ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/370277982

Techniques in Deep Learning: A Report

Technical Report · April 2023

DOI: 10.13140/RG.2.2.30086.65602/1

CITATIONS
0

READS
183

3 authors:

Chongyangzi Teng
The University of Sydney
4 PUBLICATIONS **0** CITATIONS
SEE PROFILE

Pengwei Yang
The University of Sydney
11 PUBLICATIONS **26** CITATIONS
SEE PROFILE

Mengshen Guo
The University of Sydney
2 PUBLICATIONS **0** CITATIONS
SEE PROFILE

All content following this page was uploaded by Pengwei Yang on 21 May 2023.

The user has requested enhancement of the downloaded file.

Techniques in Deep Learning: A Report

Chongyangzi Teng[1†], Pengwei Yang[1†*], and Mengshen Guo[1]

[1]*School of Computer Science, The University of Sydney, Sydney NSW 2000, Australia*
[†]*Equal contribution*
[*]*Corresponding author*

Abstract

In recent years, deep learning has expanded significantly in various real-world applications such as image and speech recognition. This technical report explores the influence of disparate values of distinct modules and assorted hyperparameters on the accuracy and loss outcomes by manually constructing a multilayer neural network classifier to ultimately identify the optimal configuration. The multilayer neural network model employed in this research encompasses several modules, comprising activation functions, momentum optimizers, weight decay, dropout, batch normalization, and cross-entropy loss.

1 Introduction

Multi-layer neural network also known as multi-layer perceptron (MLP) is one of the well-known modern modeling techniques. Its ability to handle non-linearly separable problems has made it a popular choice on a wide range of business problems. For example, image classification of fashion items for shopping platforms, face recognition for border security, detection of lung cancer on CT images, prediction of energy transmission in IoT crowdsourcing services, and more [16][20][21]. In addition, most deep learning model methods use multi-layer neural network as building blocks [7].

Due to the importance of multi-layer neural network, the primary purpose of this study is to develop understanding of its core concepts and algorithms. By the end of the study, we will understand key modules of a multi-layer neural network and can build it from scratch, can tune hyperparameters and finally be able to evaluate a neural network model.

In this technical report, our team will review the theoretical concept of the core modules of a multilayer neural network we will use in our experiment in section 2. We will include our experiment approach details in section 3. In our experiment, we will build a multilayer neural network model on a multi-class classification problem. Modules we will experiment in this study include hidden layer, activation function, loss criteria, weights initialization, optimization algorithms and regularization algorithm. We will evaluate and discuss our results in this section and finally will conclude our study in section 4.

2 Methodology

This section reviews the theoretical concept of the core modules of a multilayer neural network.

2.1 Pre-processing

In deep learning, pre-processing allows the input data to be scaled and transformed, which can solve the problem of features having different scales. And in this report, min-max normalization and normalization are the adopted pre-processing methods.

2.1.1 Missing value checking

We first check the missing value by making use of the function np.isnan().any() [8].

2.1.2 Min-max Normalization

Min-max normalization, which rescales the data to a specific range, usually [0, 1]. This technique ensures that all features have the same scale while preserving the original distribution of the data. The formula for min-max normalization is:

$$X_{norm} = \frac{x - min(x)}{max(x) - min(x)} \tag{1}$$

where $min(x)$ and $max(x)$ are the minimum value and maximum value of the input data, respectively.

2.1.3 Standardisation

Standardization is the transformation of the data so that the resulting distribution has a mean of 0 and a standard deviation of 1. The formula for Standardisation is:

$$X_{norm} = \frac{x - \mu}{\sigma} \tag{2}$$

where μ denotes the mean value and σ is the standard deviation.

2.2 Principles of modules

A multi-layer neural network model contains the three main components: input layer, hidden layer, and output layer:

- **Input layer**: Each neuron in the input layer is linked to one feature of the data sample. The dataset used in this study has 128 features, therefore there will be 128 neurons in this layer.
- **Hidden layer**: Multiple hidden layers can exist between input layer and output layer. For a fully connected network, each neuron in this layer can connect to all neurons from the previous and later layer. When a neuron receives inputs, it will calculate a weighted sum and apply an activation function to determine the value of the output.
- **Output layer**: This layer performs the required task such as prediction and classification. The number of neurons in this layer depends on the required tasks. In our studies, we want to build a classifier with ten target classes therefore ten neurons will be needed in the output, one for each class.

To train a multi-layer neural network, the following procedure is applied in our study:

1. Initialise all weights (include biases) (section 2.2.2)
2. Repeat these steps until stopping condition is satisfied:
   (a) Forward propagate an input: Data passes from the input layer to the output layer in a forward direction as illustrated in figure 1. This process is also known as forward propagation.
   (b) Compute the cost function (2.2.3)
   (c) Compute the gradients of the cost with respect to parameters using backpropagation algorithm (section 2.2.4)
   (d) Update each parameter using the gradients, according to the optimization algorithm (Section 2.2.5)
   (e) Check the stopping criteria at the end of each epoch.

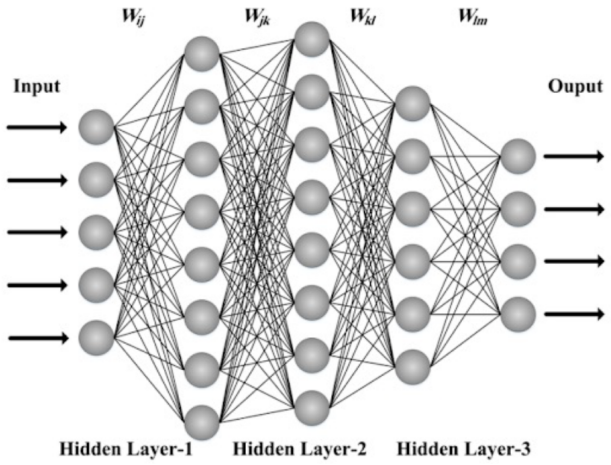

Figure 1: Multi-layer Neural Network

2.2.1 Activation function

Activation function transfers the weighted sum of input data of each neuron to an output value. It works to add non-linearity, without it every neuron will only be performing a linear classification [7]. Additionally, in real world problems, the relationship between the input and output is usually non-linear. As such the activation function must be non-linear to solve complex non-linear problems. In addition, it needs to be continuous and differentiable almost everywhere to help optimizers to find its extrema in the error surface. Popular choices of activation functions include Rectified linear unit (ReLU), Leaky ReLU and GELU will be explained in more detail in this section.

- **ReLU**: ReLU is a widely used activation function in DNNs. The function is defined as:

$$f(x) = max(0, x) \tag{3}$$

  where x is the input data. It can effectively address the gradient vanishing problem in contrast to previous activation functions such as sigmoid and tanh [2]. Nevertheless, ReLU simply gives 0 to the negative inputs, which means some neurons are not activated and thus be dead [17].

- **Leaky ReLU**: Leaky ReLUfunction are defined as follows:

$$f(x) = y_i \ (y_i > 0) \textbf{ or } f(x) = a(y_i) \ (y_i < 0) \tag{4}$$

  where y_i is the input of the nonlinear activation f on the i-th channel while a_i is a small and fixed value that can be seen as a coefficient controlling the slope of the negative part [9]. As opposed to ReLU that simply set output of the negative part to zero, Leaky ReLU introduces an extra parameter with a very small number. The slope is demonstrated to alleviate the dying ReLU problem [17].

- **GELU**: Gaussian Error Linear Unit, GELU, is a smooth approximation to the rectifier, which is defined as follows:

$$\textbf{GELU}(x) = xP(X \leq x) = x\Phi(x) = x \cdot \frac{1}{2}[1 + \textbf{erf}(x/\sqrt{2})] \tag{5}$$

where $\Phi(x)$ is the standard Gaussian cumulative distribution function. Compared to ReLU, GELU make use of percentile to input nonlinearity weights. As a result, the GELU can be seen as a smoother and more continuous ReLU, and thus more effective at learning complex patterns in the data [11].

2.2.2 Weight initialization

Weight initialization can be seen as an important step in deep neural models training process. The initial point can determine if the neural network converges or not. A poor weight initialization could

induce unstable training and thus encounters numerical difficulties and fails [7]. Various activation functions may require different initialization methods [6]. Based on the assumption of [6], [9] came up with an initialization algorithm to initialize the weight when making use of the ReLU and Leaky ReLU activation function. The central idea is to make $\mathbf{Var}(y) = \mathbf{Var}(\sum_i^n w_i x_i)$. As opposed to the Xavier initialization algorithm which is designed for tanh activation function [6], ReLU, and Leaky ReLU do not consider or only consider a minor part of the negative values. In this case, the variance for ReLU and Leaky ReLU are separately designed as $\frac{2}{n}$ and $\frac{2}{1+a^2}$ where n is the n the number of input data. However, for the initialization algorithm of GELU, as opposed to Kaiming initialization which take the arithmetic mean of these variances, [10] aims to make $\mathbf{Var}(z^l = 1)$ and $\mathbf{Var}(\delta^l = \mathbf{Var}(\delta^{l+1})$ where l is the layer depth.

2.2.3 Loss criteria

Cost function measures the differences between the target value and the predicted value. Typically, training a neural network model is to minimize the cost measured by the cost function. Depending on if the type of task is regression or classification, the choice of the cost function will be different. In this study, we will use SoftMax loss which is a softmax activation plus a cross-entropy loss 2. It is a common choice for multi-class classification. [5]

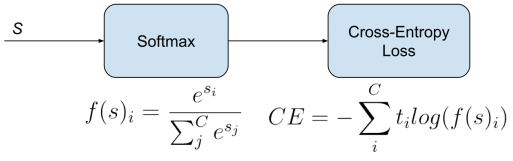

Figure 2: Softmax function and Cross-Entropy Loss [5]

In addition, one-hot encoding is used. This means there is only one non-zero element of the target vector, hence multi class labels can be transformed into binary values.

Softmax is a function, not a loss. It transforms a vector of real numbers into a vector of real numbers in the range of 0 to 1 which add up to 1. As such, when the data pass through the SoftMax activation function, a vector of predicted conditional probability distribution given x for each of the input classes k will be output. Figure 3 below illustrates its mechanics in a neural network classifier.

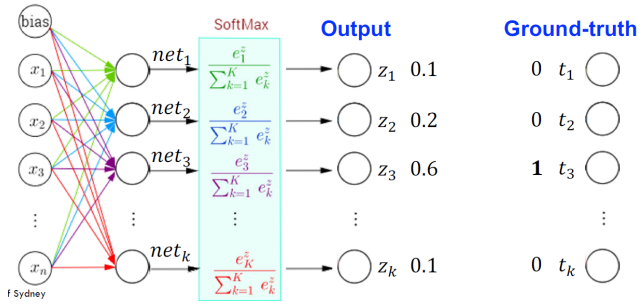

Figure 3: Softmax function in MLP [7]

Cross entropy loss is an extension of information theory, entropy, and measures the differences between two probability distributions for a given random variable and or events. It is closely related to Kullback-Leibler (KL) divergence which is a measure of relative entropy. Given two probability distribution t and z, the relationship is demonstrated in formula below:

$$
\begin{aligned}
CrossEntropy(t, z) &= -\sum_i t_i log z_i \\
&= -\sum_i t_i log t_i + \sum_i t_i log t_i - \sum_i t_i log z_i \\
&= -\sum_i t_i log t_i + \sum_i t_i log \frac{t_i}{z_i} \\
&= Entropy(t) + D_{KL}(t|z)
\end{aligned}
$$

2.2.4 Backpropagation algorithm

This is an algorithm used to train multi-layer perceptron neural networks. A loss value is calculated for the output of each neuron using the adopted loss function. The backpropagation algorithm then calculates the partial derivative of the loss function with respect to the weights in the network. It will then update weights of the network to reduce the loss. The weights adjustment is done backwards from the output neurons layer to the input neurons to minimise the loss across the entire training set. Therefore, this algorithm is also called backpropagation. One entire passing of training data through the algorithm is counted as one epoch, a stopping condition will be checked at the end of each epoch, if it does not meet the criteria the steps are repeated until stop criteria met. An example of the stop criteria is loss on the training set falls below a threshold.

2.2.5 Optimisation

Optimisation is a process for a neural network model to find its optimal parameters. After weights are initialised, the optimisation algorithm will start to update the weights to reduce the loss. This study will use the Stochastic Gradient Descent, Mini-batch Gradient Descent, Momentum and Adaptive Moment Estimation (Adam).

Stochastic Gradient Descent

Stochastic gradient descent is a variation of the basic gradient descent. The basic gradient descent updates the weights once after passing through all the data. However, the stochastic gradient descent updates the wights after each sample.

Mini-batch Gradient Descent

Mini-batch gradient descent is also a cross over of gradient descent and stochastic gradient descent. It divides the training samples into mini batches and updates the model weights after every mini batch of training samples. Advantages of mini-batch gradient descent relative to stochastic gradient descent are it can use vectorized implementation therefore faster computations and also allows smaller memory in algorithm implementations[4][7].

The batch size is a hyperparameter in this algorithm that specifies the number of samples processed before the weights are updated. The batch size parameter must be more than 1 and less than or equal to the total number of samples in the training data. In our experiment, there is a training sample size of 50,000, if a batch size of 2 is chosen, it means the dataset will be divided into 25,000 batches and each epoch will involve 25,000 updates to the model weights.

A Pseudocode for mini batch is shown in figure 4

Momentum

There are two issues with gradient descent. Firstly, the model can be stuck at saddle points, one dimension slopes up and another dimension slopes down. This is because at saddle point, the gradient of cost function is close to zero therefore can lead to small or no weight updates. Secondly, the update path followed is unable with oscillations[3][19].

Momentum can be used to overcome the two issues. It does this by adding a momentum term ,gamma, to increase the weights updates when gradients point in the same direction and reduces weights updates when gradients change directions. This therefore reduces the oscillation and gains faster convergence. Momentum term is usually set to 0.9 [18].

Function SGD:
  Set epsilon as the limit of convergence
  for $i = 1, \ldots, b$ **do**
    for $j = 0, \ldots, n$ **do**
      while $|\omega_{j+1} - \omega_j| < epsilon$ **do**
        $\omega_{j+1} := \omega_j - \alpha \cdot \frac{1}{b} \sum_{k=i}^{i+b-1} (h_\omega(x^{(k)}) - y^{(k)}) \cdot x_j^{(k)};$
      end
    end
  end

Figure 4: Pseudocode for Mini-batches

The weight update formula for momentum is:

$$\nu_t = \nu_{t-1} + \eta \nabla_\theta J(\theta)$$
$$\theta_t = \theta_{t-1} - \nu_t$$

Figure 5 demonstrates the effect of momentum graphically.

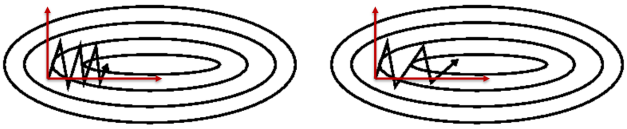

Figure 5: Convergence with and without Momentum[1]

Adaptive Moment Estimation (Adam)

Adam is an advanced optimisation algorithm that computes adaptive learning rates for different parameters. It is first introduced by Kingma et. al. in 2014 [15].

It can keep an exponentially decaying average of past squared gradients ν_t and an exponentially decaying average of gradients m_t. The convergence of Adam behaves like a heavy ball with friction and thus prefers flat minima in the cost surface [12].

The advantage of the algorithm is it is efficient in memory and computation, simple to implement and suitable for large datasets and more parameters.

A Pseudocode for Adam is shown below figure 6. The proposed default value by the Kingma et.al. is $\alpha = 0.001$, $\beta 1 = 0.9$ and $\beta 1 = 0.999$.

2.2.6 Regularisation

In the process of deep learning, overfitting is a problem that is often encountered, and regularization is a technique used to prevent overfitting. The following regularization techniques are used in the report.

- **Weight decay**: Weight decay is essentially a variant of L2 regularization. During gradient descent, the weights are multiplied by a factor slightly less than 1 before being updated, which encourages smaller weights to effectively apply L2 regularization.
- **Dropout**: Dropout is a regularization technique specifically for neural networks. During the training phase, Dropout randomly ”drops” many hidden neurons to prevent their contribution to the forward and backward directions. This process encourages the model to develop more robust and redundant representations by not relying on individual neurons.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
$m_0 \leftarrow 0$ (Initialize 1st moment vector)
$v_0 \leftarrow 0$ (Initialize 2nd moment vector)
$t \leftarrow 0$ (Initialize timestep)
while θ_t not converged **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
$\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
$\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
end while
return θ_t (Resulting parameters)

Figure 6: Pseudocode for Adam

- **Batch normalization**: Batch normalization is a way to make neural networks training process faster and more stable by re-centering and re-scaing. The internal covariate shift, i.e., the distribution of each layer's input changes during training, slows down the training process by requiring lower learning rates and makes it quite hard to train models with saturating nonlinearities. As performing mini-batch normalization, the neural network model can make use higher learning rates. The main process in model training are defined as follows:

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i}^{m} x_i \tag{6}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i}^{m} (x_i - \mu_{\mathcal{B}})^2 \tag{7}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \tag{8}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \mathbf{BN}_{\gamma,\beta}(x_i) \tag{9}$$

where μ_B is the mini-batch mean, σ_B^2 denotes mini-batch variance. Formula 8 and 9 respectively demonstrate the normalization and scale and shift process in the mini-batch. Additionally, to track the accuracy of the model as it trains, an unbiased variance estimate $\mathbf{Var}[x] = \frac{m}{m-1} \cdot E[\sigma]_{\mathcal{B}}^2$ is then implemented, where the expectation is over training mini-batches of size m and $\sigma_{\mathcal{B}}^2$ are the variances of their sample [13].

2.3 Design of best model

Our best model design is shown in the table 1 below. Justification of the model choices are discussed in the experiment section of this report.

3 Experiment and Results

3.1 Experiment setup

In this experiment, we will build multi-layer neural network models with different designs on a multi-class classification problem. The data provided has been split into a training and a testing set, with 50,000 and 10,000 samples respectively. There are ten classes with labels from 0 to 9 and are evenly distributed in both training and testing sets. Each of the samples has 128 features with value ranges from -24 to 26. There are no extreme values and no obvious noise observed in the dataset.

Table 1: Best Model Design

Modules	Baseline Model	Best Model
Weight initilisation	Kaiming	Kaiming
Pre-processing	No	No
Loss Function	Softmax and cross-entropy loss	Softmax and cross-entropy loss
Learning rate	0.0001	0.001
Hidden layer	2	1
Hidden unit	[128,128]	[512]
Activation	ReLU	Leaky ReLU
Optimiser	Adam	Adam
Batch size	64	128
Dropout	0	0
Weight decay	0	0.01
Batch Norm	TRUE	TRUE

A baseline classifier model will be set-up with design features shown in table 1. The experiment will then perform parameter analysis using this baseline model. The findings from the parameter analysis, ablation studies and comparison methods will be used to build a best model.

3.2 Performance metrics

The performance of each classier will be evaluated with:

- The top-1 accuracy metric, that is,

$$top-1\ accuracy = \frac{\textit{number of correctly classified examples}}{\textit{total number of test examples}} * 100\%$$

- Cross-entropy loss

3.3 Parameter Analysis Results

This section details the analysis of the effect changing different model parameters on the baseline model.

3.3.1 Learning Rate

Learning rate is a key hyper-parameter in the optimiser that controls the degree of changes to the model weights in response to the loss reduction. If a learning rate is too small, the model can take too long to train, and on the other hand if a learning rate is too large, it could lead the model to miss the optimal point and result in a sub-optimal weight.

From the learning rate hyper-parameter analysis using the baseline model architecture, figure 7 and figure 8, we can see the model with learning rate of 0.01 fails to converge to optimal weights and the performance is indifferent to a random guess of 10 percent accuracy. For the baseline model setting, the performance curve for 0.0001 and 0.001 learning rates follows very similar trends, losses reduce quickly in the first 10 epochs and then start to converge. Both 0.001 and 0.001 reached similar performance levels at epoch 25.

3.3.2 Batch size

As mentioned in the methodology section, the batch size is a hyperparameter of mini-batch gradient descent that specifies the number of samples processed before the weights are updated.

We have tested the effect of batch size on the baseline model using the following values: 2, 8, 16, 32, 64, 128 and 256. Figure 9 and 10 shows testing data accuracy and cross entropy losses. The result shows with a baseline learning rate of 0.0001, higher batch size such as 256 and 128 have better performance compared to smaller batch size such as 2 and 8. The performance of batch size

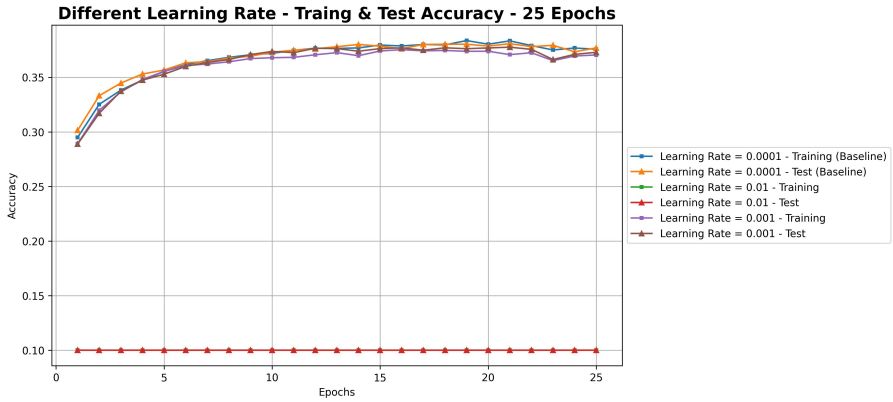

Figure 7: Different Learning Rate - Test Accuracy

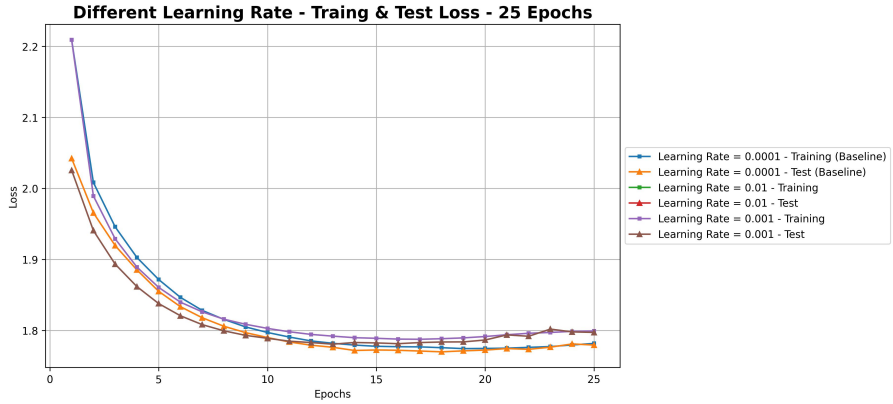

Figure 8: Different Learning Rate - Test Losses

64, 128 and 256 converged to a similar level at 25 epochs. The performance curves are smooth with small incremental improvements, suggesting the models are not stuck in local minimum.

In general, increasing batch size will reduce the model's capacity to generalise and tend to result in models becoming caught in local minima [14]. Comparing to the training sample size in the experiment, our batch size choices are all relatively small. If time allows, we will explore more options with larger batch size to demonstrate this effect.

3.3.3 Number of hidden units

The effect of the number of hidden units is tested using baseline model design of two hidden layers. Figure 11 and 12 shows models with more hidden units converge faster and perform better in less epochs. Around epoch 10, apart from [32,32] and [128,128] all other models show small deterioration in test performance, this is suggesting complicated models with more neuron units tend to overfit if an early stopping criterion is not set. The baseline model with 128 units in each hidden layer has steady improvement, however by epoch 25 its performance is still at lower level compared to models with more hidden units.

3.3.4 Number of hidden layers

The effect of number of hidden layers is shown in figure 13 and 14. We have found 1 hidden outperforms the model with 2 hidden layers.

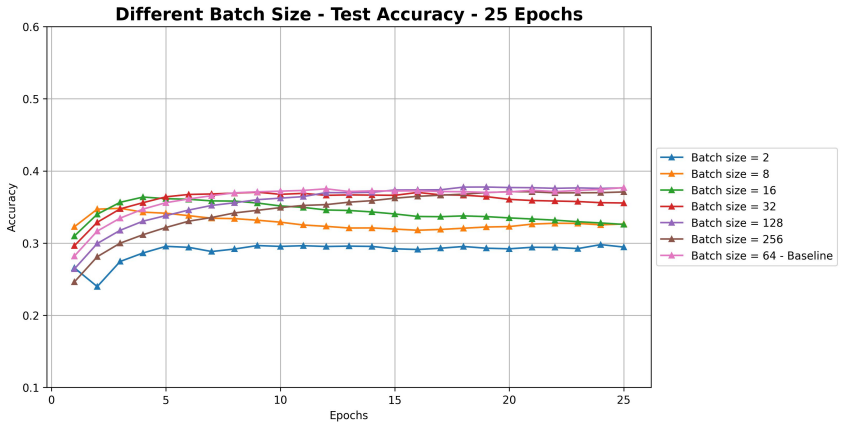

Figure 9: Different Batch Size - Test Accuracy

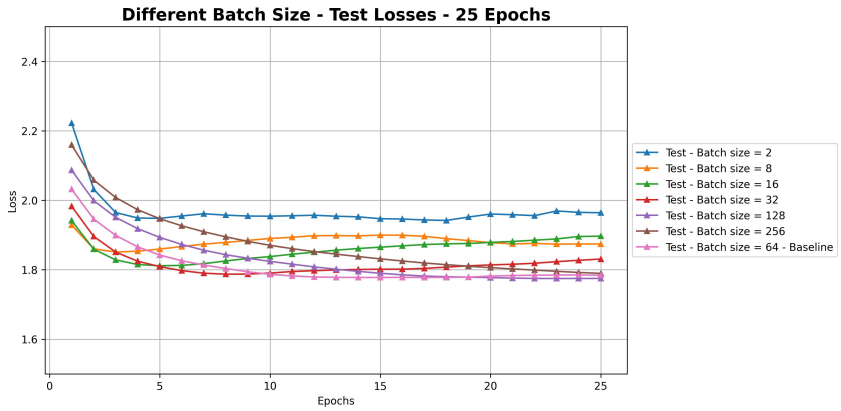

Figure 10: Different Batch Size - Test Losses

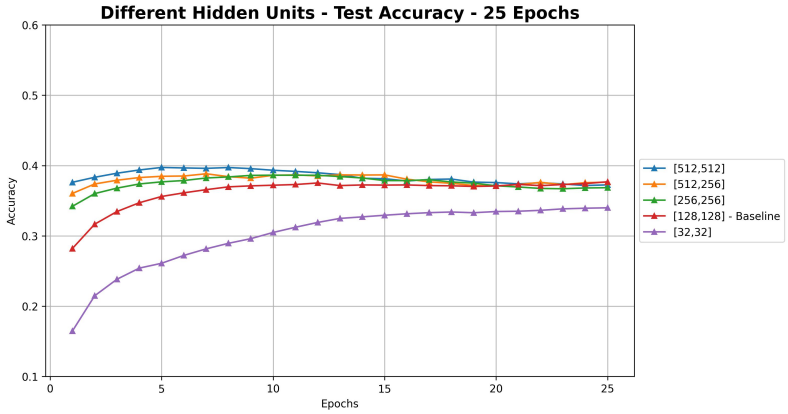

Figure 11: Different Hidden Units - Test Accuracy

Figure 12: Different Hidden Units - Test Losses

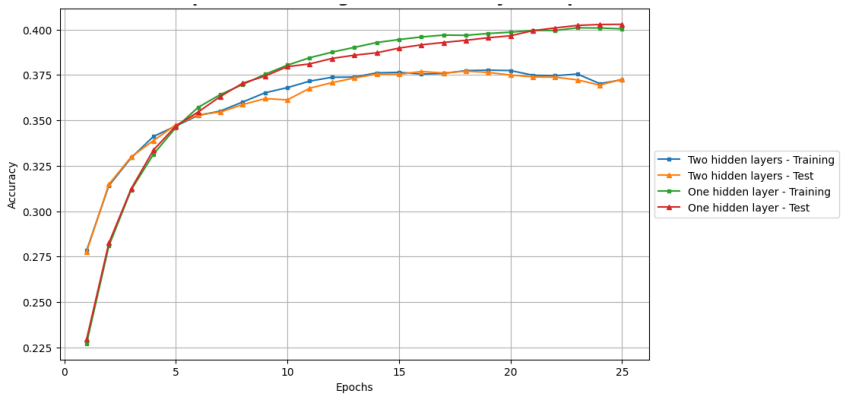

Figure 13: Hidden layer number - accuracy

3.3.5 Optimiser

An optimizer in deep learning is an algorithm that adjusts a model's parameters (weights and biases) to minimize a loss function. We compared Adam, SGD(Momentum = 0) and SGD(Momentum = 0.9). From the result, it is very clear that Adam, as the optimiser, has the highest and smooth

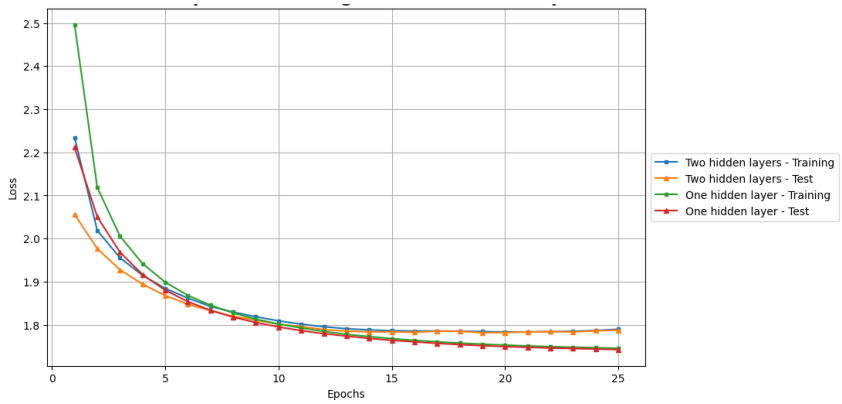

Figure 14: Hidden layer number - loss

accuracy curve with the accuracy of about 38 percent. When the number of iterations reaches 25, the accuracy of Adam still has a slow upward trend. At the same time, in terms of loss, Adam and SGD (Momentum=0) are very similar, both have very smooth and low loss values. Therefore, Adam is the best optimizer.

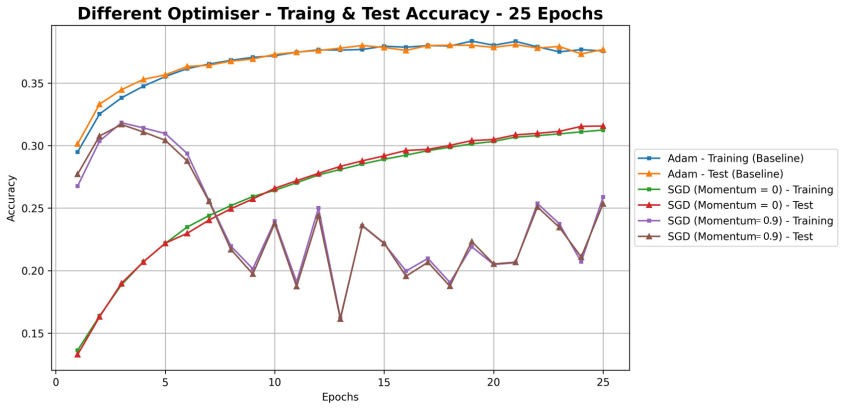

Figure 15: Different Optimiser - Test Accuracy

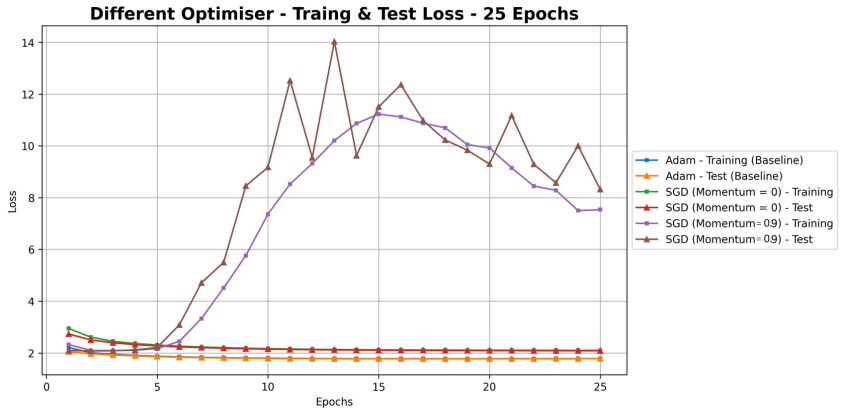

Figure 16: Different Optimiser - Test Losses

3.3.6 Dropout

Dropout is a regularization technique specific to neural networks. In this report, dropout=0 is compared with dropout=0.5. As can be seen from the figure, the trends of the accuracy curves of the two are very similar, but overall, the accuracy of the model with dropout=0 is slightly higher, about 38 percent. Similarly, the loss curves of the two dropouts are also very detailed, both in value and trend. Therefore, dropout=0 is the better hyper-parameter choice.

3.3.7 Weight decay

In this experiment, four values of weight decay were compared: False, 0.005, 0.001 and 0.01. In terms of accuracy, the four weight decays were very close in terms of accuracy and trend, but the model reached the highest accuracy of about 38 percent when weight decay=0.01. Similarly, the loss curves of all four weight decays overlap. Therefore, weight decay=0.01 is the best. Figure 19 and 20 shows the results.

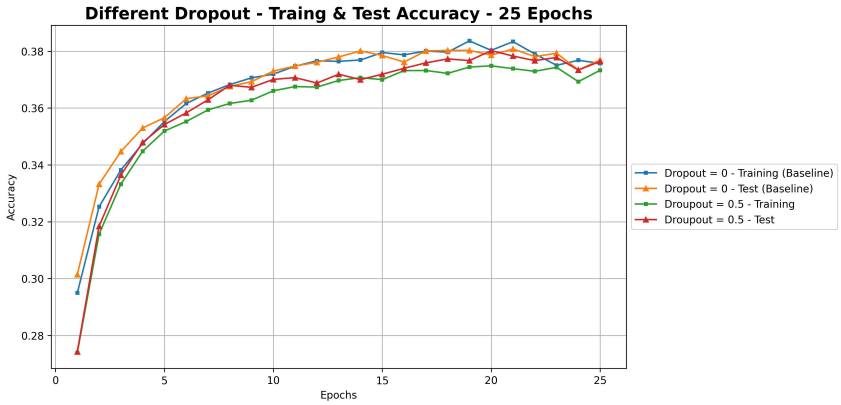

Figure 17: Different Dropout - Test Accuracy

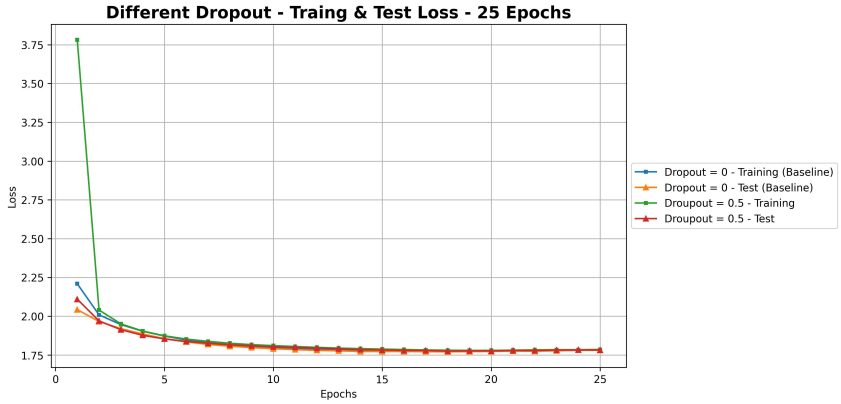

Figure 18: Different Dropout - Test Losses

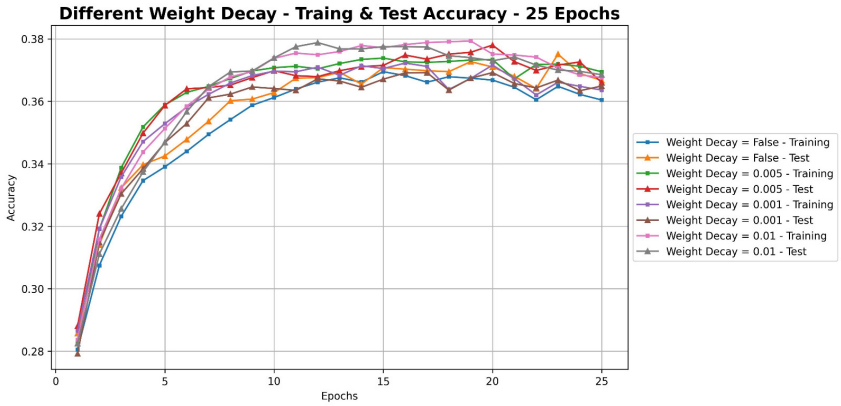

Figure 19: Different Weight Decay - Test Accuracy

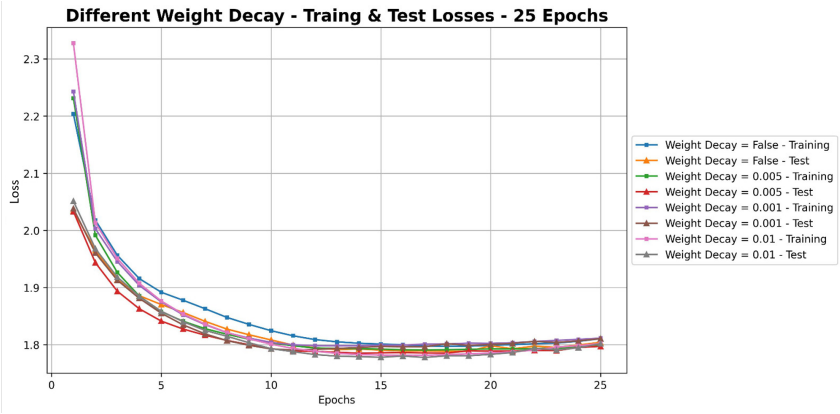

Figure 20: Different Weight Decay - Test Losses

3.3.8 Activation function

We compare the model performance regarding to ReLU, Leaky ReLU and GELU activation functions. The comparison experiment results of the activation function can be seen in Figure 21 and 22. It is easy to discover that the leaky ReLU outperforms the other two activation functions.

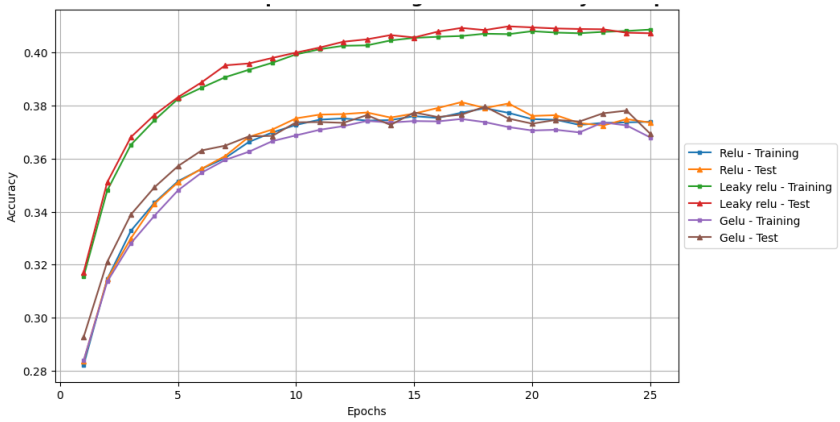

Figure 21: Activation function comparison - accuracy

3.4 Best Model

Our baseline model utilised Kaiming weight initialisation, ReLU activation and Adam optimiser. Figure 23 below compares the performance between the models using different combinations of model settings.

From the results, we find a model with 1 hidden layer (model 1) gives better performance than two layers if all parameters are the same. Model 2 which has more hidden units than baseline model can better the data. As such, in the final model we used 1 hidden layer with 512 units.

From the activation function analysis and results of Model 3 and Model 4, we found leaky ReLU has the best performance, therefore selected for the final model.

From the optimiser analysis and results of Model 5, we found Adam outperformed all other settings we have tried, thus we kept using Adam in the final model.

We also investigated different regularisation modules settings. We found that with dropout of 0.5 has no clear improvement to accuracy relative to the baseline model and weight decay of 0.01 gives

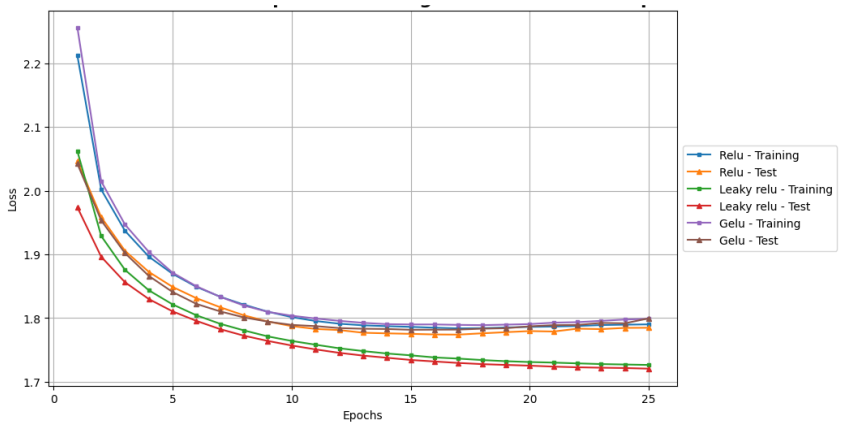

Figure 22: Activation function comparison - loss

the better results, shown in model 6 and 7 respectively. As such, we adopted no dropout and weight decay of 0.01 in our final model.

We also performed further analysis to compare a combination of different learning rate and batch size using the final model architecture mentioned above. The model with batch size of 128 and learning rate of 0.001 gives the best accuracy as shown in Figure 24. Therefore this model is selected as our best model.

More details on the best model can be find in figure 25.

Modules	**Baseline**	**Model 1**	**Model 2**	**Model 3**	**Model 4**	**Model 5**	**Model 6**	**Model 7**	**Best Model**
Weight initialisation	Kaiming								
Preprocessing	N/A								
Loss Function	Softmax and cross-entropy loss								
Learning rate	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.001
Hidden layer	2	**1**	2	2	2	2	2	2	1
Hidden unit	[128,128]	[128]	**[512,512]**	[128,128]	[128,128]	[128,128]	[128,128]	[128,128]	[512]
Activation	ReLU	ReLU	ReLU	**Leaky ReLU**	**GeLU**	ReLU	ReLU	ReLU	Leaky ReLU
Optimiser	Adam	Adam	Adam	Adam	Adam	**SGD (M= 0)**	Adam	Adam	Adam
Batch size	64	64	64	64	64	64	64	64	128
Dropout	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	**0.5**	FALSE	FALSE
Weight decay	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	**0.01**	0.01
Batch Norm	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
Epoch	25	25	4	25	25	25	25	25	8
Test Accuracy	**0.3769**	**0.4021**	**0.3973**	**0.4073**	**0.3694**	**0.3157**	**0.3764**	**0.3786**	**0.4369**
Acc diff to Baseline		**+0.025**	**+0.020**	**+0.030**	**-0.008**	**-0.061**	**-0.001**	**+0.002**	**+0.060**

Figure 23: Model Results Comparison

4 Discussion and Conclusion

This study firstly reviewed the theoretical concept of the core modules of a multilayer neural network in section 2. An experiment has been performed on a multi-class classification problem to further investigate the effect of these key modules. Finally, we have then developed a classifier that comprises one hidden layer with 512 neurons. The classification accuracy of the model is 0.4369 percent.

In terms of effect analysis and performance comparison, there are a few things we could improve for future learning. Firstly, we have only explored a limited range of different hyper-parameters, a few more extreme values could be tested to demonstrate the effect if an inappropriate parameter is used. For rigorous performance comparison, we should train each classifier multiple times with different

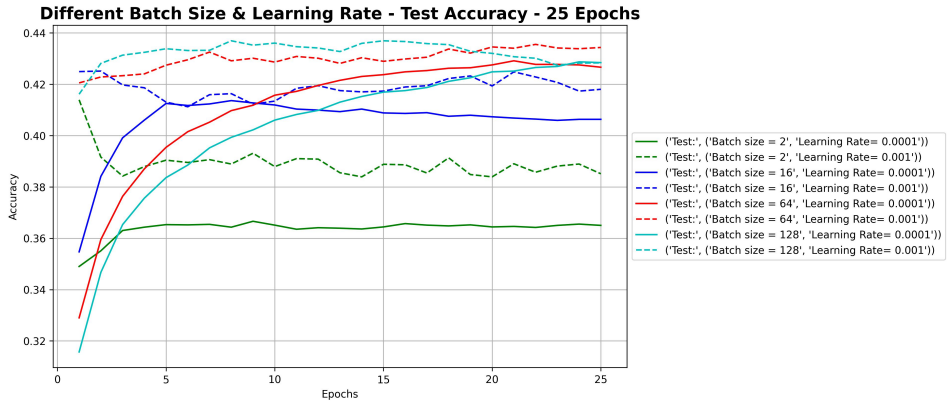

Figure 24: Best model with different batch size and learning rate

Epoch	Run Time (s)	Train Loss	Train Accuracy	Test Loss	Test Accuracy
1	50.06685853	1.828146623	0.42364	1.695859562	0.4161
2	48.94397783	1.680083191	0.43394	1.656303894	0.4281
3	50.60314727	1.650175586	0.43886	1.639720023	0.4313
4	48.97775674	1.63739368	0.44044	1.630685942	0.4324
5	50.73535585	1.631070466	0.44308	1.626332683	0.4338
6	49.05120325	1.628171412	0.44298	1.625489991	0.4331
7	50.74006724	1.626116863	0.44166	1.622522354	0.4332
8	49.05703521	1.623300082	0.44088	1.61855017	0.4369
9	50.52125645	1.620321726	0.441	1.616997583	0.4352
10	49.13927937	1.618690878	0.44184	1.617695335	0.436
11	57.19165301	1.61813441	0.44172	1.618493919	0.4346
12	49.20878601	1.618103914	0.44128	1.619528465	0.4341
13	49.88432956	1.618108273	0.44146	1.620301148	0.4327
14	48.55665779	1.61818396	0.44056	1.621032194	0.4359
15	50.27676821	1.618639429	0.44082	1.6215014	0.4369
16	48.77927232	1.619393267	0.43998	1.623493984	0.4366
17	50.41065335	1.620251912	0.43952	1.625881656	0.4358
18	49.11570835	1.621522774	0.43846	1.629006178	0.4354
19	50.07705569	1.622463973	0.43738	1.631679931	0.4328
20	49.56415105	1.62238886	0.43784	1.633072648	0.432
21	48.95404124	1.622074994	0.43702	1.634426834	0.4307
22	50.31319427	1.622148343	0.43758	1.634815122	0.4301
23	49.04960084	1.622014494	0.43762	1.636427824	0.4274
24	50.66926837	1.621823435	0.43618	1.637299098	0.4282
25	49.0390377	1.622159918	0.43562	1.638178347	0.4283

Figure 25: Evaluation Details of the Best Model

training and validation sets generated by random sampling. The average and standard deviation of the test accuracy should then be reported for result comparison purposes.

In terms of model development, to find the best model, a few more combinations of parameters or a more thorough grid search approach should be performed, this is because different modules can have interactive effects on each other.

References

[1] Momentum and learning rate adaptation, https://www.willamette.edu/~gorr/classes/cs449/momrate.html

[2] Agarap, A.F.: Deep learning using rectified linear units (relu). arXiv preprint arXiv:1803.08375 (2018)

[3] Bhat, R.: Gradient descent with momentum (10 2020)

[4] Brownlee, J.: A gentle introduction to mini-batch gradient descent and how to configure batch size (07 2017)

[5] Bruballa, R.G.: Understanding categorical cross-entropy loss, binary cross-entropy loss, softmax loss, logistic loss, focal loss and all those confusing names

[6] Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the thirteenth international conference on artificial intelligence and statistics. pp. 249–256. JMLR Workshop and Conference Proceedings (2010)

[7] Goodfellow, I., Bengio, Y., Courville, A.: Deep learning. MIT press (2016)

[8] Harris, C.R., Millman, K.J., van der Walt, S.J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N.J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M.H., Brett, M., Haldane, A., del Río, J.F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., Oliphant, T.E.: Array programming with NumPy. Nature **585**(7825), 357–362 (Sep 2020). https://doi.org/10.1038/s41586-020-2649-2

[9] He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: Proceedings of the IEEE international conference on computer vision. pp. 1026–1034 (2015)

[10] Hendrycks, D., Gimpel, K.: Adjusting for dropout variance in batch normalization and weight initialization. arXiv preprint arXiv:1607.02488 (2016)

[11] Hendrycks, D., Gimpel, K.: Gaussian error linear units (gelus). arXiv preprint arXiv:1606.08415 (2016)

[12] Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., Hochreiter, S.: Gans trained by a two time-scale update rule converge to a local nash equilibrium (2018)

[13] Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: International conference on machine learning. pp. 448–456. pmlr (2015)

[14] Keskar, N.S., Mudigere, D., Nocedal, J., Smelyanskiy, M., Tang, P.T.P.: On large-batch training for deep learning: Generalization gap and sharp minima. CoRR **abs/1609.04836** (2016)

[15] Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization (2017)

[16] LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. nature **521**(7553), 436–444 (2015)

[17] Maas, A.L., Hannun, A.Y., Ng, A.Y., et al.: Rectifier nonlinearities improve neural network acoustic models. In: Proc. icml. vol. 30, p. 3. Atlanta, Georgia, USA (2013)

[18] Ruder, S.: An overview of gradient descent optimization algorithms (01 2016), https://www.ruder.io/optimizing-gradient-descent/#momentum

[19] Sutton, R.S.: Two problems with backpropagation and other steepest-descent learning procedures for networks. In: Proceedings of the Eighth Annual Conference of the Cognitive Science Society. Hillsdale, NJ: Erlbaum (1986)

[20] Yang, P., Abusafia, A., Lakhdari, A., Bouguettaya, A.: Energy loss prediction in iot energy services. In: Proceedings of the IEEE international conference on web services. IEEE (2023)

[21] Yang, P., Abusafia, A., Lakhdari, A., Bouguettaya, A.: Monitoring efficiency of iot wireless charging. In: PerCom. IEEE (2023)

5 Appendix

5.1 Code Link

https://github.com/Pengwei-Yang/Deep-Learning/blob/main/Techniques_in_Deep_Learning_A_Report.ipynb

5.2 Code running instructions:

More detailed step-by-step instruction can be found in the Python code file.

The file will run in Python 3 requiring only the legal libraries as per the technique report description. Their download/inclusion is a part of our technique report file.

5.3 Hardware:

The experiment is conducted on Google Colab. There are two CPUs @ 2.20GHz. Both CPU has the same specifics, the below figure shows detailed information.

```
processor   : 0
vendor_id   : GenuineIntel
cpu family  : 6
model       : 79
model name  : Intel(R) Xeon(R) CPU @ 2.20GHz
stepping    : 0
microcode   : 0xffffffff
cpu MHz           : 2199.998
cache size  : 56320 KB
physical id : 0
siblings    : 2
core id           : 0
cpu cores   : 1
apicid            : 0
initial apicid    : 0
fpu         : yes
fpu_exception     : yes
cpuid level : 13
wp          : yes
flags       : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat
pse36 clflush mmx fxsr sse sse2 ss ht syscall nx pdpe1gb rdtscp lm constant_tsc
rep_good nopl xtopology nonstop_tsc cpuid tsc_known_freq pni pclmulqdq ssse3 fma
cx16 pcid sse4_1 sse4_2 x2apic movbe popcnt aes xsave avx f16c rdrand hypervisor
lahf_lm abm 3dnowprefetch invpcid_single ssbd ibrs ibpb stibp fsgsbase tsc_adjust
bmi1 hle avx2 smep bmi2 erms invpcid rtm rdseed adx smap xsaveopt arat md_clear
arch_capabilities
bugs        : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf mds swapgs
taa mmio_stale_data retbleed
bogomips    : 4399.99
clflush size      : 64
cache_alignment   : 64
address sizes     : 46 bits physical, 48 bits virtual
power management:
```

Figure 26: Hardware details

5.4 Software:

- Python version: Python 3.9.16
- Packages used: numpy, matplotlib, time

View publication stats