Implement a Multi-Layer Neural Network from Scratch

Tianyi Chen 490165227, Jiahao Wu 490517596

Abstract – **The aim of this study is to implement a multi-layer neural network from scratch to classify a 10 classes data set containing 128 attributes with a total of 50000 data. In this study, we explored the effects of different methods and deep learning techniques on multi-class classification results, in terms of accuracy and efficiency. These methods and techniques include: Data-preprocessing, Linear classification, ReLU activation, Softmax, Dropout, Batch normalization, weight decay, Cross Entropy loss, Mini Batch training, Momentum in SGD and Adam Optimizer.**

Keywords **– Implementation from scratch, Adam, Multi-layer Neural network, multi-class classification**

I. INTRODUCTION

A. Aim of the study

We live in an era of unprecedented opportunities, and deep learning technologies can help us achieve new breakthroughs. Deep learning has played an important role in exploring exoplanets, developing new drugs, diagnosing diseases and detecting subatomic particles, etc. It can radically enhance our understanding of biology, including genomics, proteomics, metabolomics, immunoomics, and more.

In the past decade, the number of papers of deep learning on ArXiv has increased nearly tenfold, reflecting the importance and exploration of the field. As the basis of deep learning, neural network is the most important to be read through its structure and the mathematical principle behind it. However, how to optimize the network is not a mature theoretical guidance like physics or mathematics, and it is still an unknown thing that needs to be explored.

On this premise, we start from the very beginning, implementing a deep neural network from scratch, aiming to understand the mathematics behind it and as well as to evaluate the performance and effects of adapting different regularisation and optimisation techniques, to make a better development in the future.

B. Significance

A neural network, in the field of machine learning and cognitive science, is a mathematical model or computational model that mimics the structure and functionality of a biological neural network (the central nervous system of an animal, especially the brain) and is used to estimate or approximate functions. A neural network consists of a large number of artificial neurons connected to perform calculations. By using statistical methods, we enable artificial neural networks to make simple decisions and make simple judgments like humans, giving them an advantage over formal logical reasoning.

Like other machine learning methods, neural networks have been used to solve a variety of problems, such as computer Vision and Natural Language Processing. These problems are difficult to solve by traditional rule-based programming. In the foreseeable future, AI will play an important role in human beings, and many positions will be replaced by AI. As one of the most important components of AI, it is necessary to have a comprehensive understanding of deep neural networks and how it is optimised to train.

Currently, there are some python libraries of Deep Learning frameworks that are extremely helpful for constructing a deep learning network: PyTorch, TensorFlow, Caffe, and KERAS. However, it is also essential to understand how these libraries actually works. Hence, building and training a deep learning network from scratch without using these packages and auto-grad tools is the key for the deep learning learners.

Therefore, the importance of this study lies on the fact that the structure (e.g., artificial neurons, linear layers and activation layers), the optimisation (e.g., criterion/loss function, forward propagation, backward propagation, different optimisers) and regularisation(e.g., dropout, batch normalisation) of a neural network are the fundamentals of the deep learning area. By fully comprehending these fundamentals, one can be possible to explore and research more in this field.

II. Methods

A. Pre-processing

1) standardization:

$$Z = \frac{X - E[X]}{\sigma(X)} \tag{1}$$

Data standardization is the process of re-scaling the attributes so that they have a mean of 0 and a variance of 1. This helps to bring down all the features to a common scale without distorting the differences in the range of the values. Standardization ensures that the distribution of each dimension of input is similar, and the neural network will not be inclined to the dimension with larger variance without knowing the attribute stands for. Data standardization also allows the neural network to converge better and faster.

B. The principle of different modules

1) multi-hidden layer:

Multi hidden layers give the neural network the ability to learn abstract features and eventually estimate a function represented by the data. In neural networks, the input of neurons of the deeper layer is the weighted sum of the output of the shallow layer, thus the features of the shallow layer are abstracted to the deeper layer. In fact, the

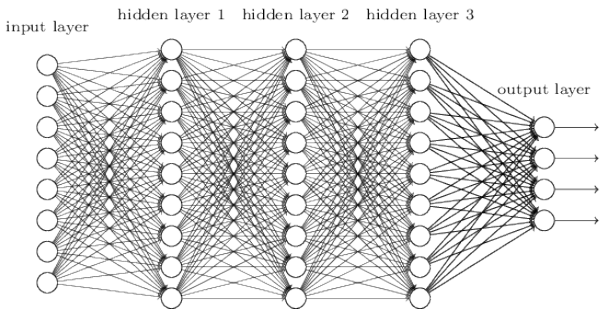


Fig. 1. multi-hidden layer neural network

learning process is the process of adjusting and optimising the weights and bias of each connection, and continuously learn from the abstract features. The deeper the network is, the feature abstraction ability is at least not lower. That is, "better" for certain tasks, which requires a high level of abstract understanding like human do.

2) ReLU activation:

$$x = \begin{cases} 0, & \text{if} \quad x <= 0 \\ x, & \text{if} \quad x > 0 \end{cases} \tag{2}$$

The rectified linear activation function or ReLU is a function that will output the input directly if it is positive, otherwise, it will output zero. Its nonlinear property helps the network to learn complex patterns in the data.

Compared with traditional neural network activation functions, such as sigmoid and tanh, linear rectification functions are more efficient on gradient descent and backpropagation: it avoids gradient explosion and gradient disappearance problems. It also simplifies the calculation process: there is no influence of other complex activation functions such as exponential functions; at the same time, the dispersion of the activity reduces the overall computational cost of the neural network.

3) Weight decay:

Weight decay is a regularisation technique used to avoid over-fitting by limiting the growth of the weights in the network. On the researcher's view, for example, l2 regularisation is added to the original loss function, penalising large weights, hence by considering the update of the gradient descent, the weight decay parameter α can be used as follows:

$$\theta \leftarrow \theta - \eta \nabla \hat{L}_R(\theta) \tag{3}$$

$$\theta \leftarrow \theta - \eta \nabla \hat{L}(\theta) - \eta \alpha \theta \tag{4}$$

That is, after normal update, reduce the parameter by the product of learning rate, weight decay α and the parameter itself.

4) Softmax:

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{5}$$

Softmax function is used to normalize the outputs, converting them from weighted sum values into probabilities that sum to one, which could be seen as the multi-class categorical probability distribution. Each value in the output of the softmax function is interpreted as the probability of its membership for each class.

5) Cross Entropy Loss:

$$\text{LOSS} = -\sum_{c=1}^{M} y_{o,c} \log(p_{o,c}) \tag{6}$$

Cross Entropy Loss measures the performance of a classification model whose output is a probability value between 0 and 1. It computes the difference in the distribution of the ground truth and the predicted results. By minimising cross entropy loss as the objective function, the probability of the correct answer to be chosen is maximised and the probability of the wrong result to be chosen is minimised.

6) Dropout:

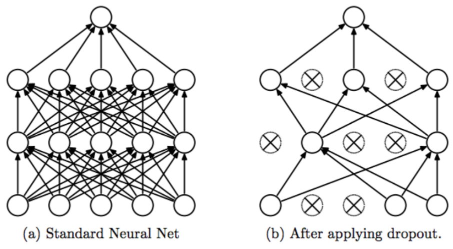
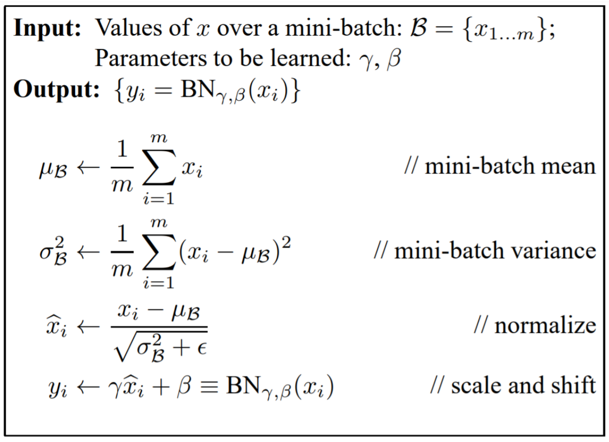


Fig. 2. Dropout in neural network

The Dropout layer randomly sets input units to 0 with a frequency of rate at each step during training time, which helps prevent overfitting. The inverted dropout is adapted in this study. Essentially, during the training time, the weights are scaled by 1 over the dropout rate so that the expectation of the output stays the same during the inference time.

7) Batch Normalization:



Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \text{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \text{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Fig. 3. Batch normalization algorithm

Batch normalization is a method used to make artificial neural networks faster and more stable through normalization of the layers' inputs by re-centering and re-scaling. It helps solving the Internal covariate shift problem.

8) Mini-batch training:
Mini-batch gradient descent is a variation of the gradient descent algorithm that splits the training dataset into small batches that are used to calculate model error and update model coefficients.

9) Momentum in SGD:

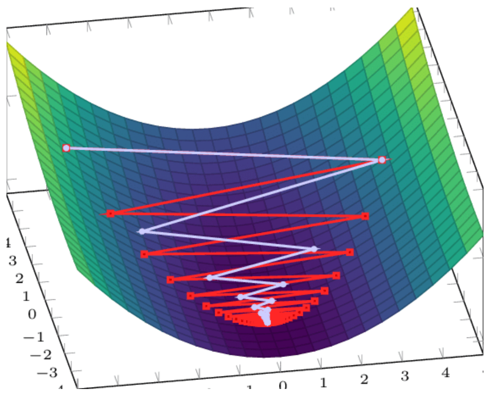


Fig. 4. SGD convergence trajectory, red: without momentum, white: with momentum

SGD with momentum is a method which helps accelerate gradients vectors in the right directions, thus leading to faster converging. The momentum term increases for directions where its gradients align with the same direction and reduces updates for dimensions where its gradients chenge directions.

The algorithm of SGD with momentum is given below:

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta) \tag{7}$$

$$\theta = \theta - v_t \tag{8}$$

10) Adam:
Among the variants of SGD, ADAM is one of the most optimiser used in training the deep neural networks, which is derived from adaptive moment estimation. It combines the best properties of the ADAGRAD and RMSPROP algorithms to provide an optimization algorithm that can handle sparse gradients on noisy problems. The update for ADAM method involves the usage of momentum and adaptive selection, picking a stochastic gradient g_t, same as SGD. Then the update is as follows:

$$\begin{aligned} m_t &:= \beta_1 m_{t-1} + (1-\beta_1) g_t && (9) \\ [v_t]_i &:= \beta_2 [v_{t-1}]_i + (1-\beta_2)([g_t]_i)^2, \ \forall i && (10) \\ [x_{t+1}]_i &:= [x_t]_i - \frac{\gamma}{\sqrt{[v_t]_i}} [m_t]_i, \ \forall i && (11) \end{aligned}$$

where m_0 and v_0 are set to be 0. The equation (10) is the *momentum term* and the equation (11) is the *2nd-order statistics* used for adaptive selection. Hence, ADAM is able to forget older weights faster and use momentum from previous gradients to prevent oscillating.

C. Best model

The structure, hyperparameters of our best model is summarised in table I. Our best model reaches an accuracy of **58.62%** and a loss of **1.22**.

Learning rate	5e-4
Number of layers	3
Hidden size	256
batch size	64
optimisation	Adam(lr=1e-4, betas=(0.9,0.999), eps=1e-8)
dropout rate	0.3
activation	ReLU

TABLE I
Structure and hyperparameters of the best model

As shown in the table II, the precision, recall and F1 score on different classes is summarised as well as the weighted average of these scores.The weighted average of F1 score is 0.58, which marks the final accuracy of our best model.

Fig 5 and Fig 6 are the accuracy plot and the loss plot of our best model during training and testing.

Finally, Fig 7 displays the confusion matrix of the prediction results of our best model.

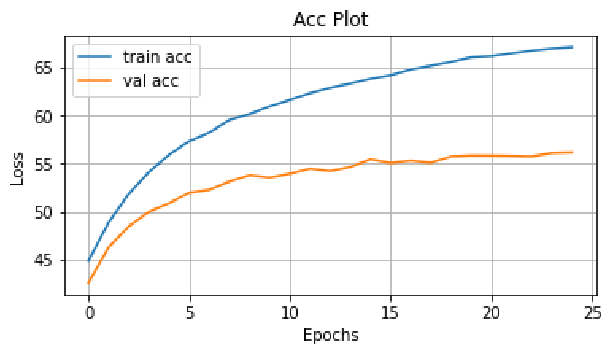


Fig. 5. Accuracy of the best model

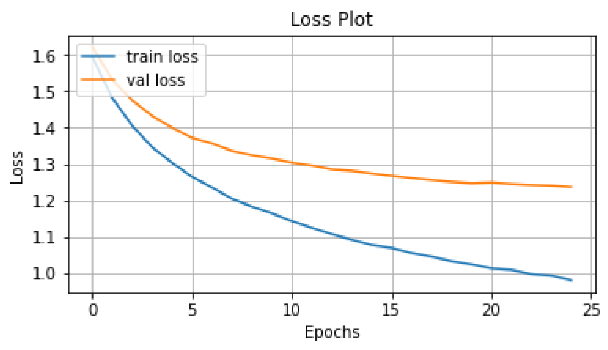


Fig. 6. loss of best model

III. Experiments and results

A. Tuning

Table III summaries the different hyperparameters used to tune the model and for comparable evaluations.

B. Performance

1) Training and testing accuracy:
For the classification task of this 10-classes data set, the best model constructed has an accuracy around 58% on the

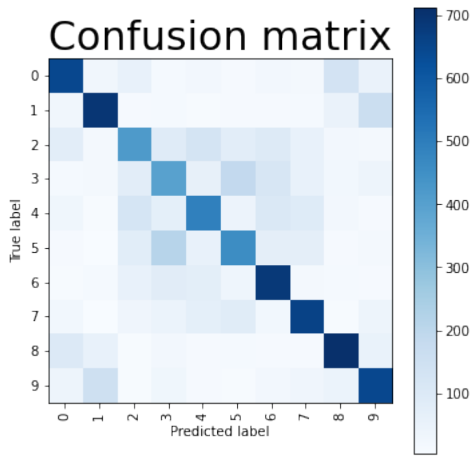


Fig. 7. confusion matrix of the best model

class	precision	recall	f1-score	support
0	0.66	0.65	0.65	1000
1	0.69	0.69	0.69	1000
2	0.48	0.42	0.45	1000
3	0.41	0.40	0.40	1000
4	0.53	0.49	0.51	1000
5	0.49	0.46	0.47	1000
6	0.59	0.69	0.63	1000
7	0.65	0.66	0.65	1000
8	0.68	0.71	0.70	1000
9	0.62	0.65	0.63	1000
avg	0.58	0.58	0.58	10000

TABLE II
F1 scores of the best model

test dataset. We reckon this model reaches the limit of multi-layer neural networks, if only considering the deep learning methods and techniques used in this study.

As shown in figure 5, as the epoch increases, the accuracy on the training set rises steadily, while accuracy in the test set almost reaches the upper limit, maintaining at about 56%. However, the testing accuracy is increasing throughout the entire training, indicating that our best model avoids the overfitting issue.

During the hyper-parameter tuning, utilising each deep learning method and technique to the network contributes positive impact on the accuracy, but increasing the network size (depth and width) does not significantly improve accuracy on the testing dataset after the accuracy exceeds 56%. One guess is that 56% may be close to the best of what we can achieve with the network structure and modules used in this study.

2) Training and testing loss:

As shown in Figure 6, the average loss of the model as per epoch on training and testing dataset shows a decreasing trend.

Number of layers	1, 2, 3, 5
Hidden size	64, 128, 256
Learning rate	1e-4, 5e-4. 1e-3
batch size	16, 32, 64
dropout rate	0.1, 0.3, 0.5
optimisation	Adam, SGD with momentum and weight decay

TABLE III
Different hyperparameters used to tune the model

3) Confusion matrix:

Figure 7 shows the confusion matrix of the prediction and ground truth of best model. It is clear to see that for some classes such as class 1-2 and class 6-9, the correct prediction are larger than the rest of classes, indicating that different class has various difficulties to be learnt by the model.

4) Precision, recall and F1 score:

Table II summarises the classification report of the proposed model. The scores among each class varies greatly. The precision of the category label= 2,4,5 is about 50%, and the precision of the category label=3 is only 41%, the precision of label=1,8 is approximately 70%, and the rest are around 60%. An important observation lies on the scores of class4 and class6 since their precision differs significantly from their recall. For class 4, the precision is quite larger than its recall, indicating that for all samples that are predicted as class 4, the samples that are actually class 4 is more than for the samples that are actually class 4 being predicted as class 4. And for class 6, it is vice versa: the ratio of class 6 being correctly predicted is larger than the ratio of class 6 among all samples predicted as class 6. Moreover,the f1-score 0.5839 is similar to the total accuracy 0.5862, implying a relatively balanced prediction.

5) efficiency:

Training a neural network with 3 hidden layers(each with 256 width), with dropout and Batchnorm, takes about 9 seconds for each epoch, slightly slower than the implementation from PyTorch (which taks 2-5 seconds). The prediction time on the test set was around 2 seconds, also slightly slower than PyTorch's. The total time it takes to train the best model we implemented is between 5-6 minutes (25 epochs).

Overall, the training and prediction time was well within the acceptable range and not significantly slower than PyTorch (without CUDA). However, as the network size increases, there will be a noticeable difference in training speed compared to PyTorch when training a very large scale network (over 5 layers and 256 layer width).

C. Extensive analysis

1) Number of Hidden Layers:

The results in figure 8 display the accuracy and loss on the testing dataset for neural networks with different number of hidden layers. The trend shows that the neural network with 3 layers outperform other models. Specifically, too less layers (only 1 layer) and too much layers (5 layers) are not the best choice.

2) Size of Hidden Layers:

Figure 9 shows the performance of neural networks with different hidden sizes. It is clearly that a larger size of hidden layer gives a better result. Hence, hidden size of 256 neurons is the best choice in this analysis.

3) Learning rate:

The Learning rate analysis is conducted by using Adam. Figure 10 depicts that the learning rate of 5e-4 produces the best result as it has a moderate value. Specifically, a small learning rate (1e-4) converges slow and does not reach the optimal value while a large learning rate (1e-3) converges fast but delivers a ordinary result since the oscillation may occur.

4) Batch Size:

Figure 11 indicates that a batch size of 64 results in the best performance.

5) Dropout rate:

The results depicted in Figure 12 are the performance of the model under different dropout rate. When the dropout rate is too small (0.1) , overfitting may happen and when the dropout is too large (0.5), the network abandons too many neurons that the entire network cannot learn enough abstract features and information and causing underfitting issue.

6) Optimisers:

Figure 12 displays the results on using different optimisers: Adam and SGD with momentum and weight decay. It is obvious that Adam is superior than the SGD variant since it has a higher accuracy and a lower loss.

7) Ablation studies:

The ablation studies are adapted based on the best model with some modifications on three sub-tasks: best model minus dropout layer, best model minus batch normalisation layer and best model minus both dropout and BN layers.

Without dropout layers, as shown in Figure 14, the accuracy on testing dataset decreases after 4 epochs, while the testing loss suddenly increases significantly after the same point, indicating that overfitting has happened. Since the dropout layer is mainly used for avoiding overfitting, this observation justifies the importance of dropout layer in training deep neural networks.

Without BN layers, the training converges slower than other cases, however, no obvious overfitting occurs.

Without both kinds of layers, the training has a relative slow convergence comparing to the best model and the overfitting still happens as no dropout layer adapted as shown in the loss plot in Figure 14.

Therefore, constructing a deep neural network with dropout layers and batch normalisation layers enable is important to avoid overfitting and speed up training and convergence process.

D. Justification of the best model

The best model is constructed by 3 hidden layers and each hidden layer consists of 256 neurons. ReLU is used as activation function for each hidden layer to adapt non-linearity and Softmax is set as the activation of the output layer. Dropout rate is 0.3 and the model also utilises the batch normalisation at each hidden layer. Learning rate is set as 5e-4 with Adam as the optimiser. Multi-class cross entropy loss is chosen as the criterion and a mini-batch of size 64 is used.

This model reaches the highest accuracy because: 1. The appropriate number of hidden layers and hidden layer size enables the model to better extract features from given training data and has enough parameters to estimate the function represented by the data. 2. The ReLu activation adds non-linearity and sparsity to the learning process for the model to better estimate the function. 3. Softmax and cross entropy loss are suitable for multi-class classification since one converts the output to probability distribution and the other measures the similarity between two distributions. Mini-batch training with proper size accelerates the training process and Adam helps to better optimise the objective function with adaptive learning rates and momentum. Finally, from the ablation studies, dropout layer and batch normalisation layers are essential for the model to avoid overfitting and to speed up convergence.

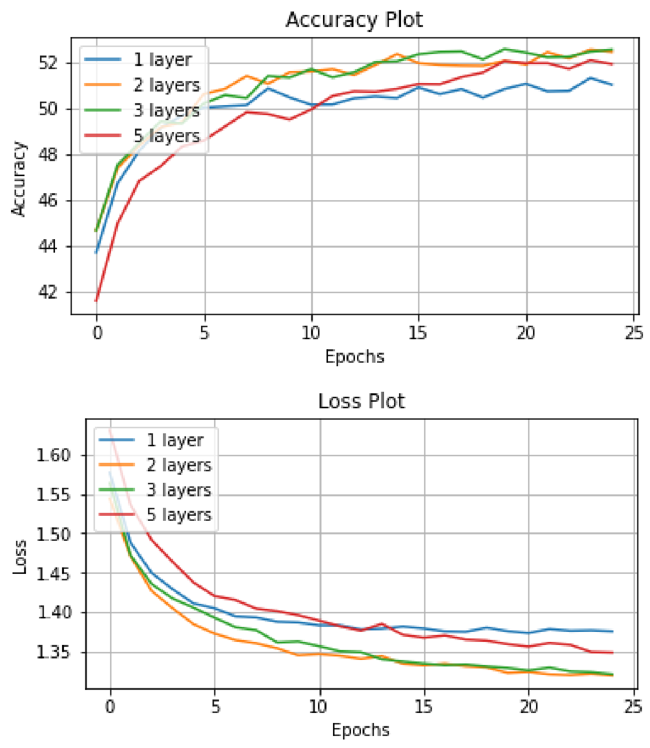
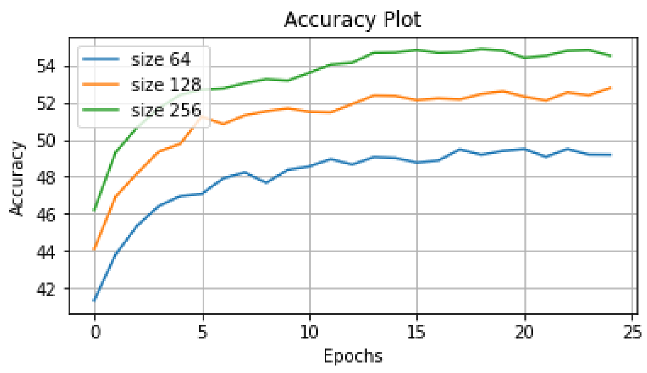


Fig. 8. Number of Hidden Layers Optimisation



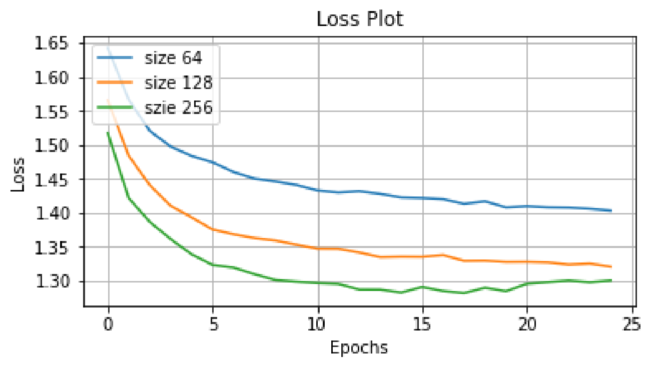
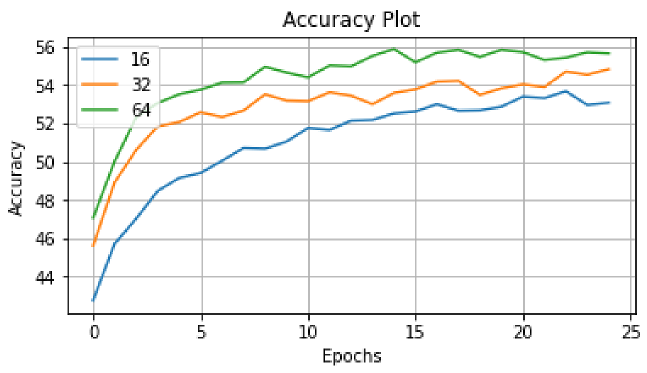


Fig. 9. Size of Hidden Layers Optimisation



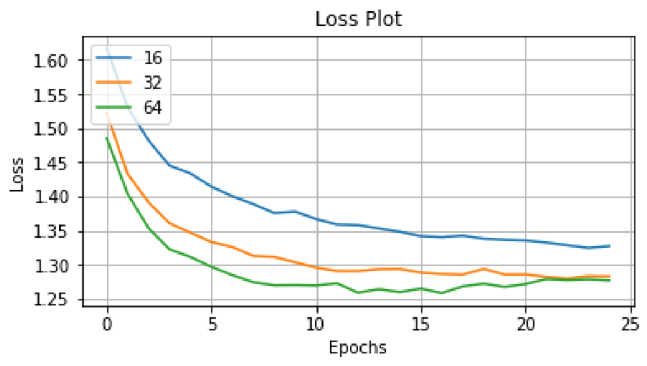
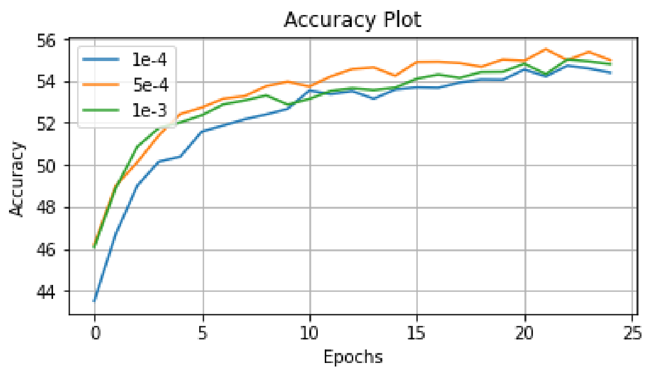


Fig. 11. Batch Size Optimisation



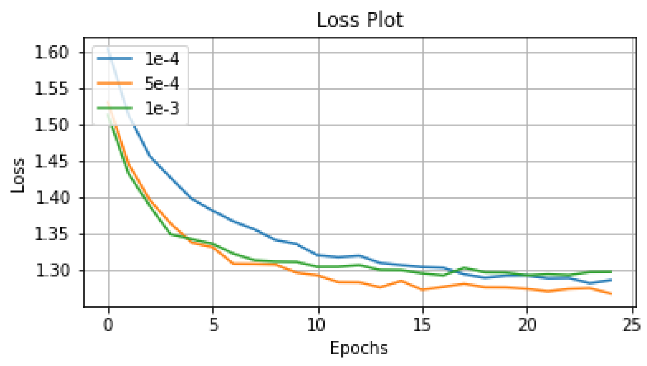
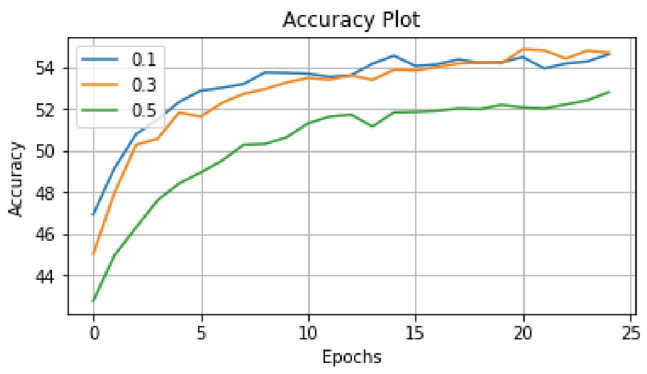


Fig. 10. Learning Rate Optimisation



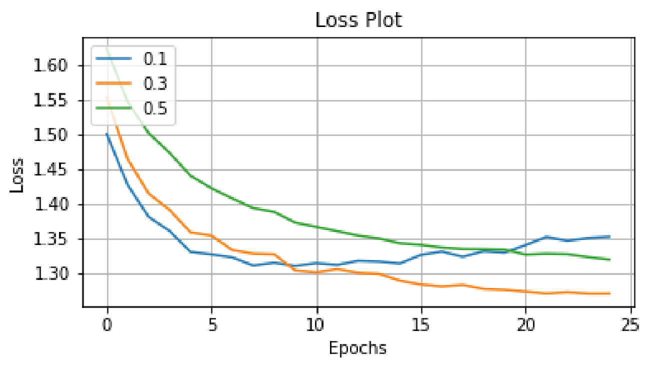


Fig. 12. Dropout Rate Optimisation

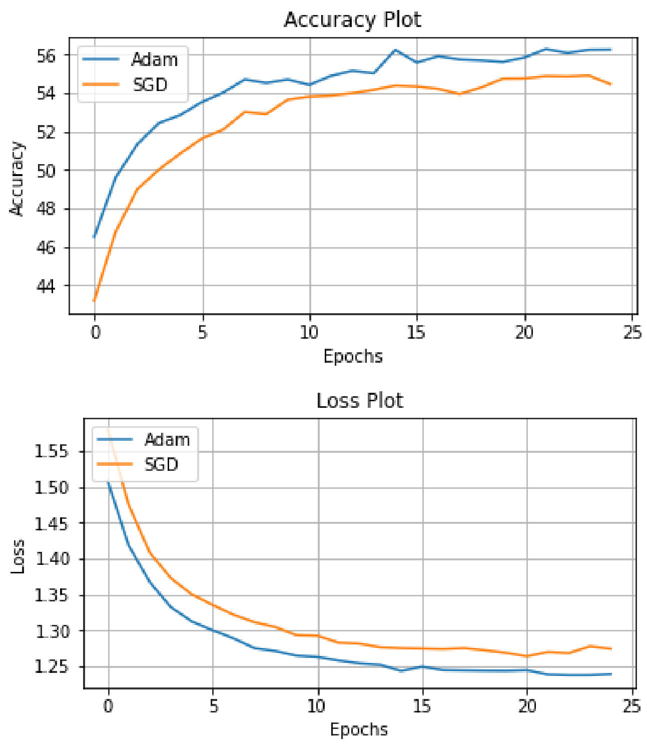


Fig. 13. Optimiser Optimisation

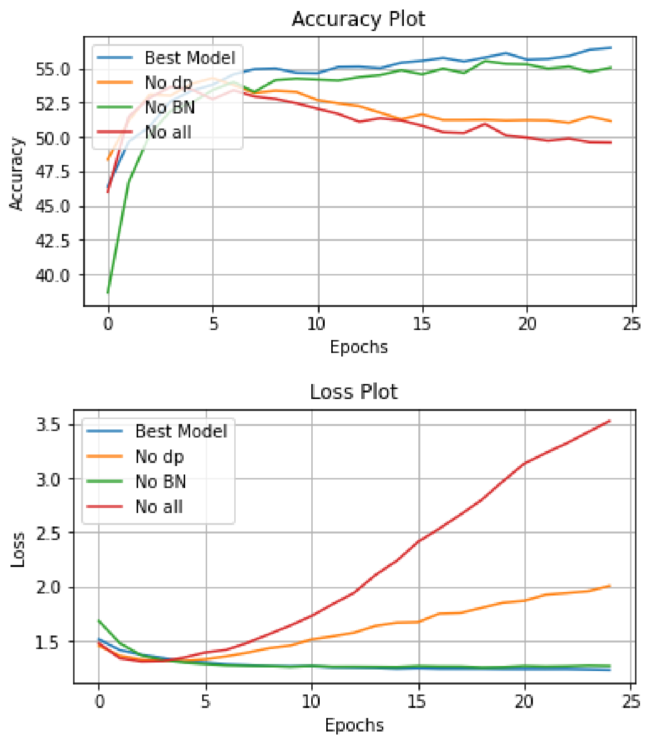


Fig. 14. Ablation Studies

IV. Discussion and conclusion

The purpose of this study is to implement a multi-layer neural network from scratch to classify a 10 classes data set with 60000 entries. In this study, we implemented:

- preprocessing
- ReLU activation
- Linear layer
- weight decay
- Momentum in SGD
- Dropout
- Softmax and cross entropy loss
- Mini batch training
- Batch normalization
- Adam optimizer

In conclusion, we build a neural network to test the influence of all the above deep learning methods and techniques on the datasets provided, and eventually reach an accuracy of around 58% on the testing dataset. During this study, we encountered several difficulties, including the understanding and realisation of mathematical logic behind, the connection between modules, and their effects on the overall accuracy. By tackling with these problems, we genuinely deepen our understanding of neural networks. We figured that not all modules improves the accuracy, and the neural network is not just a simple stack of fancy methods. The parameters and network structure should be fine-tuned according to the practical dataset to achieve the best effect.

For future studies, we may focus on exploring different neural networks such as convolutional neural networks, recurrent neural network, long short-term memory, etc. We could also conduct experiments on different combinations of activation functions (such as tanh, leaky ReLU, etc) with various optimisation techniques (Newton's method, L-BFGS and other stochastic gradient-based methods).

V. Appendix

A. Instruction on how to run the code

The link to our code is:https://colab.research.google.com/drive/1Xlq5T5LDH38yob1FW5PoFerwsG2PmKsr?usp=sharing. There are two main components in this notebook:
1. PyTorch-free implementation (Building from scratch)
2. PyTorch implementation

The first one is the assignment 1 code for COMP5329, and the second one is only for comparasion prupose. This instruction is for the first implementation only.

1. The data is downloaded by the sharing id from one of the author's google drive.

2. Run section 1-Libraries Util, 2-Data Preprocessing and 3-Multilayer Perceptron Model sequencially, since they are only the defination of the functions and classes used in the notebook.

3. Run section 4-Training and Evaluation. 4.1 is the data preparation for the rest of the code, which should not be modified. 4.2 and 4.3 are the defination of functions for training and evaluating respectively, which will be used in the follwing parts. 4.4 is the main training and evaluating for different methods and techiques used in deep neural networks and please run them sequencially. Each subsection (4.4.x) is able to run separately. Please re-run 4.5 if you want to play around with the TensorBoard dashboard since the UI is not saved when saving this notebook.

B. Environment

This study is conducted entirely on Google Colab environment. The neccessary modules' versions are:
- python=3.7.13
- numpy=1.21.5
- matplotlib=3.2.2

However, this program should be solid with any Colab environment without changing startup settings manually.