# Multilayer Neural Network

Yang Ge (450005028), Suwan Zhu (480090900), Tianyao Xu (480268059)

**Abstract**

Deep learning is part of machine learning methods based on different layers in Artificial Intelligence. In this assignment, we implemented Multilayer perceptron, which is a class of artificial neural network, and other different modules. This report will show our experiments processes, results and discuss the performance of different modules, which are ReLU activation, Weight decay, Momentum in SGD, Dropout, softmax and cross-entropy, mini-batch training, Batch Normalization in machine learning and deep learning. The report is organized as follows. The introduction section will discuss the aim of the study and its importance. The methods section will illustrate pre-processing method and the principle of different modules. The experiment section will show the experiment processes and results, followed by conclusions and references.

**Keywords**: Machine learning, Deep learning, Classification, ReLU activation, Weight decay, Momentum in SGD, Dropout, Softmax and cross-entropy, Mini-batch training, Batch Normalization

## 1.Introduction

**1.1 The aim of the study.** Deep learning is a powerful machine learning method. The aim of this study is to further explore the nature of artificial neural network and related algorithms. The given data set has 60000 data and 10 different classes. We divided them into training data set and testing data set. We implemented MLP classifier on the given data set. Then we developed ReLU activation, Weight decay, Momentum in SGD, Dropout, Softmax and cross-entropy, Mini-batch training and Batch Normalization, learnt how they can improve our MLP model and tested their performance in the experiments.

**1.2 The importance of the study.** Deep Learning has generated huge breakthroughs in machine learning field over the last few years, and it is widely used in real world. Currently, deep learning is applied in automatic speech recognition, image recognition, natural language processing, recommendation systems and many other areas. It is important to learn how deep learning and artificial neural network work in this study. Deep learning contains many methods and parameters to generate good performance, understanding and implementing them are also the significance of this study.

## 2. Methods

**2.1 Pre-processing.**

**Standardization.** Feature standardization is to set mean of each feature in dataset to 0 and variance to 1. It is a common requirement for many machine learning method. The standard score of a sample x can be calculated as:

$$x' = \frac{x - \overline{x}}{\sigma} \tag{1}$$

where x is the original sample, $\overline{x}$ is the mean of training samples, $\sigma$ is the standard deviation of training samples.

**2.2 Activation function.** Activation functions are functions used in neural networks to computes the weighted sum of input and biases, of which is used to decide if a neuron can be fired or not(reference). The activation function of a neuron defines the output of that neuron and the output will be used as input of next neuron. It works like mapper which maps the input values into the desired range such as 0 to 1, -1 to 1. There are basically two types of Activation Function, which are Linear Activation Function and Non-linear Activation Functions.

**Sigmoid Function.** Sigmoid Function is a non-linear AF and It's curve looks like a S-shape.

Sigmoid Function:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2}$$

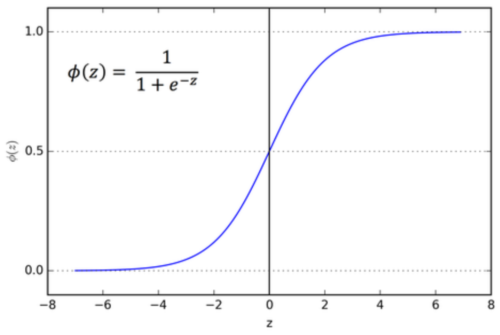

*Figure 1.* graph for Sigmoid function

Sigmoid Function is used for the model where we need to predict the probability as an output because it maps the input values into the range of 0 to 1.

**Tanh Function.** Tanh Function is also a non-linear AF and s - shaped curve. Tanh Function:

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{3}$$

Tanh function maps the input values into the range of -1 to 1 and the advantage of it is that the negative inputs will be mapped strongly negative and the zero inputs will be mapped near zero.

**Rectified Linear Unit (ReLU) Function.** ReLU Function Relu(x) is x when the x is above or equal to zero, otherwise is zero. It's also a non-linear AF. ReLU Function:

$$Relu(x) = max(0, x) \tag{4}$$

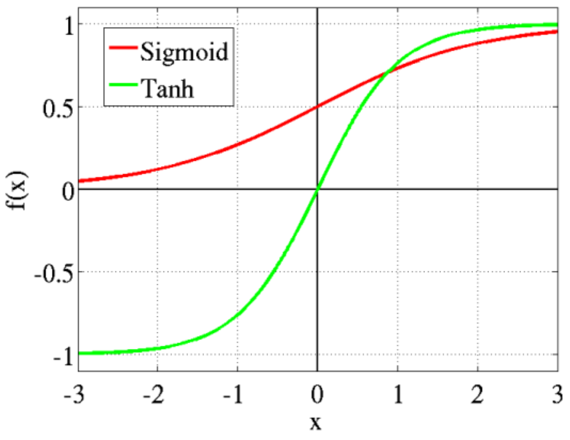

*Figure 2.* graph for Tanh function

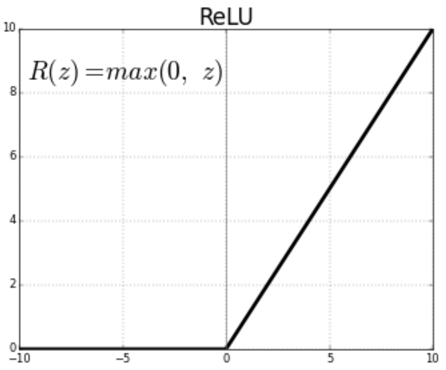

*Figure 3.* graph for ReLU function

ReLU Function can map values input the range of 0 to infinity but it can not map negative values appropriately.

**Softmax Function.** Softmax Function is a logistic activation function and it is most used for multiclass classification. The Softmax regression is a form of logistic regression that normalizes an input value into a vector of values that follows a probability distribution whose total sums up to 1(reference). In multiclass classification, Softmax function assigns input value conditional probabilities to each class.

Softmax Function:

$$\sigma(j) = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \tag{5}$$

Compared with Sigmoid function, Softmax is

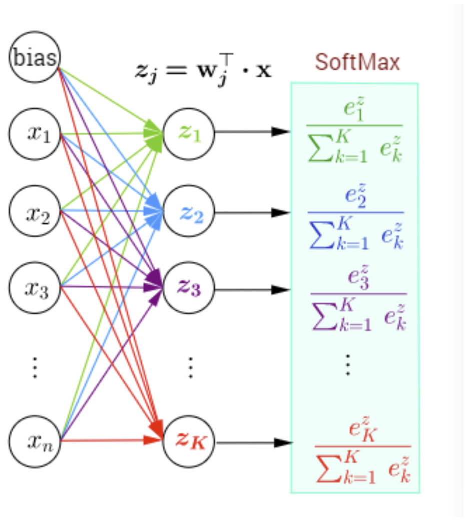
bias
$x_1$
$x_2$
$x_3$
$x_n$
$z_j = \mathbf{w}_j^\top \cdot \mathbf{x}$
$z_1$
$z_2$
$z_3$
$z_K$
SoftMax
$\frac{e_1^z}{\sum_{k=1}^K e_k^z}$
$\frac{e_2^z}{\sum_{k=1}^K e_k^z}$
$\frac{e_3^z}{\sum_{k=1}^K e_k^z}$
$\frac{e_K^z}{\sum_{k=1}^K e_k^z}$

*Figure 4.* graph for Softmax function

optimal way that estimates maximum-likelihood. It gives a probability distribution over all different possible classes.

**2.3 Cross-entropy Loss.** For a neural network classifier with softmax output node activation, during training you can use mean squared error or cross-entropy error, and cross-entropy error is considered slightly better(ref). Cross-entropy Loss:

$$J(t, z) = -\sum_{k=1}^{C} t_k \log_{Z_k} \tag{6}$$

t and z are two probability distributions.

**2.4 Dropout.** Dropout is applied to alleviate the occurrence of over-fitting When a complex feedforward neural network is trained with a small data set. Dropout let some neurons stop working with a certain probability When the neural network forward, which makes the model more generalized because it won't be influenced too much by some local features.

**2.5 Weight decay.** Weight decay is a coefficient placed in front of the regularization in the loss function and it aims to to adjust the influence of model complexity on the loss function. Generally, regular term indicates the complexity of the model.If the weight decay is large, the value of the complex model loss function is also large. The weight decay works like a penalty for L2-Normal.

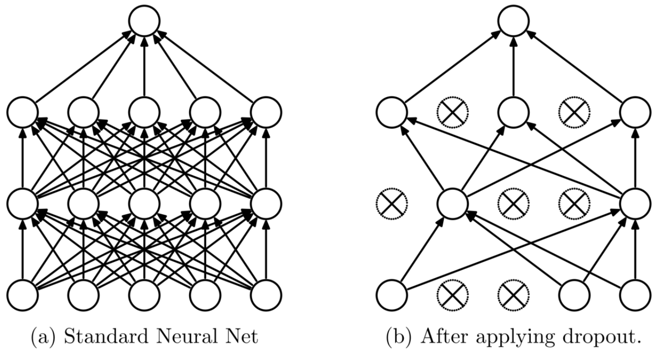

*Figure 5.* graph for Dropout

Assume C is a loss function with L2-normal, C0 is the original loss function.

$$C = C_0 + \frac{\lambda}{2n} \sum_w w^2 \tag{7}$$

We add a coefficient lambda placed in front of the regularization as a penalty of complex model.

**2.6 Mini-batch training.** Mini-batch gradient decent is a compromise method to overcome the shortcomings of Batch gradient descent and stochastic gradient descent.

Batch gradient descent need to traverse all the data sets to calculate the loss function, then calculating the gradient of the function to each parameter, and updating the gradient. This method must look at all the samples in the data set every time the parameter is updated.The cost is large and speed is slow.

In Stochastic gradient descent, we can calculate the loss function and ask the gradient to update the parameters by just using one data. This method is faster, but the convergence performance is not so good, it may wobble around the optimized point but can't get it.

In Mini-batch gradient decent divides the data into several batches and updates the parameters in batches. A set of data in a batch determines the direction of the gradient, and it is not easy to go off and reduce randomness. As the number of samples in the batch is much smaller than the entire data set, the amount of calculation is not very large.

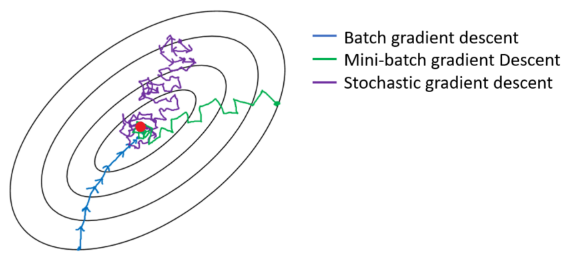

*Figure 6.* graph for Mini-batch gradient descent

**2.7 Momentum.** Momentum in gradient decent is a method to increase chance of reaching global optimized point rather than local optimized point. Momentum is mainly used when weights are updated in gradient decent.

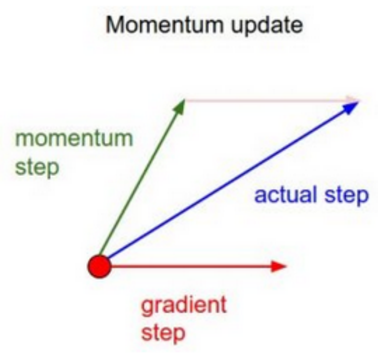

*Figure 7.* graph for Momentum in gradient descent

Generally, when the neural network updates the weight, the following formula is used:

$$w = w - learningrate * dw \tag{8}$$

After introducing the momentum, adopt the following formula:

$$\begin{aligned} v &= mu * v - learningrate * dw \\ w &= w + v \end{aligned} \tag{9}$$

Among this equation, v is initialized to 0, mu is a super variable set and the most common setting is 0.9. The formula means if the last momentum(v) is the same as the negative gradient direction this time, the magnitude of this drop will increase, thus accelerating convergence.

**2.8 Batch Normalization.** Batch Normalization is a way of batch standardization. It is similar to ordinary data standardization which is a way to unify scattered data. As we know that it's easier to learn the laws in the data with uniform specifications. Batch normalization divides the data into small batches in stochastic gradient descent step and when each batch of data is forwarded, normalization is processed for each layer.

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

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

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

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

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

*Figure 8.* Algorithm Batch Normalization

The first three steps are the normalization process and there is also a reverse operation after the formula, which expands and translates the normalized data. This step aims to let the neural network learn to use and modify this extension, the parameter gamma and the translation parameter . After that, neural network can slowly figure out whether the previous normalization operation has been optimized. If it does not work gamma and belt are used to offset some normalization operations.

## 3. Experiments and Discussions

### 3.1. Input

**Datasets.** The first dataset has 60000 samples and 10 classes, each sample has 128 features. We used 80% of them for training and the rest for testing.
The second dataset has 10000 sample and also 128 features. We predicted labels on the second dataset and then submitted them.

### 3.2 Measurement methods

**Accuracy** Accuracy formula is an important index of performance, which aims to represent the algorithm performance. The formula of accuracy metric is defined as:

$$Accuracy = \frac{number\ of\ correct\ classifications}{total\ number\ of\ test\ examples} \quad (10)$$

### 3.3 Experiments

In the experiments, we implemented the controlling variables to find the best parameters for the experiments. Comparing each experimental result, we need search for the best tuple with the best accuracy. The runtime is also needed to be considered. In experiment 1 to experiment 6, we used 3 hidden layers, first hidden layer has 80 units. The second hidden layer has 90 units and the last hidden layer has 70 units. For input layer, there are 128 units. Additionally, the output layer has 10 units. In experiment 7, we added more layers.

**Experiment1.** Comparing these two experimental results(shown on table 1), the only difference is in epoch. The accuracy is nearly same, but the Run time is much longer when epoch is equal to 200. Thus the volume of Epoch has effect on Run time. Small number of epoch may lead to underfitting.

**Experiment2.** Dropout is a method to deal with overfitting. These two sets of results show that the value of dropout has few effect on run time(table 2) based on our implementation. However, it influenced the accuracy. Not using dropout, in other words, the value of dropout or keep probability is equal to 1, resulted a lower accuracy score. Therefore, the value of dropout is an important factor for accuracy.

**Experiment3.** Batch size often determine the speed of converge and the direction of gradient. Shown on table 3, these three have different batch size. The accuracy is almost same. Run time is different, so the different may affect the run time.

**Experiment4.** In this set of experiments(table 4), one used batch normalization, another one did not. Batch normalization is used for improving performance and stability. Batch normalization needs to be set before activation function, so the higher learning rate is acceptable. Otherwise, there would be gradient vanishing or gradient exploding. We can see the accuracy was only 0.1413, when batch normalization was set to False, the solution was lowering the learning rate (e.g. learning rate=0.01, the accuracy became normal.) or use batch normalization.

**Experiment5.** Weight decay is used for solving overfitting. We compared two experimental sets(table 5). One has weight decay 0.001 and another one does not has weight decay. It was shown that the accuracy of the one with weight decay was 0.828, which was better than another one 0.8227 without weight decay.

**Experiment6.** Momentum is a parameter for accelerating the function convergence. Comparing the followed two loss function table, if we have momentum as a parameter and set it with 0.9(Figure 15), it is shown that the function begins converging at epoch 5. However, if we do not use momentum(Figure 16), the function is hard to converge.

**Experiment7.** Finally, we set the last set of experiment to compare difference in different units. In this set of experiment, 3 more hidden layers are added. Each layer has 50 units. The 6 hidden layers could lower the accuracy (0.8287), compared with 3 hidden layers (0.8825) and time is longer (143.23) against(93.04). Other modules are same. So, we have a conclusion: the more hidden layers, the Neural Network becomes more complicate, which has influence on performance.

### 3.4 Experiment Analysis

we implemented controlling variables methods for comparison with each experimental result. The best result was the first one in table 3, which was 89%. According to the experiments, the best dropout is 0.7 with better accuracy than dropout

| learning rate | Epoch | dropout | weight decay | batch size | momentum | Accuracy | Time(s) |
|---|---|---|---|---|---|---|---|
| 0.1 | 200 | 0.7 | 1e-5 | 200 | 0.9 | 0.8869 | 143.57 |
| 0.1 | 100 | 0.7 | 1e-5 | 200 | 0.9 | 0.8807 | 66.51 |

*Figure 9*. table 1

| learning rate | Epoch | dropout | weight decay | batch size | momentum | Accuracy | Time(s) |
|---|---|---|---|---|---|---|---|
| 0.1 | 100 | 1 | 1e-5 | 100 | 0.9 | 0.8117 | 91.22 |
| 0.1 | 100 | 0.5 | 1e-5 | 100 | 0.9 | 0.873 | 93.62 |

*Figure 10*. table 2

| learning rate | Epoch | dropout | weight decay | batch size | momentum | Accuracy | Time(s) |
|---|---|---|---|---|---|---|---|
| 0.1 | 100 | 0.7 | 1e-5 | 100 | 0.9 | 0.89 | 92.04 |
| 0.1 | 100 | 0.7 | 1e-5 | 60 | 0.9 | 0.8879 | 121.33 |
| 0.1 | 100 | 0.7 | 1e-5 | 30 | 0.9 | 0.8866 | 213.11 |

*Figure 11*. table 3

| learning rate | Epoch | weight decay | momentum | batch norm | Accuracy | Time(s) |
|---|---|---|---|---|---|---|
| 0.1 | 100 | 1e-5 | 0.9 | False | 0.1413 | 49.60 |
| 0.01 | 100 | 1e-5 | 0.9 | False | 0.8877 | 43.92 |
| 0.1 | 100 | 1e-5 | 0.9 | True | 0.89 | 92.04 |

*Figure 12*. table 4

| learning rate | Epoch | weight decay | batch size | Accuracy | Time(s) |
|---|---|---|---|---|---|
| 0.001 | 50 | 0.001 | 100 | 0.828 | 30.26 |
| 0.001 | 50 | 0 | 100 | 0.8227 | 28.499 |

*Figure 13*. table 5

| learning rate | Epoch | Batch size | momentum | dropout | Accuracy | Time(s) |
|---|---|---|---|---|---|---|
| 0.001 | 50 | 256 | 0.9 | 1 | 0.863 | 17.38 |
| 0.001 | 50 | 256 | 0 | 1 | 0.7409 | 17.34 |

*Figure 14*. table 6

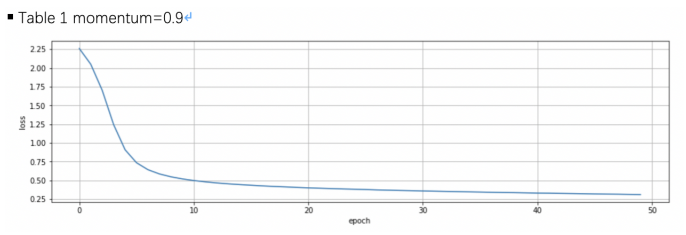

*Figure 15*. momentum=0.9

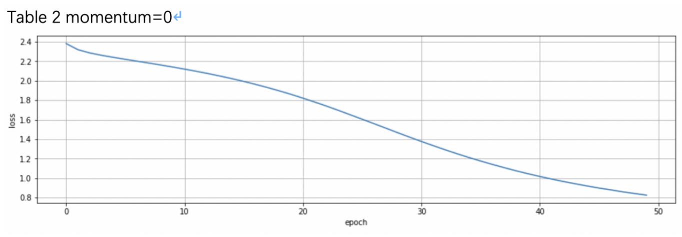

*Figure 16*. momentum=0

is 1. Different batch size could have different effect on accuracy and runtime. If batch size is too small (e.g. less than 60). There will be some concussions in the process during the epoch=25 and epoch=50. Additionally, the runtime is longer than batch size=100. We do not need to set the batch size too large, because the proper batch size could converge well (at the epoch=5, the loss function begins restraining.). Besides, large batch size brings high pressure on memory and the gradient direction does not change. The momentum is used for accelerating the restraining process. The weight decay could deal with overfitting.

## 4.Personal Reflection

**What we learned.** In this assignment, we learned how neural network work and how to implement a MLP classifier. We then learned different methods and their implementations to improve neural network, which were ReLU activation, Weight decay, Momentum in SGD, Dropout, Softmax and cross-entropy, Mini-batch training, Batch Normalization and the meaning of these methods.Through using of these methods, the chance of overfitting has been decreaseed obviously.
**Shortcomings.** There were too many options of hyper-parameter combinations, it is difficult to find the optimised hyper-parameters, which lead to a higher accuracy score. Besides, the combination and relation among these optimized methods are also needed to be explored.

## 5. Conclusion

In this assignment, we implemented MLP classifier and other methods to improve our classifier, they were ReLU activation, Weight decay, Momentum in SGD, Dropout, Softmax and cross-entropy, Mini-batch training, Batch Normalization. We divided our given dataset to training data and testing data. We found that MLP classifier was a powerful classifier in this assignment. The highest accuracy we got was around 89%. Increase number of layers and units lead to more complex model. Weight decay and dropout would adjust complexity of MLP model. Momentum could accelerate SGD in the relevant direction and dampens oscillations. Mini-batch training is a compromise method to overcome the shortcomings of Batch gradient descent and stochastic gradient descent. Batch normalization is a way of batch standardization, it allows larger learning rate, reduces covariate shift and effects of exploding and vanishing gradients.

## References

https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

https://jamesmccaffrey.wordpress.com/2013/11/05/why-you-should-use-cross-entropy-error-instead-of-classification-error-or-mean-squared-error-for-neural-network-classifier-training/

https://arxiv.org/pdf/1811.03378.pdf

https://towardsdatascience.com/softmax-function-simplified-714068bf8156

[1] Bishop, C. M. (2006). Pattern recognition and machine learning. New York: Springer

[2] Cheng, W., Hüllermeier, E. (2009). Combining instance-based learning and logistic regression for multilabel classification. Machine Learning, 76(2-3), 211-225. doi:10.1007/s10994-009-5127-5

## Appendix

### 1.Hardware and Software.

- Processor: 2.6 GHz Intel Core i7
- Memory: 16 GB 2400 MHz DDR4
- Coding tool: jupyter notebook
- Interpreter: Python 3.6.4
- Libraries: h5py, numpy, matplotlib, time

### 2.Data Set.

- train_128.h5
- train_label.h5
- test_128.h5

**3.Code Instruction.** Code is written in Python 3.6.
Use following command to run the code in terminal

```
jupyter notebook MLP.ipynb
```

To load the unlabeled test data result, we use following command:

```
with h5py.File('../Output/pPredicted_labels.h5','r') as H:
    test_result = np.copy(H['label'])
```

**4.contribution of group members.** The code were written in team work as each person in charge of particular parts of code, also as the work of data collection and analysis. After all coding work finished, we wrote the report together through using Overleaf. All in a word, each group member did the same contribution.