4: Increase Dropout Rate

Thirdly, **model4** increases the dropout rates while keeping the same number of neurons in each Dense layer (compared to **model2**).

1. The model structure includes 4 Dense layers with 32, 16, 8, and 2 units respectively. The activation function used is ReLU for the first three layers and softmax for the last layer.
2. The Dropout layers follow each Dense layer, but the rates have been increased to 0.5, 0.4, and 0.3 respectively.
3. This model is compiled with the RMSprop optimizer, categorical cross-entropy as the loss function, and accuracy as the metric.
4. The model is trained on the training data for 50 epochs with a batch size of 32 and validated on 20% of the training data.

Model 5: Increase Number of Neurons and Dropout Rate

Finally, **model5** increases both the number of neurons in each Dense layer and the dropout rates (compared to **model2**).

1. The model structure includes 4 Dense layers with 64, 32, 16, and 2 units respectively. The activation function used is ReLU for the first three layers and softmax for the last layer.

2. The Dropout layers follow each Dense layer, with rates of 0.5, 0.4, and 0.3 respectively.
3. This model is compiled with the RMSprop optimizer, categorical cross-entropy as the loss function, and accuracy as the metric.
4. The model is trained on the training data for 50 epochs with a batch size of 32 and validated on 20% of the training data.

4.2 Results Comparison

Model 5 performed the best in terms of both loss and accuracy. However, the differences between the models are minimal, as illustrated in Table 1 below.

Model	Loss	Accuracy
Model 1	0.2728	86.91%
Model 2	0.2529	87.13%
Model 3	0.2564	87.29%
Model 4	0.2561	86.84%
Model 5	0.2520	87.48%

Table 1. Summary of Loss and Accuracy

Model 1: The base model achieved a loss of 0.2727631 and an accuracy of 86.91%. This serves as the baseline for comparison with the other models.

Model 2: The fine-tuned model with an extra layer and Dropout achieved a lower loss (0.2529272) and slightly higher accuracy (87.13%) compared to the base model, indicating better generalization.

Model 3: The model with more neurons in each layer (compared to Model 2) shows a slightly increased loss (0.2563752) but also a higher accuracy (87.29%) than Model 2. It indicates that having more neurons did improve accuracy, but also increased the loss slightly.

Model 4: The model with an increased dropout rate (compared to Model 2) achieved a loss of 0.2560653, and the accuracy slightly decreased to 86.83%. This suggests that increasing the dropout rate led to a higher test loss and reduced accuracy, perhaps indicating that the model was slightly underfitting the data.

Model 5: The model with both increased neurons and dropout rate (compared to Model 2) achieved the lowest loss (0.2520169) and the highest accuracy (87.48%) among all models. It shows that increasing both model complexity (more neurons) and regularization (higher dropout) managed to achieve a good balance, leading to the best performance on the test data.

Loss and Accuracy Graphs

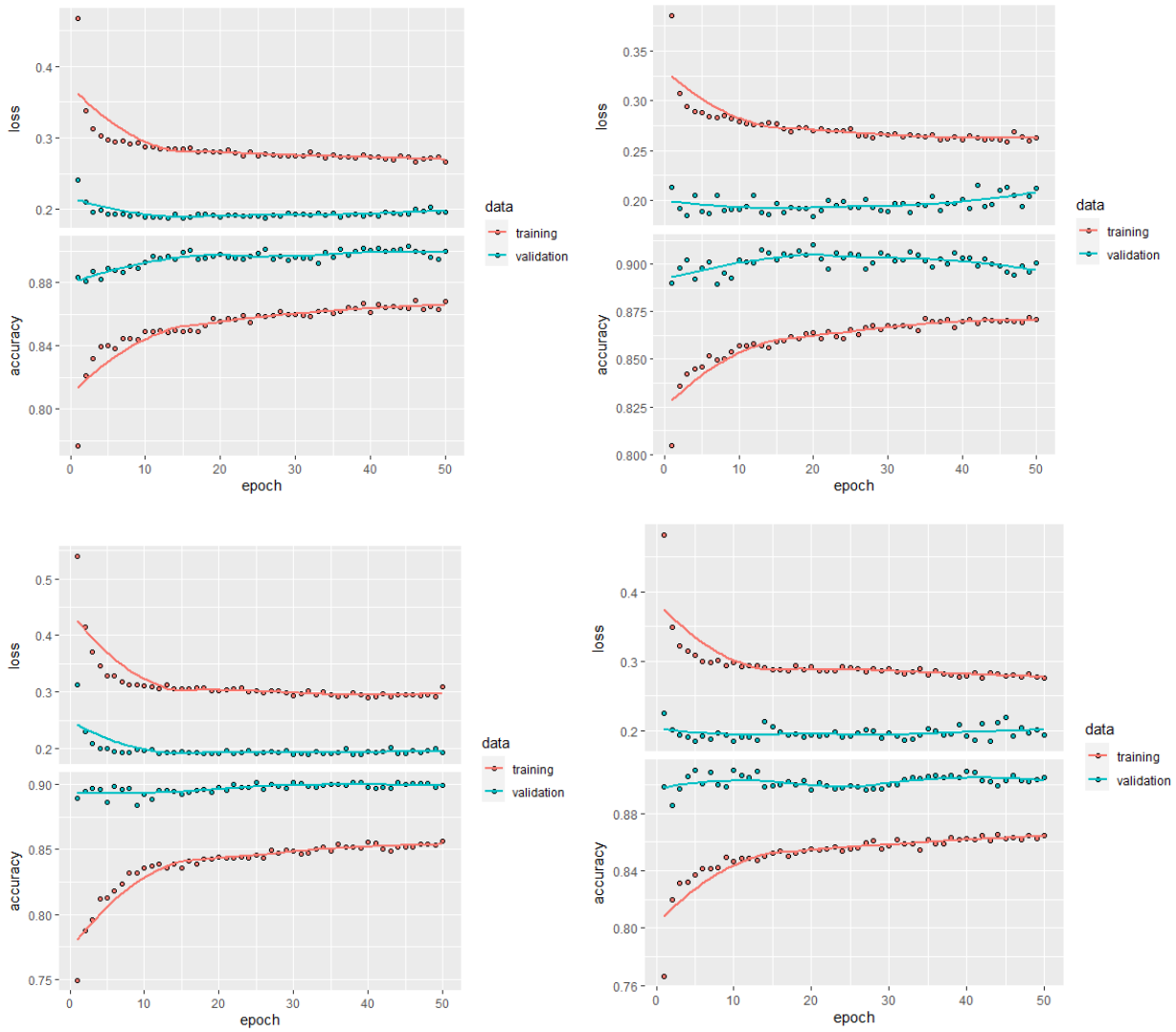

Figure 2. Training Histories of Models 2-5 (Top Left: Model 2, Top Right: Model 3, Bottom Left: Model 4, Bottom Right: Model 5)

The four models display similar trends in the Figure 2, with two primary observations:

The first observation is that both the training and validation losses decrease, yet a gap exists between them. This discrepancy might indicate potential overfitting. While the model demonstrates satisfactory performance on the training data, it struggles to generalize to unseen validation data (Carremans, 2020).

The second observation is the validation loss dips below the training loss. Generally, this isn't an issue. It may arise due to the application of dropout or other regularization techniques during the training phase, which aren't employed during validation (Carremans, 2020).

Alternatively, it could be attributed to the inherent randomness and variability in the subsets of data designated for training and validation (Carremans, 2020). Lastly, this trend might signify the potential for further training epochs to enhance model performance.

5. Comparison with Traditional models

The deep learning model does perform better than the traditional models. The accuracy of Model 5, which is the best model in this report, is 87.42%. This accuracy is higher than that of the SVM (85.39%), NB (83.86%), and KNN (83.01%) models. Further comparisons are as follows:

How much does it improve?

Compared to the best performing traditional model (SVM, 85.39%), the deep learning model improved the accuracy by approximately 2.03 percent.

Is it worth it computationally?

Given the moderate size of the dataset (11,878 entries and 18 features), the computational cost of the deep learning model is not prohibitively high. However, deep learning models can still take longer to train than traditional models (Brownlee, 2017). The increase in accuracy of 2.03 percentage points needs to be weighed against this potential increase in computational cost. If the minor increase in accuracy significantly impacts the classification task (for instance, if it considerably reduces the cost associated with misclassifications), it could be worth the additional computational expense (Brownlee, 2017).

Is it worth it effort-wise?

Solely in this Assignment, an increase of 2.03% in accuracy is not effort worthwhile. Deep learning models are generally more complex and have more hyperparameters to tune compared to traditional models, which can require more effort to optimize (Brownlee, 2017). However, the value of this increased effort would be higher if the stakes of our classification task are high and a 2.03% increase in accuracy can lead to substantial benefits.

In brief, while the deep learning model does outperform the traditional models, whether the additional computational costs and development effort are justified depends largely on the specific needs and constraints of the project, as well as the costs associated with misclassification in this binary classification task. However, given the context of this assignment, the additional computational costs and development effort are not worthwhile to a certain extent.

Bibliography

Brownlee, J. (2017) ‘How Much Training Data is Required for Machine Learning?’, *MachineLearningMastery.com*, 23 July. Available at: https://machinelearningmastery.com/much-training-data-required-machine-learning/ (Accessed: 29 May 2023).

Carremans, B. (2020) *How to Handle Overfitting in Deep Learning Models*, *freeCodeCamp.org*. Available at: https://www.freecodecamp.org/news/handling-overfitting-in-deep-learning-models/ (Accessed: 29 May 2023).