Rotterdam School of Management
Erasmus University

Assignment 3

Experiment with Parametrization in Neural Networks

Group 11

Khoi Gia Pham – 523755kg

Linping Tang – 656553lt

Zi-Xin Chen – 592198zc

Hong Yin Chi – 499347hc

Course: Introduction to Deep Learning CMME139

MSc Business Information Management

Rotterdam School of Management, Erasmus University

Table of contents

1. Introduction

In the rapidly evolving field of neural networks, effective parameter optimization is critical for building successful models. This report offers an in-depth analysis of how different parameters - including the *number of epochs, number of layers in a neural network, number of neurons in the layers, number of parameters, structure of the neural network, effect of dropout, effect of class imbalance,* and *number of training samples* - impact model performance and computational cost. In this study, we utilize the "17K Mobile Strategy Games" dataset, originally prepared for Assignment 2. This dataset comprises 16,983 observations across 19 variables. And the target variable, 'Categorical Rating Count', has been transformed through one-hot encoding.

2. Baseline model

2.1 Model Creation

1. This model has three hidden layers with 32, 16, and 8 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.

2.2 Performance Indicators

The variability in performance, both computational cost and accuracy, among baseline models with different parameterizations test may arise due to the execution of experiments on different computers. Nonetheless, we ensure the execution of all models within a given task to be on a same computer, thereby facilitating reliable performance comparisons.

3. Experimental Setting

3.1 Number of Epochs

To see how changing the number of epochs can affect model performance, the original plan was to test respective epoch values at: 10, 20, 50, 70 and 90 to see how adding more epochs can affect accuracy and computational costs, with 50 epochs being the baseline. After testing until 70 epochs, it was observed that the loss value and accuracy did not have significant improvement, and rather there were signs of decreasing loss values and accuracy (see Table 1 below). Indeed, the accuracy of 90 epochs is lower than that of 70 epochs. Later, it was decided to test epoch value at 40 since there was quite a big accuracy difference between the results of 30 and 50 epochs.

Table 1: Models with different number of epochs

epochs	loss	accuracy	user time	system time	elapsed time
10	0.2596	0.8641	6.086	0.813	7.088
30	0.2531	0.8699	17.766	1.832	19.169
40	0.2522	0.8752	22.729	2.199	23.976
50	0.2524	0.8731	26.988	2.583	29.161
70	0.2528	0.8742	41.772	4.389	43.278
90	0.2513	0.8737	51.428	5.751	55.241

Table 1 shows that increasing the number of epochs would initially improve accuracy, but there should be a point where increasing the epochs no longer significantly enhances the performance. This point is observed to be around epoch value 40, since the results after that showed no better accuracy. After that, the performance became stagnant and started to decrease, while the computational time kept rising. In terms of computational time, it was observed to increase with more epochs since each epoch requires additional computation.

Next, to test whether a small number of epochs can produce a model with sufficient performance, it was decided to test with epoch value 5 (Table 2). It is more likely for models with small numbers of epochs to underfit; however, if the computational resources are limited and the patterns in the dataset in question are not complex, a small number of epochs may be sufficient already to achieve reasonable performance.

Table 2: Models with 4 epochs

epochs	loss	accuracy	user time	system time	elapsed time
5	0.2642	0.8603	3.598	0.362	4.195

Looking at its performance, we can see that the accuracy performance is the worst among all when there are only 5 epochs, and the loss value of the model with only 5 epochs is the highest among all. However, the computational time of the 5 epochs model, the system time, is approximately 7 times faster than that of the baseline model. Depending on the priority of the research goal, researchers should consider whether to focus on loss value, accuracy or computational costs. In this case, since the highest computational time, which is 5.751 system time, is affordable, it can be concluded that a low number of epochs such as 5 is unsatisfactory and not optimal.

3.2 Number of Layers

To understand what happens when more hidden layers are added, it was decided to test models with hidden layer values from 1-7 (excluding the output dense layer), with 3-layer being the baseline model. Looking at the change in the hidden dense layer, we can observe the following change in accuracy, computational time and the number of parameters (see Table 3).

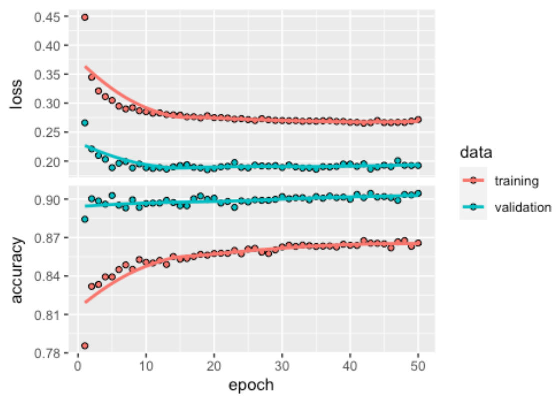
Table 3: Models with different number of layers

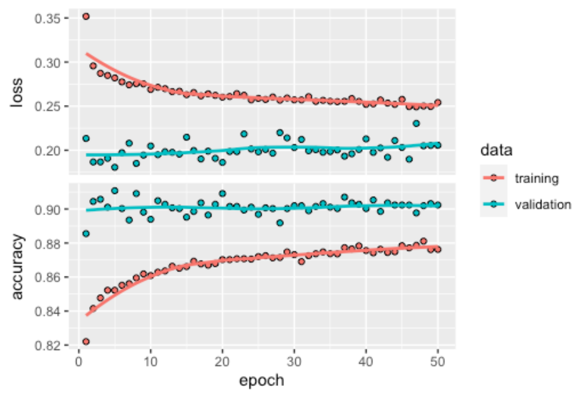
layers	accuracy	loss	system time	parameter	unit composition	dropout rate composition
1	0.8697	0.2520	3.522	674	32-2	0.3

2	0.8723	0.2517	6.401	1170	32-16-2	0.3-0.2
<u>**3**</u>	<u>0.8731</u>	<u>0.2524</u>	<u>2.583</u>	<u>1290</u>	<u>32-16-8-2</u>	<u>0.3-0.2-0.1</u>
4	0.8721	0.2541	4.897	1462	32-16-12-8-2	0.3-0.2-0.2-0.1
5	0.8711	0.2574	2.727	1490	32-16-12-8-4-2	0.3-0.2-0.2-0.1-0.1
6	0.8699	0.2539	3.122	1958	32-20-16-12-8-4-2	0.3-0.3-0.2-0.2-0.1-0.1
7	0.8454	0.3066	4.329	2802	32-28-20-16-12-8-4-2	0.3-0.3-0.3-0.2-0.2-0.1-0.1

1 layer

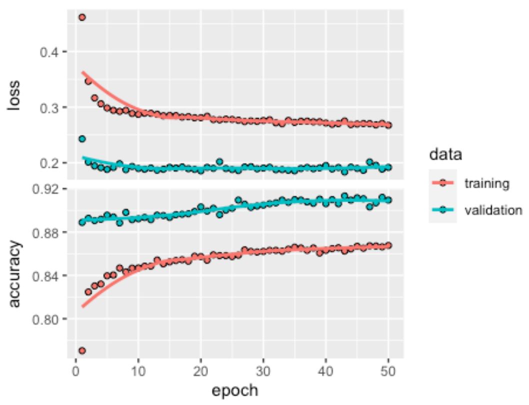
2 layers

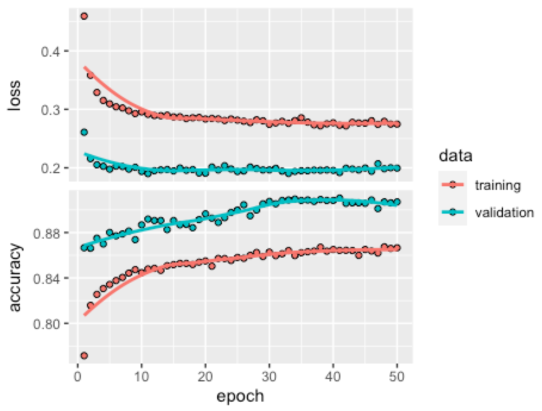




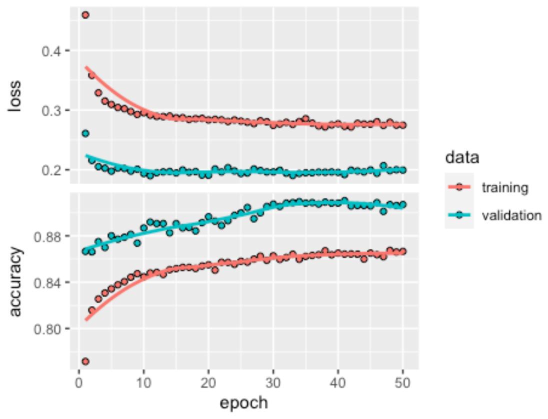
3 layers

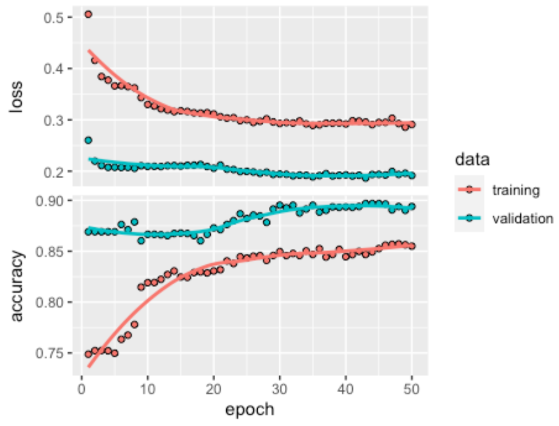
4 layers



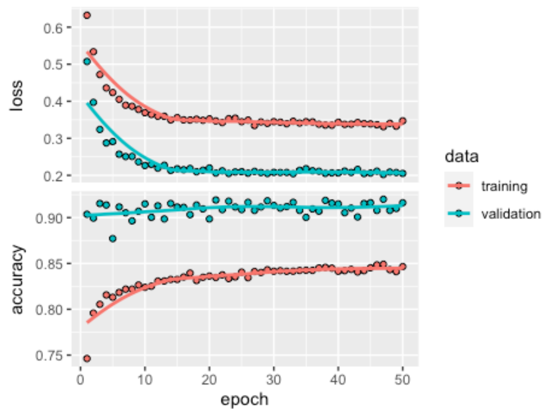


5 layers *6 layers*





7 layers



Generally, there are no rules to creating an optimal number of hidden layers. It was suggested by the lecturer to start with just a few and add one each time and observe the change in the outcome.

In Table 3, we can see that the accuracy rises gradually until it reaches the 3-layer model; after that, a consistent decreasing pattern can be observed as more hidden layers are added. Signs of overfitting can, for example, be seen in the error graph of the 7-layer model, where the validation set has a stagnant accuracy curve while the training set has a rising one. In terms of the number of parameters, it consistently increases as more layers are added. These findings are in line with the fact that due to the accumulated noises/misinformation contributed by each layer, errors are also accumulated as the layer number goes up.

The computational time should also reach a high point as more layers are added, as more and more parameters are required to be calculated. However, it is not the case with our models. It is observed that the system time does not consistently increase as more layers are added. Rather, the system time does not have drastic change as layers go up. There are different factors that could cause this phenomenon. It could be due to the hardware and software optimization. The frameworks that Keras uses often have optimizations that leverage hardware capabilities to speed up training, meaning that for example with a powerful GPU and proper configuration, the training time can be non-linear relative to the number of layers and parameters. It could also have been affected by different configurations of dropout rate, as different settings can potentially lower the model complexity and speed up convergence. Dataset complexity and the model's capacity can also contribute to these findings, as the dataset might be relatively simple, leading to no significant differences in processing time as the numbers of parameters and layers increase. In this case, this might imply that the computational time is not as relevant as the performance when the models are compared.

In terms of how linearity affects accuracy when changing the layers, we can also look at Table 3. When there are fewer layers, it means the model is less complex and more linear, because it has fewer opportunities for non-linear transformations to occur; as the layers increase, the model becomes more complex and less linear, and can grasp information from datasets more comprehensively. Results suggest that accuracy values are relatively consistent in the first 4 models, and that they start decreasing after the fifth model, with the most significant drop in the 7-layer model. It could suggest that before the 5-layer model, the less linear the models are, the better the ability to capture complex features and to generate food performance. However, after that point, increasing nonlinearity no longer makes the model perform better, meaning 3-4 layers might be the optimal complexity for this specific predicting goal.

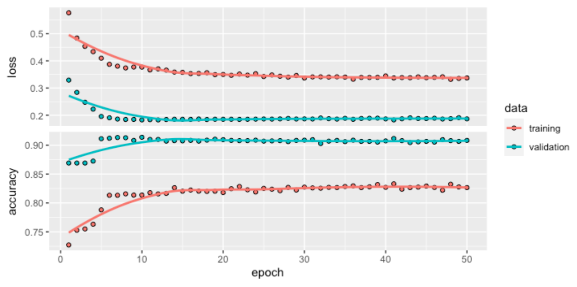
3.3 Number of Neurons in the Layers

There are two hypotheses proposed in the assignment description, concerning the change that adjusting the number of neurons in the layers can bring about. Firstly, a higher number of neuron representations should be suitable for solving non-linear problems but increases the probability of possible overfitting. And secondly, a lower number of neurons in a layer should support generalization and avoid overfitting but they are not suitable for non-linear problems. Hence, the change caused by adjusting the number of neurons in layers is observed for this experiment. In short, a range of values is defined for the number of neurons we want to experiment with. The values 8, 16, 32, 64, and 128 are chosen, two values lower and two values higher than the number of neurons from our initial model. Following a similar structure to the baseline model, the first layer has the appointed number of neurons as units, the second layer has neurons/2 units, third layer has neurons/4 units, and the last layer has 2 units (i.e. for the initial model, first layer has 32 units, second layer has 32/2 (16) units, third layer has 32/4 (8) units, and last layer has 2 units). The model is then compiled, trained, and evaluated, and the test accuracy is printed for each iteration.

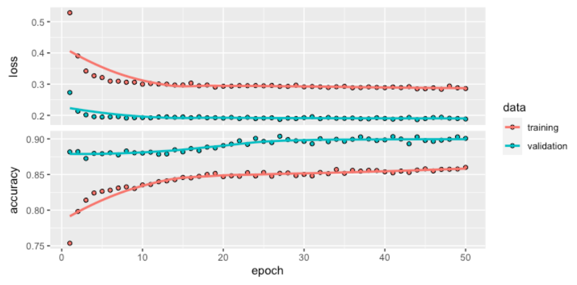
Table 4: Models with different number of neurons in layers

model	unit composition	loss	accuracy
neuron1	8, 4, 2, 2	0.2670	0.8641
neuron2	16, 8, 4, 2	0.2565	0.8701
baseline	32, 16, 8, 2	0.2524	0.8731
neuron3	64, 32, 16, 2	0.2523	0.8737
neuron4	128, 64, 32, 2	0.2553	0.8723

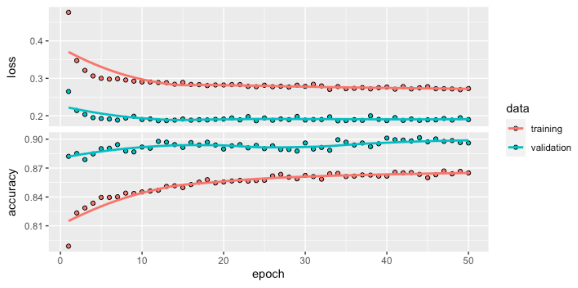
8 neurons



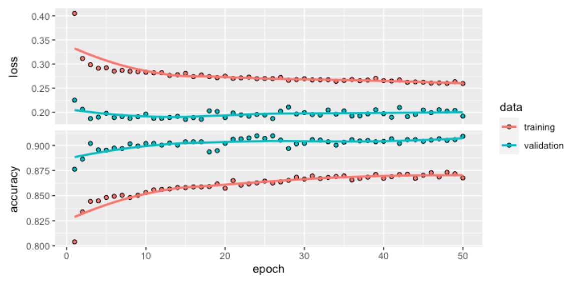
16 neurons



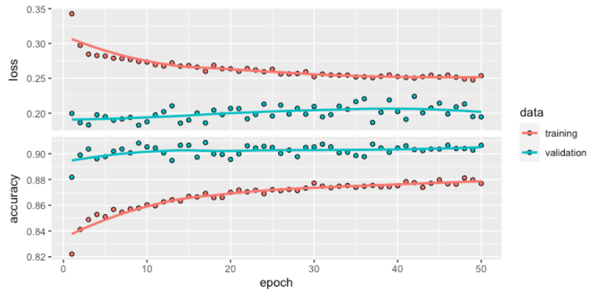
32 neurons



64 neurons



128 neurons



Hypothesis A: A higher number of neuron representations should be suitable for solving non-linear problems but increases the probability of possible overfitting

The accuracy values show a slight increase as the number of neurons increases from 8 neurons to 64 neurons, indicating that a higher number of neuron representations might be beneficial for solving non-linear problems. However, with 128 neurons, the accuracy slightly decreases, suggesting that further increasing the number of neurons could potentially lead to overfitting. Overfitting refers to a situation where the model becomes too closely tuned to the training data and performs poorly on new, unseen data It occurs when the model captures noise or irrelevant patterns in the training data instead of learning the underlying patterns that generalize well to other data (Chollet & Allaire, 2018). Looking at the accuracy, overfitting can be inferred if the training accuracy keeps increasing, but the validation accuracy saturates or even decreases. A growing gap between the training and validation accuracy curves implies that the model is becoming overly specialized in the training data and may not perform well on new data. The graphs above show that the highest number of neurons (128) has an increasing training accuracy but a stagnating validation accuracy, which could infer overfitting. When higher numbers of neurons are used in a neural network, the model has a larger capacity to memorize and fit the training data. This increased capacity allows the model to potentially capture more intricate details and complex relationships within the training set. However, if the number of neurons becomes too high, the model may start to learn and represent noise or idiosyncrasies specific to the training data, rather than generalizing to new data. This leads to a decrease in performance on unseen data, resulting in lower accuracy.

Hypothesis B: A lower number of neurons in a layer should support generalization and avoids overfitting but they are not suitable for non-linear problems

The results partially support this hypothesis as well. The accuracy values show relatively consistent performance across 8 neurons to 64 neurons, ranging from 0.8641 to 0.8737, which suggests that a lower number of neurons in a layer can support generalization. Generalization refers to the ability of a model to perform well on unseen data beyond the training set (Chollet & Allaire, 2018). Looking at the convergence of both loss and accuracy curves over time as can be seen in the graphs above, both training and validation loss curves seem to stabilize and remain relatively consistent. By using a lower number of neurons, the model has fewer parameters and, therefore, less capacity to memorize specific details of the training examples. This constraint can help prevent the model from fitting noise or irrelevant patterns in the training data, thus reducing the risk of overfitting. Although the results don’t show a significant effect, a lower number of neurons might not be suitable for non-linear problems because non-linear problems often involve intricate interactions and complex relationships between input features. These interactions may require the model to capture and represent highly nonlinear patterns and decision boundaries. With a lower number of neurons, the model's capacity to learn and express these complex non-linear relationships is limited. The model may struggle to capture the intricacies and nuances present in the data, leading to reduced accuracy. Therefore, for non-linear problems, it is often necessary to have a higher number of neurons or a more complex model architecture that can better capture and represent the non-linear nature of the data.

3.4 Number of Parameters

Based on the combined results from the experiments with the number of layers and the number of neurons, we can observe the effects of changing the total number of parameters in a neural network on both computation time and accuracy.

For computation time, expressed in system time, the computation time generally tends to increase as the total number of parameters increases. This is because more parameters require more computations during forward and backward propagation, resulting in longer training and inference times. Furthermore, more parameters generally require more computational resources, such as memory and processing power, leading to increased computation time. By decreasing the model's complexity and the number of connections, the computational requirements can be reduced, resulting in faster training and inference. However, there may be exceptions where the relationship between parameters and computation time is not strictly linear due to other factors such as hardware capabilities.

As for performance in terms of accuracy, it is apparent that the accuracy of the neural network is not solely dependent on the total number of parameters. Other factors like network architecture, dataset, and training process also play crucial roles. In the table, we can see that the accuracy fluctuates across different network configurations despite changes in parameter count. Generally, increasing the number of parameters can potentially improve accuracy by allowing the model to capture more complex patterns and relationships in the data. However, there is a diminishing return effect, where adding more parameters may not significantly increase accuracy or even lead to overfitting. The accuracy reaches a peak and then starts to decline in some cases (e.g., 1290 parameters, 3 layers). This indicates that there is an optimal parameter range for achieving the best accuracy.

Table 5: Models with different parameters

parameters	layers	unit composition	system time (s)	loss	accuracy
204	3	8, 4, 2, 2	3.115	0.2670	0.8641
486	3	16, 8, 4, 2	4.616	0.2565	0.8701
674	1	32, 2	3.522	0.2520	0.8697
1170	2	32, 16, 2	6.401	0.2517	0.8723
1290	3	32, 16, 8, 2	2.583	0.2524	0.8731
1462	4	32, 16, 12, 8	4.897	0.2541	0.8721
1462	5	32, 16, 12, 8, 4, 2	2.727	0.2574	0.8711
1958	6	32, 20, 16, 12, 8, 4, 2	3.122	0.2539	0.8699
2802	7	32, 28, 20, 16, 12, 8, 4, 2	4.329	0.3066	0.8454
3858	3	64, 32, 16, 2	6.540	0.2523	0.8737
12834	3	128, 64, 32, 2	6.181	0.2554	0.8723

3.5 The Structure of the Neural Network

To investigate the impacts of varying network structure on performance, we devised two different neural networks:

1. Pyramid-Shaped Neural Network: This architecture involves an input layer, followed by three hidden layers, with neurons decreasing in a pyramidal fashion - 32 neurons in the

first hidden layer, 16 in the second, and 8 in the third. There is also an output layer in this configuration. The total number of parameters in this network is 1290.

2. Uniform Layer Neural Network: Unlike the pyramid shape, this network has a uniform architecture, meaning each hidden layer contains the same number of neurons. In this case, each hidden layer consists of 20 units. However the total number of parameters can only be almost the same with that of the Pyramid-Shaped Neural Network. The total count of parameters in this network structure stands at 1,262.

The computational cost and accuracy results of the two neural network models are as follows:

1. Pyramid-Shaped Neural Network:
   - Computational cost (system time): 4.13 units
   - Loss: 0.2512875
   - Accuracy: 87.34574%
2. Uniform Layer Neural Network:
   - Computational cost (system time): 2.28 units
   - Loss: 0.2559346
   - Accuracy: 86.91479%

From these results, it's clear that the pyramid-shaped neural network yields slightly better accuracy (87.35% vs 86.91%) and a marginally lower loss value (0.251 vs 0.256) compared to the uniform layer neural network. However, it does so at a higher computational cost, taking almost twice the time to run, 4.13 over 2.28 units.

Therefore, the choice between these two architectures depend on specific requirements. If the task prioritizes model accuracy and are less concerned about computational time, the pyramid-shaped neural network is preferred. Conversely, if reducing computational cost is paramount—for instance, if the task operates in a real-time system or under hardware constraints— uniform layer network is preferred, despite the small drop in accuracy.

3.6 Effect of Dropout

The baseline model has dropout rates of 0.3, 0.2, and 0.1 for the three layers respectively. The model dropout1 has higher dropout rates of 0.5, 0.2, and 0.1 for the three layers. The model dropout2 has the same dropout rates as dropout1, except for a higher dropout rate of 0.3 for the second layer. The model dropout3 has higher dropout rates of 0.6, 0.3, and 0.1 for the three layers. The model without dropout does not utilise any dropout, as indicated by dropout rates of 0 for all three layers. Their performances are shown in Table 6 and compared as follows:

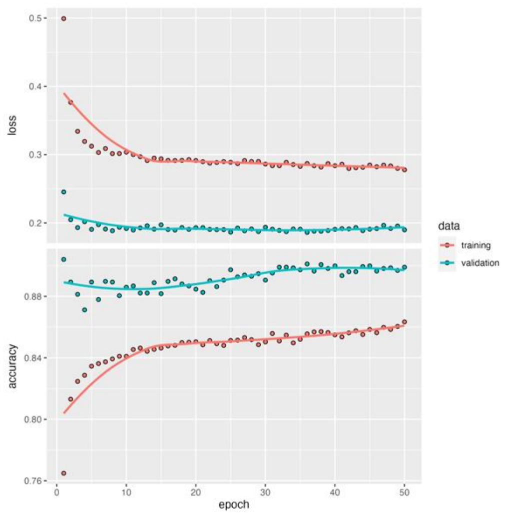
1. **Computation Time:** The baseline model has the longest computation time of all models. Dropout1 shows improved efficiency, Dropout2 and Dropout3 have slightly longer computation times than Dropout1, and the model without dropout exhibits the highest efficiency among all models with the lowest computation time of 4.924, significantly lower than the baseline model's 8.284.
2. **Accuracy:** Comparing all the models with the baseline model, Dropout1 shows a slight improvement in accuracy while maintaining a similar loss value, while Dropout2 and Dropout3 exhibit slightly lower accuracy with similar or slightly higher loss values, and the model without dropout has a higher loss value and lower accuracy.

Table 6: Models with different dropout rates in layers

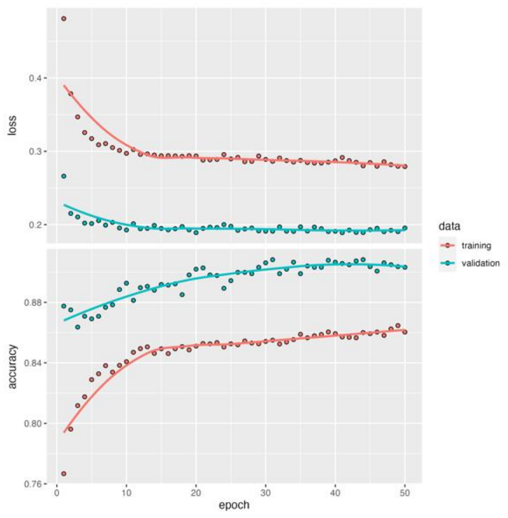
model	dropout rate	system	loss	accuracy
baseline	0.3, 0.2, 0.1	8.284	0.2518	0.8715
dropout1	0.5, 0.2, 0.1	4.924	0.2536	0.8719
dropout2	0.5, 0.3, 0.1	5.245	0.2546	0.8703
dropout3	0.6, 0.3, 0.1	5.663	0.2544	0.8693
without dropout	-	4.483	0.2759	0.8676

There is no significant evidence of overfitting in the models except for the model without dropout, as its training loss and validation loss behave differently from each other at later epochs. Considering both accuracy and computation time, dropout1 emerges as the preferred choice with a slightly improved accuracy compared to the baseline model and significantly reduced computation time. The result demonstrates that the dropout rates of layers can indeed affect both training time and model performance in terms of accuracy. By slightly increasing the dropout rate of the first layer, the training time can be decreased while the accuracy can be improved. However, one should note that sometimes merely increasing the dropout rate of a layer does not necessarily improve the performance in any sense (computation time, loss, and accuracy). When deleting all the dropout layers, even though the computational cost decreases for a small amount, the performance in terms of loss and accuracy also worsen.

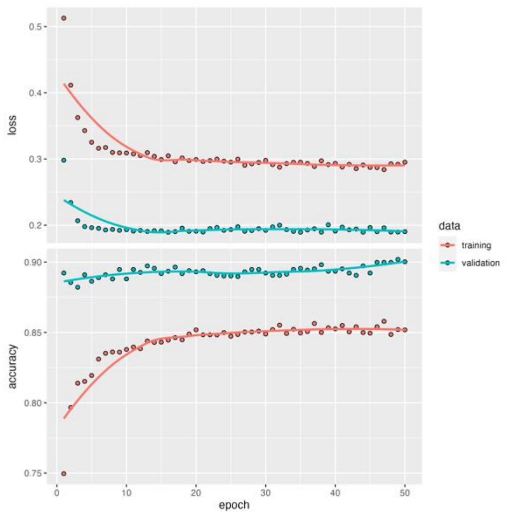
dropout1



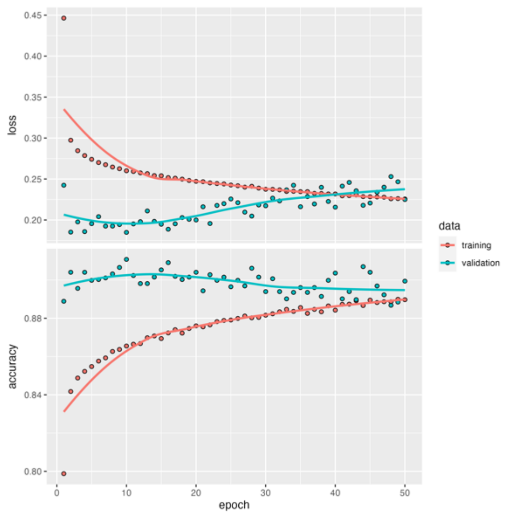
dropout2



dropout3



without dropout



3.7 Effect of Class Imbalance

Our original dataset is inherently imbalance as the target variable *Categorical Rating Count* has 3786 value high and 13197 value low. To observe the effect of class imbalance on the performance of neural network, two experiment settings were created as follows.

3.7.1 Experiment Setting 1

In this setting, we modify the baseline model to specifically address the class imbalance issue by assigning weights to observations from different classes. This is accomplished by employing the class_weight parameter, where we assign a weight of 1 to "low" and a weight of 3 to "high" instances. These weights reflect the approximate imbalance ratio in the classes. We then compare the performance of this model against the baseline model when run on the original dataset.

We observe the following results:

1. Base Line Model (without class weighting):
    - Computational cost (system time): 4.13 units
    - Loss: 0.2512875
    - Accuracy: 87.34574%
2. Modified Base Line Model (with class weighting):
    - Computational cost (system time): 2.15 units
    - Loss: 0.2881563
    - Accuracy: 85.19099%

From these results, we can see that the base line model (without class weighting) outperforms the modified model (with class weighting) in terms of accuracy (87.35% vs 85.19%) and loss (0.251 vs 0.288). However, the modified model has a significant reduction in computational cost, with its system time almost halved compared to the base line model.

It seems that while adding class weighting has reduced computational time, it has not improved the model performance in terms of accuracy and loss. This suggests that although class weights can be useful for addressing class imbalance, they may not always lead to improved model performance, particularly if the classes are not significantly imbalanced or if the model is complex.

3.7.2 Experiment Setting 2

The second setting aims to balance the training set by resampling the dataset. Here, we randomly sample 3300 rows each of "low" and "high" values from the "Categorical Rating Count." This reduces the risk of the model being biased towards the "low" value class. The baseline model is then run on this balanced dataset and its performance compared against its performance on the original dataset. However, this approach has the potential downside of losing potentially significant information by excluding approximately 10000 observations with "low" values. Furthermore, the reduction in the number of data points may cause the comparison of system time to be less meaningful, as the computational cost may decrease simply due to the smaller dataset size.

We observe the following results:

1. Base Line Model (running on the original, imbalanced dataset):
    - Computational cost (system time): 4.13 units
    - Loss: 0.2512875
    - Accuracy: 87.34574%
2. Base Line Model (running on the balanced dataset):
    - Computational cost (system time): 1.22 units
    - Loss: 0.3009980
    - Accuracy: 88.19918%

From these results, we can see that balancing the dataset leads to a significant reduction in computational cost, with the system time reduced to about a third of the original. Furthermore, the accuracy of the model on the balanced dataset is slightly higher than on the original dataset (88.20% vs 87.35%). However, the loss value is higher on the balanced dataset (0.301 vs 0.251), indicating that the model makes larger errors on average.

3.8 Number of Training Samples

Different models are compared based on different proportion rates (0.7, 0.8, 0.85, 0.9, and 0.6). By varying the proportion rate, we assess how the size of the training set impacts model performance and computational efficiency. Generally, increasing the proportion rate leads to a larger training set, which can potentially improve model accuracy by allowing for more data to be used for training. Furthermore, considering that increasing the size of the training set provides more data for the model to learn from, and that increasing the batch size when enlarging the size of the training set is a general idea, we also examine the effect of batch size on the training efficiency. the performances of the models are shown in Table 7.

1. **Computation time:** within the expectation, size4 with the lowest proportion split rate of 0.6 in the test is the most efficient in terms of computation time among all other models. Also as the batch size increases from 32 to 64, the training efficiency increases.
2. **Accuracy:** Comparing all the models to the baseline model, size1, size2, size3, and size4 exhibit slight variations in accuracy and loss values, while size2 shows the highest accuracy (0.8783) among all models, indicating potential improvements over the baseline.

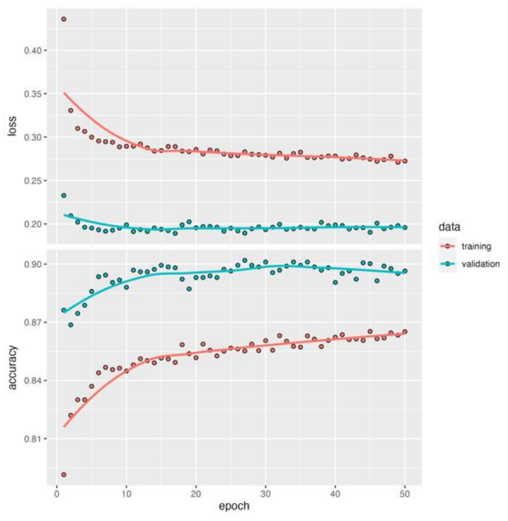
Table 7: Models with different proportion of training set

model	proportion of training set	system	loss	accuracy	batch size
baseline	0.7	8.284	0.2518	0.8715	32
size1	0.8	6.760	0.2519	0.8697	32
size2	0.85	5.767	0.2660	0.8783	32
size3	0.9	5.733	0.2571	0.8674	32
size4	0.6	3.923	0.2554	0.8689	32
batch	0.9	3.457	0.2583	0.8627	64

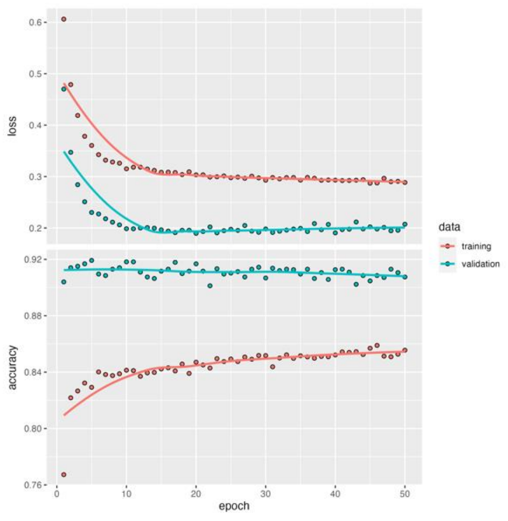
There is no significant evidence of overfitting in the models. Considering both accuracy and computation time, there is a trade-off between performance and efficiency, which indicates that the selection of the size of the training set can help balance performance and efficiency. Although the model size4 with a smaller sample of training set performs slightly worse than the baseline model, its cost in terms of computation time is significantly less than that of the baseline model, indicating the potential trade-off between accuracy and computational costs. The models with larger proportion rates (Size1, Size2, and Size3) have similar or slightly higher accuracy values compared to the baseline model, suggesting that increasing the size of the training set can have a positive impact on performance.

Therefore, in scenarios where computational resources are limited or time constraints are a priority, a smaller sample can potentially be a practical choice in the real setting. Moreover, it is possible that a proportion would improve the performance in terms of both training time and accuracy, verified by the result that model size2 performs better in any situation than the baseline model. Importantly, one should note that a larger size of the training set does not necessarily lead to a better performance in terms of accuracy.

size1

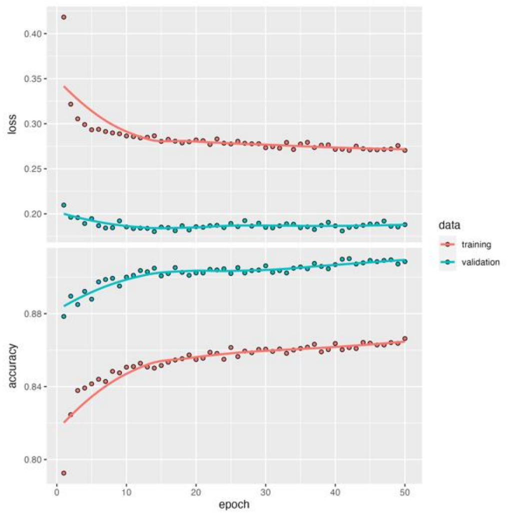


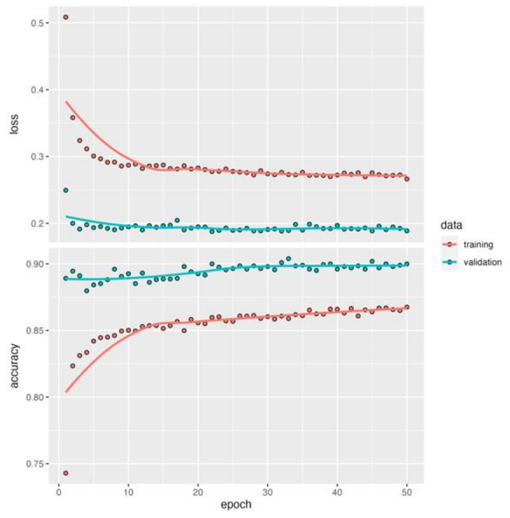
size2



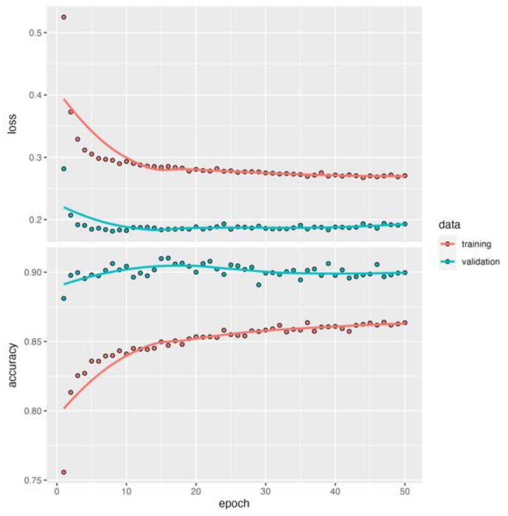
size3

size4





batch



4. Conclusion

To conclude, this comprehensive analysis on parametrization in neural networks illuminates how each aspect influences computational cost and model performance. The number of epochs, hidden layers, and parameters directly impact both time complexity and model accuracy, showcasing a delicate balancing act between optimizing performance and maintaining efficiency. Our examination further revealed that dropout rates and the size of training samples significantly affect model performance and computational cost, often in a non-linear fashion.

The number of neurons, while integral in solving complex non-linear problems, had its optimal threshold beyond which it led to overfitting, thus stressing the importance of cautious selection. Additionally, the total number of parameters, while influencing computation time and accuracy, did not exhibit a strictly linear relationship, further underscoring the intricate dynamics of model optimization.

Our exploration of the effects of class imbalance revealed the subtle trade-off between accuracy and loss when mitigating class imbalance. This highlights the need for careful selection and testing of different strategies to address this issue. Lastly, comparing different neural network structures emphasized the role of architecture in model performance and efficiency, rendering the selection of network structures as a context-dependent decision.

This analysis has demonstrated the complex nature of optimizing a neural network's parameters, where a multitude of factors must be taken into consideration. While maximizing model accuracy remains a primary objective, this must be carefully balanced with computational cost, model complexity, and efficiency. Ultimately, it underscores that there is no one-size-fits-all solution, and careful tuning and consideration of each parameter are essential to optimize the performance of a neural network for any given application. Each experiment provides insights that are valuable for understanding how various parameters influence neural networks, thus providing essential guidance for future optimizations and implementations in the field of machine learning and beyond.

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).

Chollet, F., & Allaire, J. J. (2018). Deep Learning with R. Manning Publications.