华中科技大学
计算机科学与技术学院
School of Computer Science &Technology，HUST

COMPUTER VISION

Experimental Report 1

CS2110 U2021XXXXX Gao Lang
Huazhong University of Science and Technology

1 Introduction

1.1 Background

Deep Learning (DL) is a specialized subfield of Machine Learning (ML) that employs algorithms with multiple processing layers to learn representations from data.DL is based on Artificial Neural Networks (ANNs), which are computational systems designed to mimic the functioning of the human brain.

The history of DL can be traced back to the 1943 paper by W. Pitts and W. McCulloch that introduced the mathematical model of biological neurons (McCulloch and Pitts 1943). Approximately a decade later, F. Rosenblatt created a new version of the McCulloch-Pitts neuron called "Perceptron" (Rosenblatt 1957), which had the ability to learn binary classification. This event marked the beginning of AI.[1]

In the 1980s, the Multi-Layer Perceptron (MLP) was invented, which is a type of feed-forward artificial neural network. It is characterized by having multiple hidden layers, which allows the model to overcome the limitation of the perceptron that can only handle linear classification problems. By adding activation function layers in the hidden layers, the nonlinearity of the model is further enhanced. The MLP can be trained using the backpropagation algorithm, allowing it to solve more complex nonlinear problems.Figure 1 reveals structure of this type of model.[2][3]

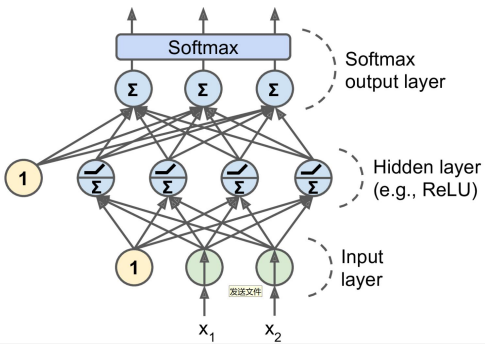


Figure 1:Feed-forward Neural Network

1.2 Experimental Objective

Design a feed-forward neural network to perform classification on a set of data.

Download the `"dataset.csv"` dataset, which contains 4 types of two-dimensional Gaussian data and their labels. Design a feed-forward neural network with at least 1 hidden layer to predict the classification of the 2-dimensional Gaussian samples. This dataset needs to be randomly sorted first, then 90% is used for training and the remaining 10% is used for testing.

2 Framework

In this experiment, we used the *Keras* framework supported by *TensorFlow* to build the neural network. Appendix A shows the implementation details of building a feed-forward neural network using the *Keras* framework.

When actually constructing the model, we used the *softmax* activation function in the output layer to simulate the probability distribution of a 4-class classification problem. At the same time, considering the 4-class classification task that the model needs to solve, we chose *sparse_categorical_crossentropy* as the loss function and *sgd* as the optimizer for the experiment.

In addition, in order to explore the impact of different modules of the model on the overall prediction effect, we changed the number and dimensionality of the hidden layers, the activation functions of the hidden layers, and the batch size during training, and conducted various experiments for testing. Details can be found in Section *3 Experiments*.

3 Experiments

3.1 Experimental setup

Datasets and Models In the provided dataset, we randomly selected 90% of the data as the training set and the remaining 10% as the testing set.We further extracted 10% of the training set as the validation set. In this experiment, we used four different model architectures: *Model A* has 2 hidden layers with a dimension of 16.*Model B* has 2 hidden layers with a dimension of 32.*Model C* has 1 hidden layer with a dimension of 32. *Model D* has 3 hidden layers with a dimension of 16.These four models formed the following sets of control experiments: Model groups A and D, as well as Model groups B and C were used to explore the impact of the number of hidden layers on the prediction accuracy of the model; Model groups A and B were used to explore the impact of the dimension of hidden layers on the prediction accuracy of the model.

Metrics All the methods get similar predictive accuracy.In order to more fully reflect the experimental results, we used precision(P), recall(R), and F1 score($F1$) as evaluation metrics besides accurcy(Acc).

Implementation details The output layer of these four models in the experiments is a 4-dimensional linear layer with softmax activation function. During testing, each model was trained for 200 epochs.All experiments in this report were completed on a CPU.

3.2 Experimental Results

Table 1 reflects the impact of changing the activation function and batch size of hidden layers on the prediction performance of different models.

Table 1:Experimental results on the dataset using different model hyper-parameters.The bolded data in each row represents the optimal model parameters. The criteria are as follows: the model is considered optimal when it has the highest $F1$, and when $F1$ are the same, the model with fewer parameters is considered better.

Actv	Bs	Model A				Model B				Model C				Model D			
		Acc	P	R	$F1$	Acc	P	R	$F1$	Acc	P	R	$F1$	Acc	P	R	$F1$
Sig moid	8	0.91	0.91	0.91	0.90	**0.91**	**0.91**	**0.91**	**0.91**	0.91	0.90	0.91	0.90	0.91	0.91	0.91	0.91
	16	**0.90**	**0.90**	**0.91**	**0.90**	0.90	0.90	0.90	0.89	0.90	0.90	0.90	0.90	0.88	0.89	0.88	0.88
	32	0.90	0.90	0.90	0.89	0.90	0.90	0.90	0.90	**0.91**	**0.91**	**0.91**	**0.91**	0.83	0.83	0.83	0.83
	64	0.82	0.82	0.82	0.82	0.89	0.89	0.89	0.88	**0.89**	**0.90**	**0.90**	**0.89**	0.79	0.81	0.79	0.78
Relu	8	0.92	0.92	0.92	0.92	**0.93**	**0.93**	**0.93**	**0.93**	0.92	0.91	0.92	0.91	0.92	0.92	0.92	0.92
	16	0.92	0.92	0.92	0.92	0.92	0.92	0.92	0.92	0.92	0.92	0.92	0.92	**0.93**	**0.93**	**0.93**	**0.93**
	32	0.92	0.92	0.92	0.92	**0.92**	**0.92**	**0.93**	**0.92**	0.92	0.92	0.92	0.92	0.91	0.91	0.91	0.91
	64	0.92	0.92	0.92	0.92	0.92	0.92	0.92	0.92	0.91	0.91	0.91	0.91	0.92	0.92	0.92	0.92
L-Relu	8	**0.93**	**0.93**	**0.93**	**0.93**	0.90	0.90	0.90	0.90	0.92	0.92	0.92	0.92	0.93	0.93	0.93	0.93
	16	**0.93**	**0.93**	**0.93**	**0.93**	0.93	0.93	0.93	0.93	0.92	0.92	0.92	0.92	0.91	0.91	0.91	0.91
	32	0.92	0.92	0.92	0.92	0.92	0.92	0.92	0.92	**0.92**	**0.92**	**0.92**	**0.92**	0.92	0.92	0.92	0.92
	64	**0.92**	**0.92**	**0.92**	**0.92**	0.92	0.92	0.92	0.92	0.91	0.90	0.91	0.90	0.92	0.92	0.92	0.91
tanh	8	0.92	0.92	0.92	0.92	0.92	0.92	0.92	0.91	**0.92**	**0.92**	**0.92**	**0.92**	0.91	0.91	0.91	0.91
	16	0.91	0.91	0.91	0.91	0.92	0.92	0.92	0.92	**0.92**	**0.92**	**0.92**	**0.92**	0.92	0.92	0.92	0.92
	32	**0.92**	**0.92**	**0.92**	**0.92**	0.92	0.92	0.92	0.92	0.91	0.90	0.91	0.90	0.92	0.92	0.92	0.92
	64	0.91	0.91	0.91	0.91	0.91	0.91	0.91	0.91	**0.91**	**0.91**	**0.91**	**0.91**	0.91	0.91	0.91	0.91

Impact of the number of hidden layers In the 16 sets of control experiments with the same loss functions and batch sizes for models A and D, there were a total of 10 instances where $F1$ were not equal. Among these instances, model A had a higher $F1$ in 7 cases compared to model D. This suggests that in this experiment, having more parameters is not always better for the model. When using Sigmoid and Relu as activation functions, model D achieved better results than model A. This indicates that when the number of parameter updates is consistent and sufficient, having more parameters gives an advantage.

For models B and C, there were 11 instances where $F1$ were not equal, with model B having a higher $F1$ in 7 instances. This implies that in this experiment, having fewer parameters is not always better for the model. When using Relu and Leaky-Relu as activation functions, model B achieved better results than model C. This suggests that when the number of parameter updates is consistent and sufficient, having more parameters provides an advantage.

Impact of the dimension of hidden layers In the 16 sets of control experiments with the same loss functions and

batch sizes for models A and B, there were a total of 8 instances where $F1$ were not equal. Among these instances, model B had a higher $F1$ in 5 cases compared to model A. Additionally, both model A and model B achieved an $F1$ of 0.93 on separate occasions. This suggests that in this experiment, having more parameters under the same conditions has a positive impact on the prediction performance.

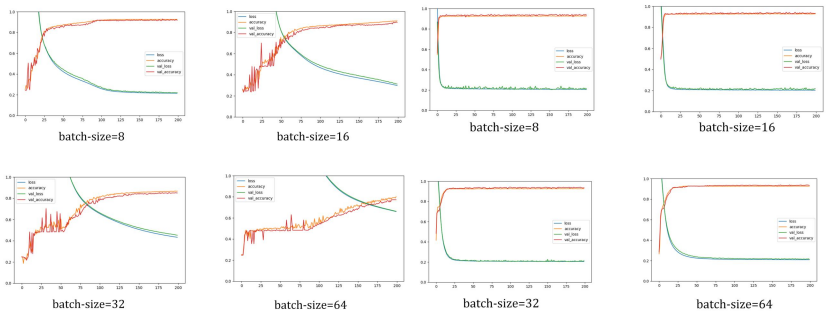
Impact of batch sizes We surprisingly found that using a batch size of 64 rarely produces the optimal results when employing the same model. For almost all models, the optimal results are concentrated on either a batch size of 8 or a batch size of 16. One possible explanation for this is that while using a larger batch size ensures faster convergence of gradients, it also leads to a slower frequency of model parameter updates. Due to the limited size of the dataset, the model may not undergo sufficient parameter updates after training, resulting in underfitting.

Impact of activation functions Based on the available experimental data, it is not possible to draw valuable conclusions about the overall testing effect of different activation functions for all models. However, the choice of activation function can have an impact on the overall performance for different model architectures. For example, when using Leaky-Relu, model A achieved global optimality in 3 experiments, which is higher than the sum of times it achieved global optimality under other activation functions. Similarly, when using tanh, model C achieved global optimality in 3 experiments, which is also not lower than the sum of times it achieved global optimality under other conditions. This suggests that different model scales are suitable for different activation functions.

4 Discussion

In the following parts, we will explore some issues arising from the experimental data.

Why did model D perform poorly? As observed in the experimental data, model D had a very low F1 score when using the sigmoid activation function. Model D is the largest among the four models, with more parameters and hidden layers, which results in slower convergence in certain situations. As shown in Figure 2(a), the loss of model D decreased significantly slower when using the sigmoid function as the activation function, leading to non-convergence after 200 epochs and thus a low F1 score. When using the Relu function, as shown in Figure 2(b), the gradient of model D quickly reached saturation, and training for 200 epochs no longer resulted in significant decreases. Due to its large number of parameters, it is reasonable to believe that model D suffers from overfitting to some extent, which also contributes to its poorer performance compared to the smaller models on these datasets.



a.Sigmoid b.Relu

Figure 2:The loss,validation loss,accuracy and validation accuracy curves of model D using different activation functions.

Why did small-scale models and small batch sizes perform well in this experiment? This phenomenon

contradicts traditional views on the relationship between model scale and performance, as well as batch size and performance. The first main reason is the difficulty of the task: the model needs to complete a four-classification task for a two-dimensional vector. This task is very easy for artificial neural networks, and even traditional machine learning algorithms can solve it, so large-scale models may not necessarily outperform smaller models in this task. The second reason is dataset limitations. In this experiment, there are 4000 samples in total, and after 10% are extracted for testing, only 3240 samples are left for training. Large-scale models are prone to overfitting. In addition, when the dataset is insufficient, there is a trade-off between batch size and gradient update frequency: larger batches result in more accurate gradient descent, but the frequency of gradient descent decreases. For large models, the gradient converges very slowly in some cases, which also easily leads to underfitting. Smaller models have fewer parameters and faster gradient convergence, and they can better demonstrate their performance in limited data.

5 Conclusion

In this experiment, we implemented a four-classification model for a twoIn this experiment, we implemented a four-classification model for a two-dimensional vector, with a feedforward neural network as the main architecture. We conducted extensive experiments with different model structures and hyperparameters, and drew valuable conclusions: when dealing with simple problems, the number of parameters in the model is not the decisive factor, and small models can also achieve satisfactory results; batch size selection must follow the trade-off between batch size and gradient update frequency, and for different models, the choice of activation function is also important. Therefore, we encourage finding the most suitable activation function for the model through experimentation.

Reference

[1] Rosenblatt, F.: The Perceptron, a Perceiving and Recognizing Automaton Project Para. Cornell Aeronautical Laboratory (1957)

[2] Jackydu, J. (2021). A deep learning-based natural language processing technique for text classification and sentiment analysis. CSDN Blog. https://blog.csdn.net/Jackydu1/article/details/130668607

[3] Somani, A., Horsch, A., & Prasad, D. K. (2019). Interpretability in Deep Learning. Springer.

Appendix A

Implementation of MLP using *Keras* structure(here we use an example of a 3-layer MLP,using tanh as activation function and 64 as batch size).

```
import keras
model=keras.Sequential()
model.add(keras.Input(shape=(2,)))
model.add(keras.layers.Dense(16,activation='tanh'))
model.add(keras.layers.Dense(16,activation='tanh'))
model.add(keras.layers.Dense(16,activation='tanh'))
model.add(keras.layers.Dense(4,activation='softmax'))
# model.summary()
model.compile(optimizer='sgd',loss='sparse_categorical_crossentropy',metrics='accuracy')
history = model.fit(x_train, y_train, epochs=EPOCHS,batch_size=64, validation_data=(x_val, y_val))
```