Multilayer Perceptron Review

Xu Deng
Faculty of Engineering
University of Sydney
Sydney, Australia
xden5493@uni.sydney.edu.au
Student ID: 510307864

Yingbin Mo
Faculty of Engineering
University of Sydney
Sydney, Australia
yimo6410@uni.sydney.edu.au
Student ID: 510202271

Yiran Zhang
Faculty of Engineering
University of Sydney
Sydney, Australia
yzha5806@uni.sydney.edu.au
Student ID: 510556912

***Abstract*—Multiple classification has been a classic problem in machine learning, and multilayer neural networks are widely used today as a high-performance classification structure. Therefore from in this project, we build a multilayer neural network from scratch, aiming to achieve the task of classifying a given dataset. A total of 70,000 data sets were used in this experiment, of which 60,000 were set as the training set and 10,000 were set as the test set, and these data corresponded to 10 different categories (labels: 0-9). The core of this study is the comparison experiment of hyperparameters and the ablation experiment of functional modules. In the comparison experiments, we investigate the effect of hyperparameters such as learning rate, batch size, dropout rate on the accuracy of the model. In the ablation experiments, we analyse the effect of different functional modules on the performance of the model, such as Dropout, BatchNormalisation, etc. Compared to the baseline model, our optimal model achieves the highest accuracy in a shorter training time and the final test set accuracy is 55.65%.**

***Index Terms*—Multiple layer perceptron, Deep learning**

I. Introduction

With the deepening of deep learning research in the past ten years, deep learning has begun to play an increasingly important role in various fields. By establishing an artificial neural network structure, deep learning enables machines to learn any information and is successfully applied in multiple areas of life: stock forecasting, natural language processing, image recognition and classification, information screening and retrieval, signal processing, auxiliary decision-making, artificial intelligence, etc. Domain success has proven its worth.

Deep learning is a highly parallel information processing system with strong adaptive learning ability [1]. It is composed of interconnected connections between many nodes (or neurons). Each node represents a specific output function, called an activation function. The relationship between nodes represents a weighted value and a bias value for the signal passing through the connection [1]. These two parameters are the objective function parameters that the artificial neural network hopes to train through forward-propagation and back-propagation.

The data set used in this project comprises 60,000 data with 128 dimensions. A given label has ten unique categories corresponding to each data: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Also, the total data has been split into 50,000 training instances and 10,000 test instances.

This project aims to study the impact of each component and different parameters in a deep learning neural network (MLP) on model performance.

This research is significant because, in this project, the aim is to effectively explain the design process of deep learning networks while also evaluating the impact of different hyper-parameter metrics, as well as various functional building blocks, on the efficiency and performance of neural network models. This is helpful for readers to understand the characteristics and operation mode of neural networks from macro to micro.

II. Methodologies

This section mainly outlines the method of pre-processing the data before running the model. Then, it introduces the principles of different modules in MLP and explains their roles in deep learning neural networks.

A. Data pre-processing

Our experiments mainly use three data pre-processing methods: Mean Normalisation, Min-max normalisation and standardisation [2]. These three methods can effectively perform feature scaling on the input data. Feature scaling helps make the gradient descent in the neural network converge faster. It is also beneficial to the numerical stability of the numerical calculation.

1) Min-max normalisation: Min-max normalisation is a process of a linear transformation of the original input data. The processed result of the original input data falls into the [0,1] interval, and the features of the original input data are scaled to the same range. The formula is shown in Formula 1.

$$x_{normalization} = \frac{x_i - x_{Min}}{x_{Max} - x_{Min}} \tag{1}$$

2) Mean normalisation: Mean normalisation is the feature scaling of the original input data pair by computing the original input data minus the mean of its features divided by the difference between the maximum and minimum values in its features. The formula is shown in Formula 2:

$$x_{normalization} = \frac{x_i - \mu}{x_{Max} - x_{Min}} \tag{2}$$

3) Standardisation: When the original input data is centred by its mean and then scaled by the standard deviation, the data will follow a normal distribution with a mean of 0 and a variance of 1. This process is called Standardisation, also known as Z-score normalisation. The formula is shown in Formula 3:

$$z_{Standardisation} = \frac{x_i - \mu}{\sigma} \tag{3}$$

B. Neural network related principles

1) Forward Propagation: Forward propagation is the process by which intermediate variables of a neural network are computed and stored in sequence from the input layer to the output layer [3]. In this forward process, the input data is weighted and summed, biased and non-linearly activated to become the result of the output layer. As Figure 1,the hidden layer is responsible for receiving input data, and each neuron in the hidden layer adds nonlinear transformation to the input data through an activation function. Then, the output of the hidden layer of this layer, that is, the input of the subsequent layer, is obtained through the weighted sum error calculation.

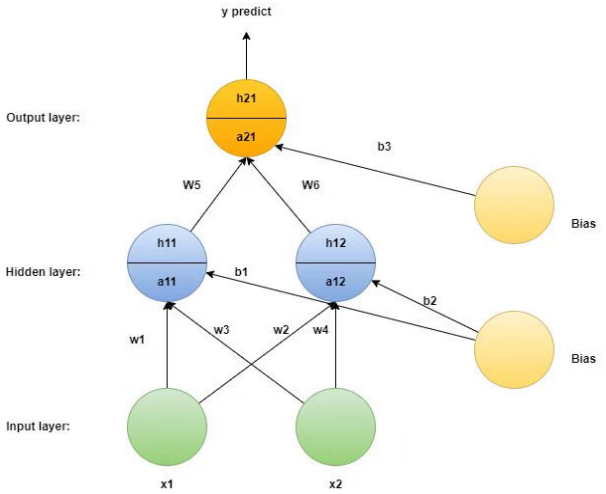

Fig. 1. Forward Propagation

2) Backward Propagation: Backward propagation plays a crucial role in training neural networks as it updates the parameters in the model (such as weights and biases) by minimising the value of the loss function, making the predictions closer to the actual values [4]. As Figure 2, this mechanism for updating parameters is based on gradient descent, which is calculated by computing the derivative of the loss function for each layer using the chain rule. The first back propagation is essentially to update the weights initialized by various initialization methods, and the subsequent parameter updates are through continuous iterative process until the final convergence.

3) Hidden layer (More than one, MLP): In a neural network, as figure 3, the hidden layer is located between the input and output layers of the neural network model [5]. The hidden layer consists of most neurons in the neural network and is the core part of processing data to obtain the desired

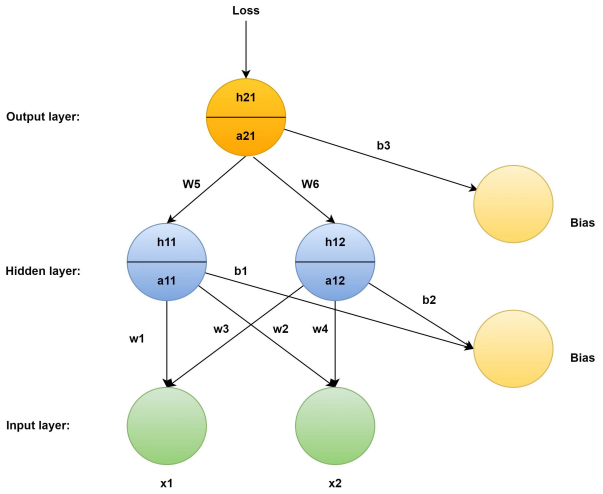

Fig. 2. Backward Propagation

output. The activation function of the hidden layer allows for the abstraction of the input features to enable the non-linear transformation of the data [5]. While multiple hidden layers allow for multiple levels of abstraction of the input features,as figure 4, with the ultimate aim of better linear segmentation of different types of data [5]. In this case, the neural network can be seen as a Multi-layer Perceptron. For example, a neural network with multiple hidden layers can classify the data according to the unique labels (0, 1, 2, 3, 4, 5, 6, 7, 8, 9) of this experiment.

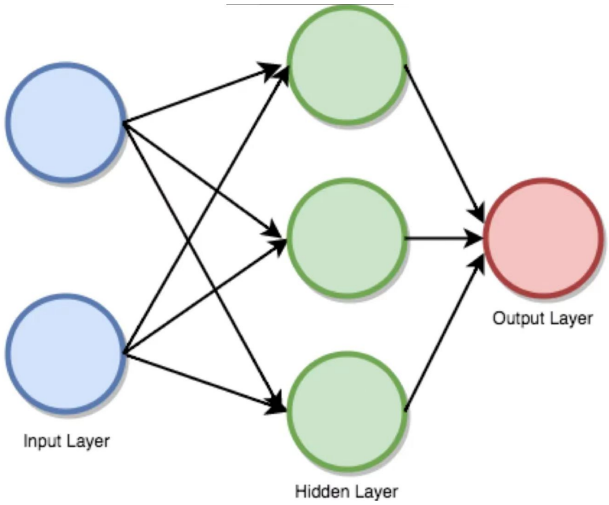

Fig. 3. Only one hidden layer

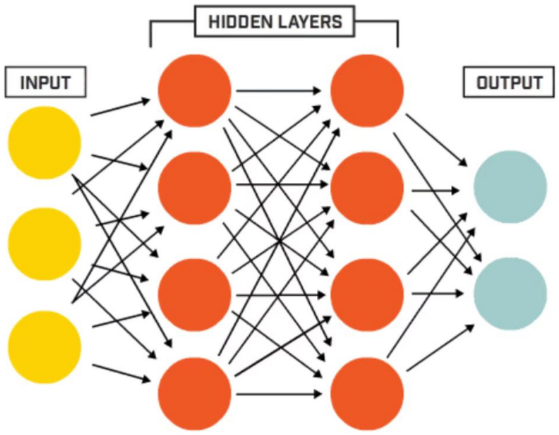

Fig. 4. More than one hidden layer

4) Activation function: The Activation Function is a function that runs on the neurons of the artificial neural network [6], and is responsible for adding nonlinear factors into our network model in mapping the input of the neuron to the output, as figure4. The neural network can approximate any nonlinear function arbitrarily so that the neural network can be applied to many nonlinear models [6]. If the activation function is not used, the output of each layer is a linear function of the input of the upper layer. No matter how many layers the neural network has, the result is a linear combination of the information. Activation Function mainly includes Tanh, ReLU, and LeakyReLU. They all meet the following three conditions: nonlinear, continuous, and derivable at any position, monotonic function. The Activation function is shown in figure 5.

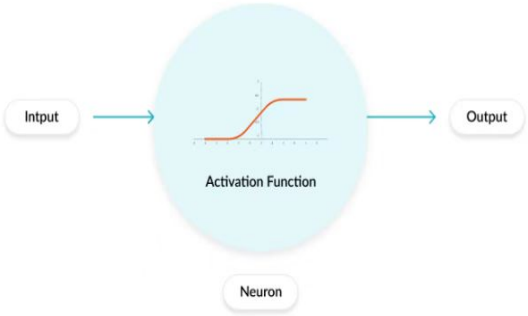

Fig. 5. Activation function

- **Tanh activation function:** Tanh, also known as the hyperbolic tangent function, is a nonlinear function in the range (-1, 1), as formula 4 and formula 5 and Figure 6 and Figure 7. It was proposed to solve the Sigmoid non-zero centre resulting in the slowing convergence problem, as its output value is averaged at 0. Moreover, the range of its derivatives is in (0, 1), so there is no need to worry about the gradient explosion caused by the cumulative multiplication calculation in back-propagation. However, as the diagram shows, the Tanh function is still bounded. When the input value is far from the origin, its output value will be infinitely close to the boundary, i.e. it still suffers from the gradient disappearance problem and is thus difficult to train.

$$Tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{4}$$

$$Tanh'(z) = 1 - Tanh(z)^2 \tag{5}$$

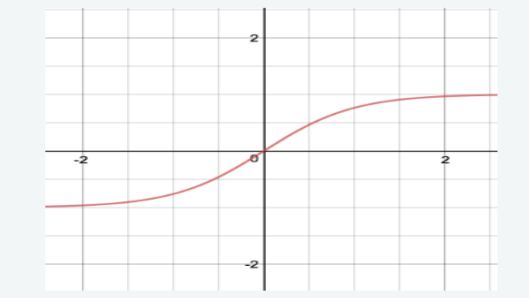

Fig. 6. Tanh activation function

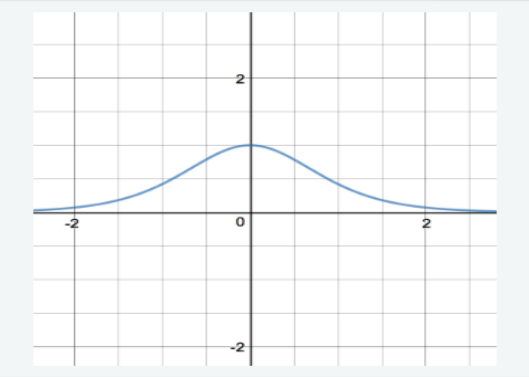

Fig. 7. Tanh derivative function

- **ReLU activation function:** Because the rectified linear activation function (ReLU) makes the model easier to fit, it is the default activation function for many types of neural networks [7]. As Formula 6 and Formula 7 and Figure 8 and Figure 9, ReLU is a linear and non-saturating function. When the input is greater than or equal to zero, the input and output are similar, otherwise, the result is zero. Furthermore, the derivative of the function is 1 when x is greater than 0, so there is no gradient vanishing problem [7]. But it also brings up another problem, which is gradient death. It can be seen that the function is hard saturated and its derivative is zero when the input value is less than 0. Once the input falls into this area, the neuron no longer updates its weight, a phenomenon known as neuronal death.

$$Relu(z) = max(0, z) = \begin{cases} 0 & \text{for} \quad z \leq 0 \\ z & \text{for} \quad z > 0 \end{cases} \tag{6}$$

$$Derivative f'(z) = \begin{cases} 0 & \text{for} \quad z \leq 0 \\ 1 & \text{for} \quad z > 0 \end{cases} \tag{7}$$

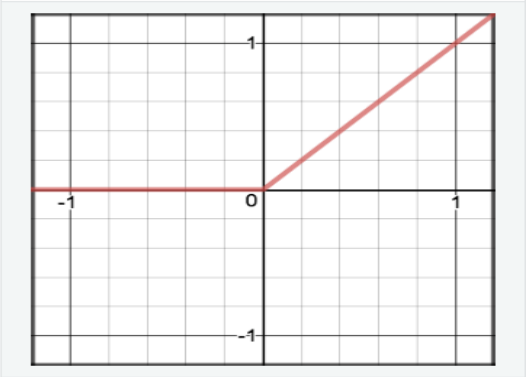

Fig. 8. ReLu activation function

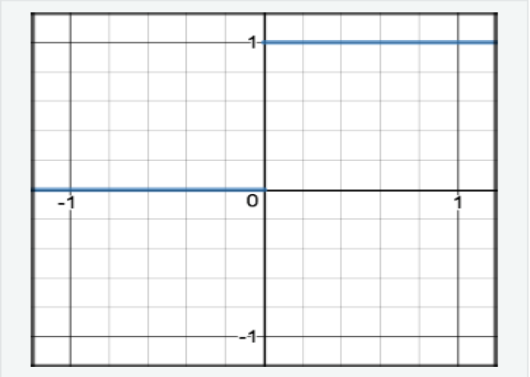

Fig. 9. ReLu derivative function

- **LeakyReLU activation function:** To address the drawback of the Relu function causing neuron death, a Leaky value can be introduced in the negative half interval of the Relu function, thus generating a new Leaky Relu function [8]. As Formula 8 and Formula 9 and Figure 10 and Figure 11, The Leaky ReLU (Linear Unit with Leakage Correction) function is a variant of the classic (and widely used) ReLu activation function, which has a small slope in the output for negative inputs [8]. Since the derivative is always non-zero, this reduces the appearance of silent neurons, allows gradient-based learning (although it can be slow), and solves the problem of the Relu function entering negative intervals, which causes neurons not to learn.

$$LeakyReLU(z) = \begin{cases} \alpha z & \text{for} \quad z \leq 0 \\ z & \text{for} \quad z > 0 \end{cases} \tag{8}$$

$$Derivative f'(z) = \begin{cases} \alpha & \text{for} \quad z \leq 0 \\ 1 & \text{for} \quad z > 0 \end{cases} \tag{9}$$

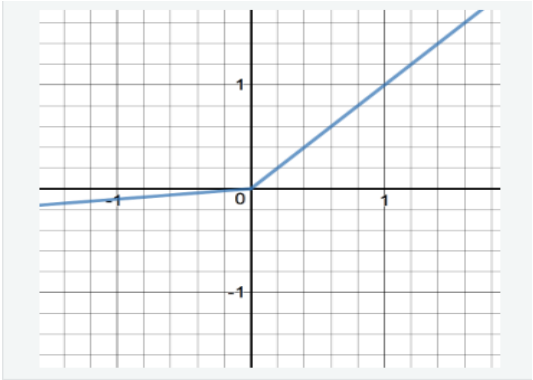

Fig. 10. LeakyReLU activation function

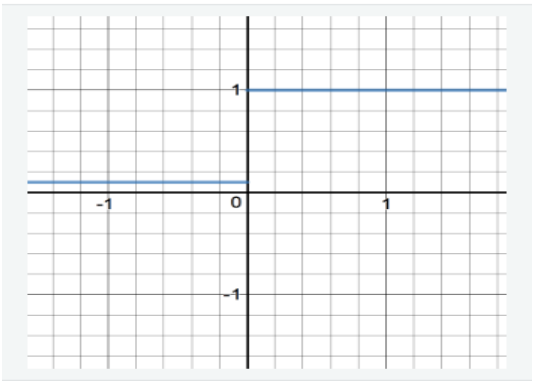

Fig. 11. LeakyReLU derivative function

5) Weight Initialization: The essence of the deep learning model training process is to update the weights in the model. The initialisation of the weights plays a crucial role in both the forward propagation and the subsequent propagation processes. Because the purpose of parameter initialisation is to allow the neural network to learn useful information during training, this means that the gradient of the parameters should not be 0 [9]. Using appropriate weight initialisation methods can avoid gradient scattering and gradient explosion [9]. Therefore, weight initialisation is a fairly important module in neural networks. We mainly use He initialisation and Xavier initialisation. Xavier initialisation will be used to initialise the weights when it is detected that the current hidden layer is using the Tanh activation function. The He initialisation is used to initialise the weights when the current hidden layer is found to be using the ReLu or LeakyReLU activation function.

- **Xavier initialization:** The idea of Xavier initialization is that assuming the activation function is Tanh, we want the variance and mean to remain the same in each hidden layer. This helps the model prevent the signal from exploding to high values or disappearing to zero. Although the derivation procedure for the Xavier initialization is based on a linear function, it was found to be effective in some nonlinear neurons in the study. The formula is shown in Formula 10.

$$xavier_{uniform} = \mu(-\frac{\sqrt{6}}{\sqrt{n^{l-1}+n^{l}}}, \frac{\sqrt{6}}{\sqrt{n^{l-1}+n^{l}}}) \quad (10)$$

- **He initialization:** The idea behind He initialization is that assuming we use the ReLU activation function, half of the neurons in each layer of the neural network are activated and the other half is 0. So, to keep variance constant, simply divide by 2 on top of Xavier's. Therefore, the weights are initialised keeping in mind the size of the previous layer, which helps to more quickly and efficiently obtain the global minimum of the cost function. The formula is shown in Formula 11.

$$kaiming_{uniform} = \mu(-\frac{\sqrt{6}}{\sqrt{n}}, \frac{\sqrt{6}}{\sqrt{n}}) \quad (11)$$

6) Optimiser: In the process of deep learning back-propagation, the optimiser guides each parameter in the model to update the appropriate size in the right direction, so that the updated parameters make the loss function value approach the global minimum continuously. However, the selection of optimiser and the setting of the learning rate will affect the optimal parameters of the model. For instance, if the learning rate is set too large or too small, the result will not converge. Therefore, a suitable optimizer and learning rate are crucial to the model fitting effect.

- **Momentum in SGD:** Momentum is also known as Stochastic Gradient Descent with momentum, which is an extension of traditional gradient descent [10]. SGD has trouble navigating ravines, i.e. areas where the surface curves much more steeply in one dimension than in another, which are common around local optima. Momentum is a method of optimising the gradient update by minimising the cost function J(θ) (making the predicted value closer to the actual value) to fit a higher performance model [10]. Momentum mainly achieves accelerated convergence by accelerating the gradient vector in the relevant direction. If the current gradient direction is consistent with the accumulated historical gradient direction, the previous gradient will be strengthened and the step will fall more sharply [10]. If the current gradient direction is inconsistent with the cumulative gradient direction, the magnitude of the current descending gradient will be attenuated. The Momentum formula is shown in Formula 12.

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta \nabla_\theta J(_\theta) \\ \theta_t &= \theta_{t-1} - v_t \end{aligned} \quad (12)$$

- **RMSProp:** RMSProp (Root Mean Square Prop algorithm) is similar to the Momentum method, both can accelerate gradient descent velocity. However, in some cases, if the feature varies in importance and frequency, the Momentum method will not work well as it assigns the same learning rate to all features [10]. Therefore, RMSProp, one of the Adaptive Learning Rate Methods was prompted. The core of RMSProp is adapting the learning rate to the parameters based on the past gradients that have been computed. Specifically, storing fixed previous squared gradients cannot accumulate to infinity and instead becomes a local estimate using recent gradients. Thus, the RMSProp method implements it as an exponentially decaying average of all past squared gradients [11]. Because the gradient of parameters will indirectly affect the learning rate, even if the parameters are close to the optimal value, the parameter update can still maintain a good effect, avoiding the problem that the parameter will not be updated when the learning rate is too small in the end [11]. The RMSProp formula is shown in Formula 13.

$$\begin{aligned} \theta_{t+1} &= \theta_t + \nabla\theta_t \\ \nabla\theta_t &= -\frac{\eta}{\sqrt{E[g^2]_t + \varepsilon}} g_t \\ E\left[g^2\right]_t &= \lambda E\left[g^2\right]_{t-1} + (1-\lambda) g^2 \end{aligned} \quad (13)$$

7) Regularisation: For the deep learning model, we not only require it to have a good fit to the training data set, but also expect it to have a good generalisation ability to the test set. However, the over-fitting problem often occurs because of the limitation of data amount. Therefore, regularisation, a method of modifying the learning algorithm, is proposed to reduce generalisation errors rather than training errors [12].

- **Dropout:** It is a simple but very effective technique that could alleviate over-fitting in the training phase. Dropout is to make the neurons stop working with a certain probability p during forward propagation, which can reduce the dependence of the model on some local features [13], as shown in the figure 12. However, all neurons will be used in the test phase and the weights are scaled down by a factor of p. The purpose of parameter scaling is to make the output values expected during training and testing the same. Therefore, dropout can be thought of as a combination of many different neural network predictions [14].

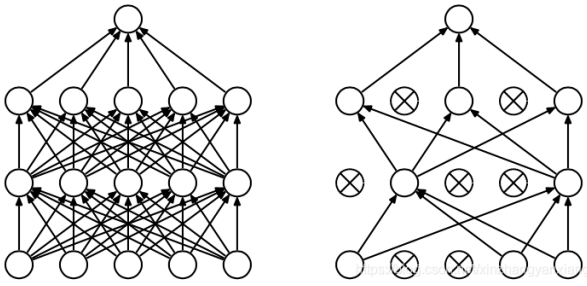

Fig. 12. Dropout

- **Weight decay:** To prevent over-fitting during model training, weight decay is used to penalise the complexity

of the neural network by limiting the growth of the weights in the network, i.e. to lighten the weights of some features [15]. Typically, a regular term (L2 regularisation) as a soft constraint is added to the original loss function, penalising large weights. In this project, weight decay is achieved in the optimiser"s updating parameters formula, penalising the weights and preventing over-fitting during model training. The Weight decay formula is shown in Formula 14.

$$min\hat{L}_R(\theta) = \hat{L}_R(\theta) + \frac{a}{2}[[\theta]]_2^2 \tag{14}$$

- **Batch Normalisation:** Batch Normalisation is an approach proposed in 2015. Although it is a relatively new method, it is already widely used by many researchers and personnel. As its name suggests, Batch Normalisation is based on the mini-batch in the learning process and normalises the batch of data [16]. In particular, it is the process of normalising the data distribution with a mean of 0 and a variance of 1, which display in formula 15. Then, the Batch Normalisation layer will scale and transform the normalised data in formula 16. γ and β are parameters corresponding to the effect of scale and transform. They are usually set to γ = 1 and β = 0 respectively and then adjusted to appropriate values by back-propagation. As for the benefits of implementing Batch Normalisation, it reduces covariate shift by converting each neuron activation to Gaussian distribution, avoiding the probability of exploding and vanishing gradients in the model [16]. In practice, it can also reduce the demand for regularisation, e.g. dropout or L2 norm. Because the means and variances are calculated over batches and therefore every normalised value depends on the current batch, i.e. the network can no longer just memorise values and their correct answers. Moreover, it allows for a higher learning rate in back-propagation as less danger of exploding or vanishing gradients. Furthermore, it enables training with saturating nonlinear function in deep networks, such as Sigmoid and Tanh function, due to the normalisation preventing them from getting stuck in saturating ranges [16]. Last but not least, since Batch Normalisation transformations are scalar invariants, Batch Normalisation Will can stabilise the growth of the parameter.

$$\begin{aligned} {}_B &= \tfrac{1}{m}\textstyle\sum_{i=1}^{M} x_i \\ \alpha_B^2 &= \tfrac{1}{m}\textstyle\sum_{i=1}^{M}(x_i - \mu_B)^2 \\ \hat{x}_i &= \tfrac{x_i - \mu_B}{\sqrt{\alpha_B^2 + \varepsilon}} \end{aligned} \tag{15}$$

$$y_i = \gamma\hat{x}_i + \beta \tag{16}$$

8) Other tricks: In addition to the key techniques described above, there are some more common but still important tricks in this experiment.

- **Softmax and Cross-entropy Loss:** As a generalisation of the two-classification, the Softmax function is often used in the multi-classification process. By normalising the outputs of multiple neurons to the (0, 1) interval as probabilities, the results of multi-classification are presented in the form of possibilities [17]. Cross-entropy loss is mainly used to measure the difference between two probability (true value and predicted value) distributions [18]. The Softmax function combined with cross-entropy loss adopts the competition mechanism between multi-class labels, and only cares about the accuracy of the prediction probability of the correct label, ignoring the difference of other incorrect labels [17]. The Softmax formula is shown in Formula 18.

$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)} \tag{17}$$

$$\begin{aligned} &-(y\log(p) + (1-y)\log(1-p)) \\ &-\textstyle\sum_{c=1}^{M} y_{o,c}\log(p_{o,c}) \end{aligned} \tag{18}$$

- **Mini-batch training:** The core idea of the Mini-Batch Training method is to ensure that the variance of each input data is not particularly large. This ensures that the parameters in the back-propagation are updated to the optimal values more quickly, further accelerating the convergence of the model [19]. First, the batch size is set and then the data set is cut according to the size of the batch so that the number of times the model is trained is equal to the length of the total data set divided by the value of the batch size.
- **One Hot Encoding:** One hot encoding mainly uses an N-bit state register to encode N states and converts the class to a binary value, making it easier for machines to recognize [20]. For instance, as shown in the figure 13, there are five different categories in the data. Thus, each encoded data has a length of 5 and the index corresponding to its true category is only 1(the first data belongs to the cat class and its index [0] is 1), otherwise 0. In this experiment, One Hot Encoding is mainly used as a tool with Softmax function and cross-entropy loss.

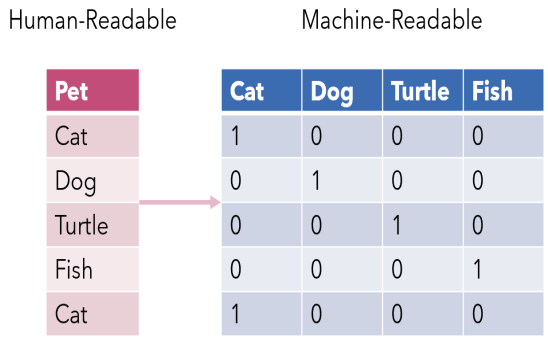

Fig. 13. One-hot encoding example

C. Best Model Architecture

Our best models come from an ablation study. We obtained the best model after adjusting the parameters of the model from the ablation study. For the best model, We first pre-process the data by standardization. We then built two hidden layers with 512 and 256 neurons. We used Kaiming for weight initialization and used the LeakyReLU as the activation function. For back-propagation, we used the SGD optimizer with a 0.001 learning rate and cross-entropy as the loss function. Next, batch Normalization and dropout methods were added as regularization methods between hidden layers, where the dropout rate is 0.5. Compared with the baseline model, the adjusted best model improved the accuracy by approximately 7.41, which is 55.65%, and reduced loss by approximately 2.2113, which is 1.2483. The best model runs 42 epochs to achieve the highest accuracy, which takes a total of 326 seconds. The comparison between the best model and the baseline is shown in figure 14.

	Baseline	Best Model
Initialisation	Kaiming	Kaiming
Pre-processing	Standardisation	Standardisation
Activation	ReLU	LeakyReLU
Batch Norm	No	Yes
Optimizer	SGD	SGD
Weight Decay	0	0
Dropout Rate	0	0.5
Learning Rate	0.001	0.001
Batch Size	128	64
Epoch	80	42
Hidden Layers	2	2
Hidden Units	[256, 512]	[512, 256]
Validation Split	0.2	0.2
Test Accuracy	0.4824	0.5565
Test Loss	3.4596	1.2483

Fig. 14. Baseline vs Best Model

III. Experiments and Results

A. Operation Environment

Hardware

- OS System: Windows 10 64-bit operating system
- CPU: Intel(R) Core(TM) i7-9700KF
- GPU: NVIDIA GeForce RTX 2070
- RAM: 16.0 GB

Software

- Python: 3.8.3
- notebook: 6.4.10

B. Hyper-parameters Analysis

This section analyzes and fine-tunes the effect of using different values for the following hyper-parameters based on the baseline model architecture: learning rate, batch size. dropout rate and the number of hidden layer and units.

1) Learning rate: Learning rate is the most critical hyper-parameter in the optimizer in deep learning, which determines whether the objective function can converge to the local minimum and when it converges to the minimum. A smaller learning rate requires more epoch training, a larger learning rate leads to rapid changes and requires less learning rate. We can see from the validation loss diagram in Figure 16 that the largest learning rate, learning rate = 0.01 (blue line), stops decreasing and starts to rise the fastest. And when the learning rate is the smallest, learning=0.0001 (red line), it will be the slowest to stop decreasing. That's what we expected. We can see from the figure of validation accuracy in Figure 15 and Figure 16, that learning rate = 0.0001 and learning rate = 0.001 have the best performance. Since the process will get stuck when the learning rate is too small, the learning rate = 0.001 would be better than the learning rate = 0.0001.

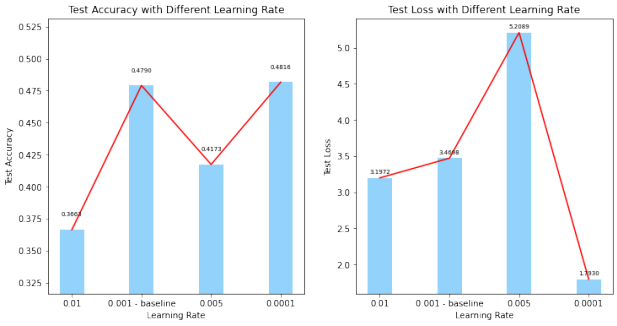

Fig. 15. Learning rate test(accuracy and loss)

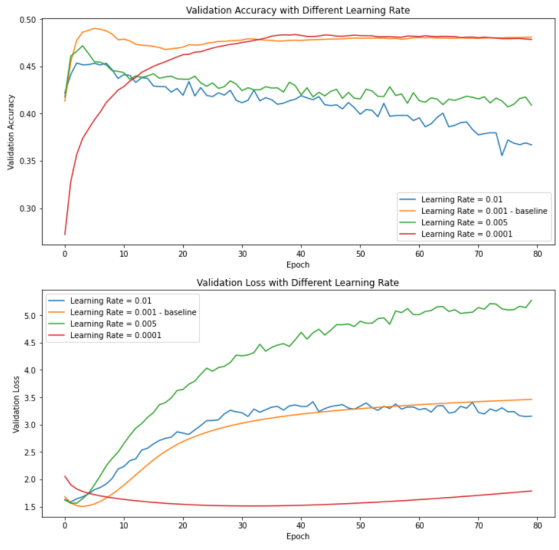

Fig. 16. Learning rate validation(accuracy and loss)

2) Batch size: The batch size will determine the number of samples for a single training of the model, which directly impacts the optimization degree and speed of the model. It is

found from figure 18 that when the batch size is between (32, 64, 128, 256, 512, 1024), the effect on the accuracy and loss of the model validation set is not apparent. When the epoch is less than ten, about 5, each different batch size reaches its highest point. Then the validation accuracy decreases as the epoch increases. Finally, when the batch size of the model is 128 (baseline), the validation accuracy is relatively the highest. When the epoch is less than 30, the validation loss of each model with different batch sizes rises rapidly. When the batch size of the model is 1024, the validation loss is the smallest. It can be seen intuitively from figure 17 that when the batch size of the model is 32, the model obtains the maximum accuracy. When the batch size of the model is 1024, the loss of the model is much larger than other models with different batch size parameters.

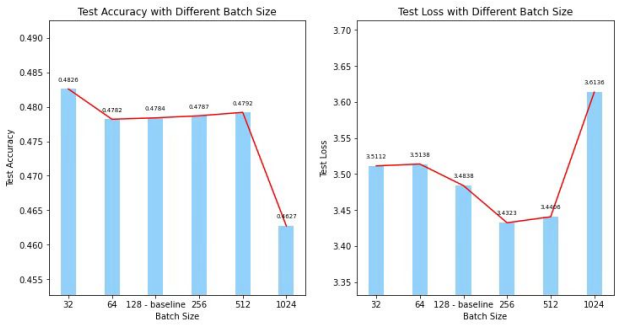

Fig. 17. Batch size test(accuracy and loss)

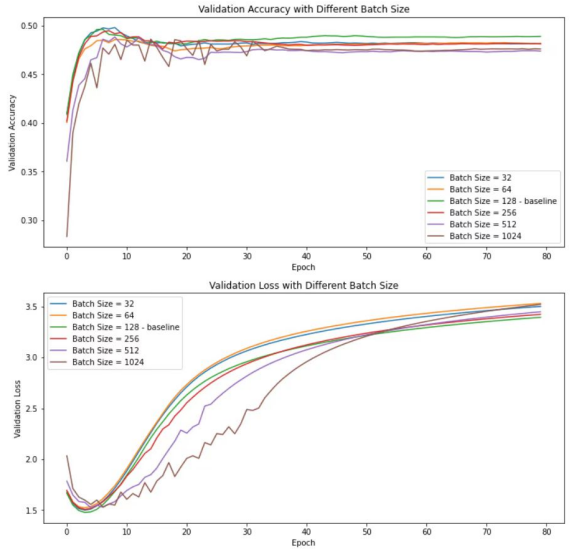

Fig. 18. Batch size validation(accuracy and loss)

3) Dropout rate: The dropout rate is the most critical hyper-parameter in the regularisation method (dropout). It represents the proportion of randomly shut down neurons in the neural network model. The dropout rate determines the ratio of features retained by the neural network, which has a significant impact on the training results of the neural network. It can be intuitively found from figure 20 that During model training, when the dropout rate is 0 (when dropout is turned off), the loss of the model is the highest, and the accuracy of the model is the lowest. This proves that dropout is a method that has a positive effect on the model. When the model dropout rate is 0.3 and 0.5, the model gets the lowest point of loss and obtains the best performance. As shown in figure 19, when the dropout rate is 0.9, the validation accuracy of the model is the lowest, because most of the neurons in the neural network model are disabled, so that the model loses most of the features of the data for learning.

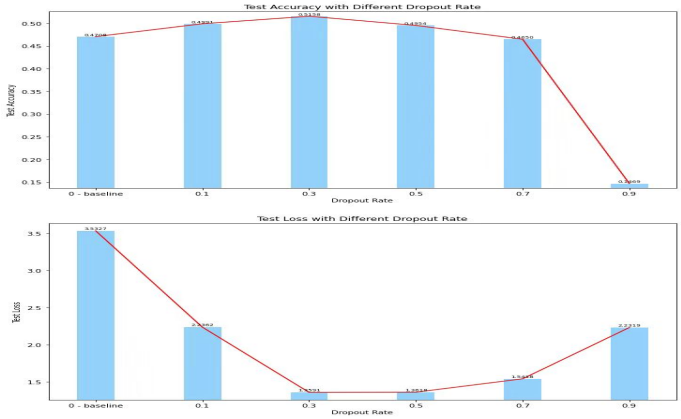

Fig. 19. Dropout rate test(accuracy and loss)

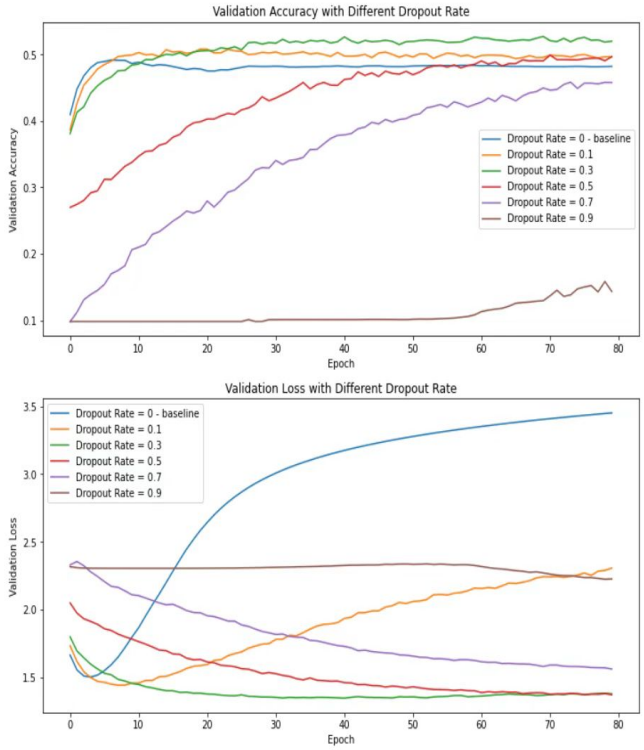

Fig. 20. Dropout rate validation(accuracy and loss)

4) The number of hidden layer and units: In neural networks, the deeper the theoretical upper number, the stronger the ability to fit the function, the better the effect. However, too deep a number of layers may cause over-fitting problems and increase the difficulty of training in the real project. For the number of neurons in the hidden layer, too few neurons will lead to under-fitting, because there are not enough neurons to train all information from the data. While too many neurons will lead to over-fitting because the amount of information in the training set is not enough to train all neurons in the hidden layer. In our experiment, we tested the influence of neural network depth, the number of neurons, and the sequence of neuron numbers on the model respectively. As shown in figure 22, by comparing the accuracy, the worst performance is the three-layer neural network model with 128, 256, and 512 neurons. The model with 512, 256 neurons performed best. By comparing their loss, we can find that the three-layer neural network is generally higher than the two-layer neural network. This is in line with our expectations. Because there are only 128 features in our data, so we don't have enough information to carry out a multi-layer with too many neurons training. According to both 21 and 22, we found a general rule that when the number of neurons in the hidden layer decreases, it usually leads to better performance than the neuron increment model. We suspect that this is because the decreasing number of neurons can assist the gradual convergence of the model, thus improving the model performance.

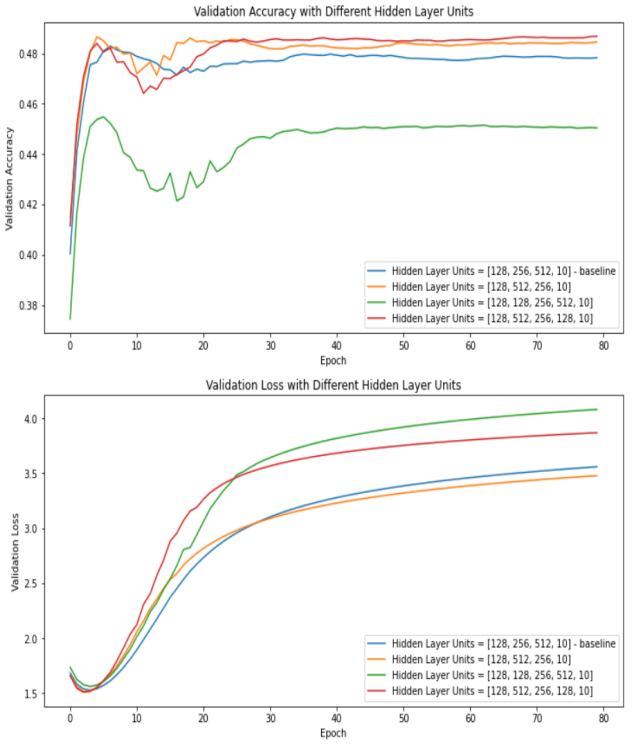

Fig. 21. The number of hidden layer and units validation(accuracy and loss)

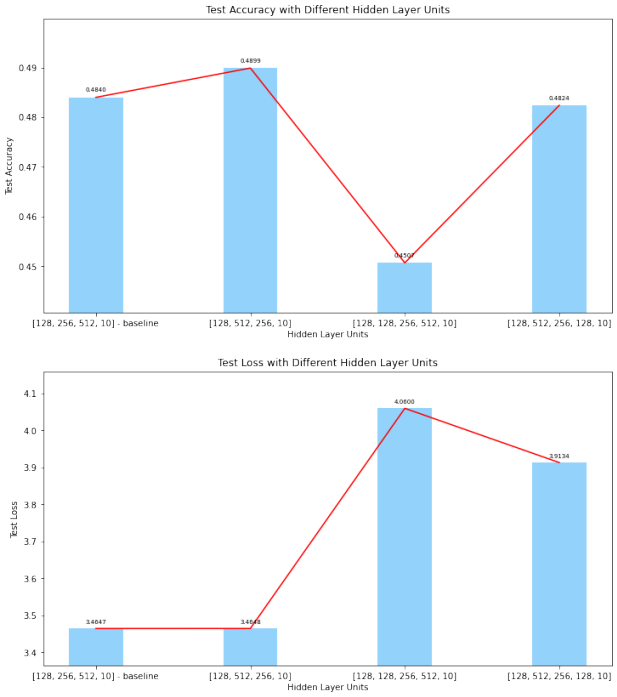

Fig. 22. The number of hidden layer and units test(accuracy and loss)

C. Ablation Studies

1) Introduction: Ablation study refers to the study of the performance of a system after the removal or addition of a component in order to understand the role of that component in the overall system. In this experiment, various advanced methods such as Batch Normalisation, Dropout etc. have been proposed to improve the accuracy of the model. However, in order to verify the effectiveness of these advanced methods, experiments with controlled variables (i.e. different advanced methods) are necessary. It can be seen that the core of the ablation experiment is the control variables. The specific steps to be followed are listed below.

2) Variable settings: In Figure 23, it can be seen that the baseline model uses the ReLU activation function combined with the Kaiming weight initialization method, and the pre-processing is Standardisation with learning rate=0.001 and Batch size=128. In addition, Epoch is set to 80 and the number of hidden layers is set to two layers, [256,512]. The terms listed above, they are constant variables in this ablation experiment, i.e. they do not become the subject of study and never change.In order to see the comparison effect more clearly, we have used bold and highlighted the objects of the experiment with boxes.

3) Comparison: In order to see the comparison effect more clearly, we have used bold and highlighted the objects of the experiment with boxes. They are different in terms of dropout rate compared to the baseline and Model 1. Model 1 adds a Dropout layer and sets the Dropout rate to 0.5, which increases the test set accuracy by about 2% and reduces the loss value by half. A comparison of Model 1 with Model 2 shows that Model 2 adds the Batch Normalisation layer,

in which case the accuracy of the test set still increases, which is not quite in line with the expectation that more components tend to lead to a decrease in accuracy. This therefore shows that Batch Normalisation is very effective. As for using Momentum optimiser and reset learning rate as 0.001 will significantly reduce the accuracy of the test set. Therefore, they do not allow for better generalisation of the model to this data set. Finally, we used Model 2 (the optimal model from the ablation experiments) as a basis and further adjusted the remaining hyper-parameters to arrive at a model with the most desirable performance results, the exact combination of parameters being explained in the next section.

	Baseline	Model 1	Model 2	Model 3	Model 4
Initialisation	Kaiming	Kaiming	Kaiming	Kaiming	Kaiming
Pre-processing	Standardisation	Standardisation	Standardisation	Standardisation	Standardisation
Activation	ReLU	ReLU	ReLU	ReLU	ReLU
Dropout Rate	0	0.5	0.5	0.5	0.5
Batch Normalisation	No	No	Yes	Yes	Yes
Optimizer	SGD	SGD	SGD	M = 0.9	M = 0.9
Weight Decay	0	0	0	0	0.001
Learning Rate	0.001	0.001	0.001	0.001	0.001
Batch Size	128	128	128	128	128
Epoch	80	80	80	80	80
Hidden Layers	2	2	2	2	2
Hidden Units	[256, 512]	[256, 512]	[256, 512]	[256, 512]	[256, 512]
Validation Split	0.2	0.2	0.2	0.2	0.2
Test Accuracy	0.4824	0.5015	0.5217	0.4979	0.4330
Test Loss	3.4596	1.3542	1.3088	1.5215	1.9476

Fig. 23. Ablation Studies

D. Best Model Performance

1) Introduction: In this section, we evaluated the performance of the optimal model by using the precision and loss values of the training and validation sets, as well as the confusion matrix of the testing set. Since our loss function is cross-entropy, the loss curve can be named as cross-entropy loss curve.

2) Loss comparison: As shown in the figure 24, in the first 20 epochs, the train loss of the baseline decreases rapidly to a number close to 0, while the validation loss increases rapidly, which is obviously an over-fitting phenomenon. The Best Model uses the Batch Normalisation layer and the Dropout layer to reduce the risk of over-fitting to a certain extent, so we can see that the Train loss and Validation loss of the Best Model are very close to each other.

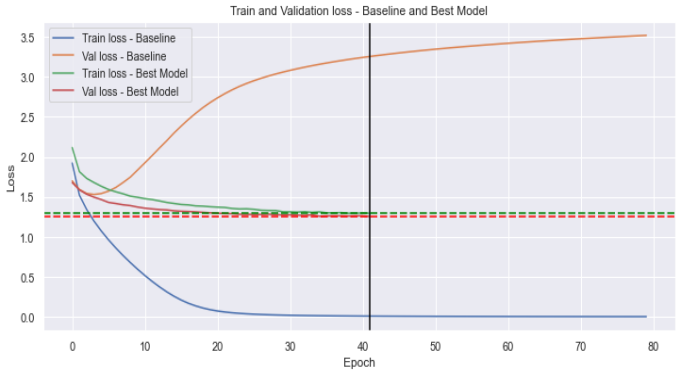

Fig. 24. Train and validation loss- Baseline and Best Model

3) Accuracy comparison: We can also measure the performance of the model by accuracy. As shown in the figure24, since Baseline has not been implemented Batch Normalisation layer and Dropout layer, the Train accuracy is close to 100% after 20 epochs, while the validation accuracy remains at around 40% and does not increase. Clearly, this also indicates that the model Baseline is over-fitting. On the other hand, as for the Best Model, its Train accuracy and Validation accuracy remain at the same level, which proves that the model has a good generalization ability.

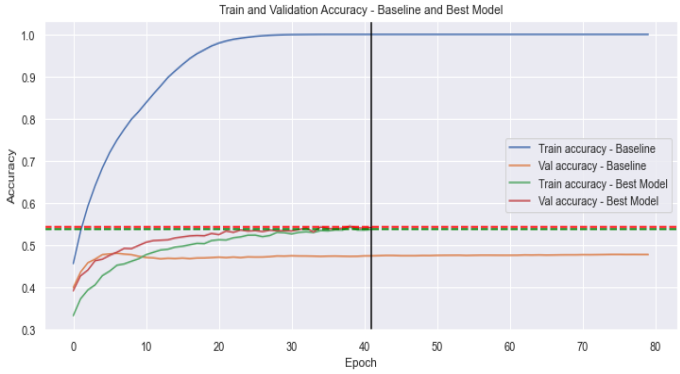

Fig. 25. Train and validation accuracy- Baseline and Best Model

4) Find the optimal model: Our optimal model was obtained by conducting an extensive ablation study and hyper-parameter tuning of one of the high accuracy models. As shown in figure 23, model 2 is the most accurate model for the ablation experiments. Compared to the baseline model, this model has an additional Batch Normalisation trick and the dropout rate is set to 0.5. So an initial thought is that using Batch Normalisation and dropout together for the model in this data set will help to improve the model performance.

Based on the settings in model 2, we tried to change other hyper-parameters to further improve the accuracy of the model in the test set. Based on figure 26, we further optimised the choice of activation function, batch size, epoch and hidden units of model 2 to obtain the optimal model for this experiment.

5) The overall process: The training process of the best model is roughly as follows: firstly, 64 data are selected as batch in the training set, the input data are pre-processed by standardize, and then the weights are initialized by the Kaiming method since the activation function is LeakyReLU. As for the hidden layers, we set up more neurons (512) in the first hidden layer and reduced the number of neurons in the second hidden layer appropriately. In the forward propagation, the input data is activated by the non-linear function LeakyReLU in the two hidden layers, but also by the Batch Normalisation layer and the Dropout layer, and the final output value is generated in the calculation of the loss function. As for the back propagation, the SGD method of updating parameters is still used, where the learning rate is set to 0.001. Finally, we find that the model achieves a highest

accuracy (55.65%) in the test set at epoch 42, which means that the training time is reduced and the prediction accuracy of the model is improved.

	Model 2	Best Model
Initialisation	Kaiming	Kaiming
Pre-processing	Standardisation	Standardisation
Activation	ReLU	LeakyReLU
Batch Norm	Yes	Yes
Optimizer	SGD	SGD
Weight Decay	0	0
Dropout Rate	0.5	0.5
Learning Rate	0.001	0.001
Batch Size	128	64
Epoch	80	42
Hidden Layers	2	2
Hidden Units	[256, 512]	[512, 256]
Validation Split	0.2	0.2
Test Accuracy	0.5217	0.5565
Test Loss	1.3088	1.2483

Fig. 26. Model 2 vs Best Model

The above analysis has used accuracy and loss value as separate measures of the model, but these are examined from the macroscopic perspective of the model. In order to further analyse the generalisation ability of the model, the predictions from the test set have been printed out in the form of a confusion matrix.

6) Confusion matrix: As shown in figure27, the vertical coordinates are the true labels and the horizontal coordinates are the predicted labels. The most serious classification error is in class3, where 390 data are misclassified as class5. The next most serious classification error is in class2, where 166 data are misclassified as class4, 169 data are misclassified as class5 and 93 data are misclassified as class6. In addition to this, 160 data were misclassified as class 5 in class 7, which also has a high error rate. It is interesting to note that in the prediction of class1, 165 of the 10,000 test data were misclassified as class9, while 153 of the class9 data were misclassified as class1. This is likely due to the fact that class1 and class9 have very similar characteristics, resulting in more classification by the model.

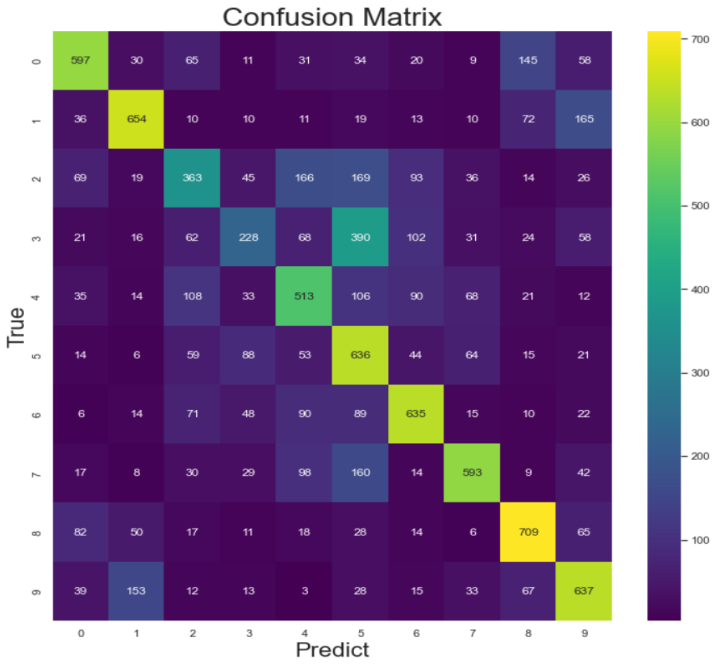

Fig. 27. Confusion Matrix

IV. Discussion and Conclusion

A. Discussion

In this project, we have investigated the influence of the different components of the neural network and developed an optimal model through comparative experiments with hyperparameters as well as ablation experiments. Specifically, the optimal model has two hidden layers, with 512 and 256 neurons in each layer, and the final model has an accuracy of more than 55.5%. It is worth noting that the model uses the LeakyReLU activation function, which avoids the death of neurons in ReLU and improves the model's fitting ability to a certain extent. In addition, we use Batch Normalisation and Dropout to suppress over-fitting, so our model has no over-fitting problems. However, we did not set weight decay on the optimal model because they are both over-fitting suppression methods. If they were used together, the final result would be reduced instead. Our guess is that this would unduly restrict the learning ability of the model and reduce the robustness of the model. Perhaps we can vary the effect of Batch Normalisation in the size of the Batch size, thus making the combined approach more effective. However, due to time constraints, we have not investigated this aspect further. Also, many scholars have pointed out that Batch Normalisation methods alone tend to be more effective than a combination of other methods. It has even been suggested that if the Batch Normalisation layer is added after the activation function, better results tend to be obtained. Therefore, this experiment directly adopts this improvement idea, but does not delve into its reasons and rationality.

B. Conclusion

In summary, we have constructed a multi-layer neural network from scratch in this experiment and then conducted comparative experiments on the hyper-parameters in it, as well as ablation experiments on different advanced methods. In the end, we arrived at the optimal model, which was more than 55% accurate in the test set and was able to complete the classification task in a short time. The best model runs 42 epochs to achieve the highest accuracy, which takes a total of 326 seconds.

The most difficult part of the experiment would be the construction of Batch Normalisation. This is because in addition to the normalisation of the Batch size dataset involved in the forward propagation, the scaling and translation parameters need to be updated in the backward propagation. And we also need to implement different optimisers, so a correct parameter update formula becomes especially crucial in the back-propagation, which directly affects the four parameters.

A shortcoming regarding this experiment should be the lack of a deeper study of the application of Batch Normalisation. Future improvements include:

- Comparing the effects of Batch size training on Batch Normalisation.
- Analysing the impact of Batch Normalisation combined with other methods.
- Analysing the implications of Batch Normalisation layers in various positions in the and adjusting to improve the accuracy of the deep learning model.

In general, this project explains the functions and performance of various components of deep learning from theory to practice, and provides a good basic framework for deep learning beginners.

REFERENCES

[1]]I. Goodfellow, Y. Bengio and A. Courville, Deep learning. Cambridge, Mass: The MIT Press, 2017.

[2] K. Cios, W. Pedrycz and R. Swiniarski, Data mining methods for knowledge discovery. Boston: Kluwer Academic, 1998.

[3] "International Conference on Artificial Neural Networks (ICANN 06)", IEEE Transactions on Neural Networks, vol. 17, no. 2, pp. 537-537, 2006.

[4] Y. Liu, S. Liu, Y. Wang, F. Lombardi and J. Han, "A Stochastic Computational Multi-Layer Perceptron with Backward Propagation", IEEE Transactions on Computers, vol. 67, no. 9, pp. 1273-1286, 2018.

[5] G. Panchal, A. Ganatra, Y. Kosta and D. Panchal, "Behaviour Analysis of Multilayer Perceptronswith Multiple Hidden Neurons and Hidden Layers", International Journal of Computer Theory and Engineering, pp. 332-337, 2011.

[6] J.Feng and S. Lu, "Performance Analysis of Various Activation Functions in Artificial Neural Network", Journal of Physics: Conference Series, vol. 1237, no. 2, p. 022030, 2019. Available: 10.1088/1742-6596/1237/2/022030.

[7] C. Banerjee, T. Mukherjee and E. Pasiliao, "An Empirical Study on Generalizations of the ReLU Activation Function", Proceedings of the 2019 ACM Southeast Conference, 2019. Available: 10.1145/3299815.3314450 [Accessed 7 April 2022].

[8] B. Yuen, M. Hoang, X. Dong and T. Lu, "Universal activation function for machine learning", Scientific Reports, vol. 11, no. 1, 2021. Available: 10.1038/s41598-021-96723-8.

[9] M. Narkhede, P. Bartakke and M. Sutaone, "A review on weight initialization strategies for neural networks", Artificial Intelligence Review, vol. 55, no. 1, pp. 291-322, 2021. Available: 10.1007/s10462-021-10033-z.

[10] N. Qian, "On the momentum term in gradient descent learning algorithms", Neural Networks, vol. 12, no. 1, pp. 145-151, 1999. Available: 10.1016/s0893-6080(98)00116-6.

[11] S. N. Gedela, "Measuring accuracy of dataset using Deep Learning Algorithm RMSPROP algorithm," International Journal for Research in Applied Science and Engineering Technology, vol. 9, no. 10, pp. 99–112, 2021.

[12] S. Mc Loone and G. Irwin, "Improving neural network training solutions using regularisation," Neurocomputing, vol. 37, no. 1-4, pp. 71–90, 2001.

[13] H. Wu and X. Gu, "Towards dropout training for convolutional neural networks", Neural Networks, vol. 71, pp. 1-10, 2015. Available: 10.1016/j.neunet.2015.07.007.

[14] D. Bacciu and F. Crecchi, "Augmenting Recurrent Neural Networks Resilience by Dropout", IEEE Transactions on Neural Networks and Learning Systems, vol. 31, no. 1, pp. 345-351, 2020. Available: 10.1109/tnnls.2019.2899744.

[15] K. Nakamura and B. Hong, "Adaptive Weight Decay for Deep Neural Networks", IEEE Access, vol. 7, pp. 118857-118865, 2019. Available: 10.1109/access.2019.2937139.

[16] Q. Zhu, Z. He, T. Zhang and W. Cui, "Improving Classification Performance of Softmax Loss Function Based on Scalable Batch-Normalization", Applied Sciences, vol. 10, no. 8, p. 2950, 2020. Available: 10.3390/app10082950.

[17] D. Zhu, S. Lu, M. Wang, J. Lin and Z. Wang, "Efficient Precision-Adjustable Architecture for Softmax Function in Deep Learning", IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 67, no. 12, pp. 3382-3386, 2020. Available: 10.1109/tcsii.2020.3002564.

[18] A. Jamin and A. Humeau-Heurtier, "(Multiscale) Cross-Entropy Methods: A Review", Entropy, vol. 22, no. 1, p. 45, 2019. Available: 10.3390/e22010045.

[19] H. Park and K. Lee, "Adaptive Natural Gradient Method for Learning of Stochastic Neural Networks in Mini-Batch Mode", Applied Sciences, vol. 9, no. 21, p. 4568, 2019. Available: 10.3390/app9214568.

[20] S. Okada, M. Ohzeki and S. Taguchi, "Efficient partition of integer optimization problems with one-hot encoding", Scientific Reports, vol. 9, no. 1, 2019. Available: 10.1038/s41598-019-49539-6.

APPENDIX

Code File Instruction

We provide two ways to run our code:

1) Run code boxes one by one
   - **Loading data:** Go to the section 2. Loading the Dataset in code file, you can a) modify the data file path b) create a new folder named *Assignment1-Dataset* in the folder where this ipynb file resides. Make sure the following files are in the folder at the specified path: *train _data.npy*, *train _lable.npy*, *test_data.npy*, *test_label.npy*
   - **Run code:** go to the section 1. Setting the Environment, run the code boxes one by one until the end of the section 5. Train and Test for Best Model code box
   - **This way you can see our detailed comments on each step and understand our code more clearly**
2) Simple one-click operation
   - **Loading data:** Go to the section 7. Easy to Use, set the data file path at the top of the code box and make sure the following files are in the folder at the specified path: *train _data.npy*, *train _lable.npy*, *test_data.npy*, *test_label.npy*.
     - Example: If my data file path is *"/Users/XXX/Desktop/Ass1/train_data.npy"*, then type *"/Users/XXX/Desktop/Ass1/"* in *"myPath"*
   - **Run code:** Simply run this single code block to get the results and visualization of our best model.
   - **This way you will be able to easily check whether our code works and whether the results are as expected**