NVLab Summer School 2023 HW2

112061517 王浩

Abstract

在本次作業裡，我們利用numpy 手刻一組Feed-forward Neural Network，並嘗試調整hyper-parameter 調整highest score、更動activation function 討論其應用、利用train and test score 確認是否overfitting，研究相關議題。

1 Code Implementation

1.1 Fashion-MNIST dataset

```
def load_data():
    path_list = [url_train_image, url_train_labels
                                , url_test_image, url_test_labels]
    with gzip.open(path_list[0], 'rb') as f:
        x_train = np.frombuffer(f.read(), np.uint8, offset=16)
                                            .reshape(-1, 28*28)/255.0
    with gzip.open(path_list[1], 'rb') as f:
        y_train = np.frombuffer(f.read(), np.uint8, offset=8)
    with gzip.open(path_list[2], 'rb') as f:
        x_test = np.frombuffer(f.read(), np.uint8, offset=16)
                                            .reshape(-1, 28*28)/255.0
    with gzip.open(path_list[3], 'rb') as f:
        y_test = np.frombuffer(f.read(), np.uint8, offset=8)

    return (x_train, y_train), (x_test, y_test)
```

1.2 Design a two-layers FCNN model (1 hidden layer + 1 output layer)

Every output neuron has full connection to the input neurons

```
class FFNN:
    def forward(self, X):
        # Forward pass through the network

        ## Note: np.dot(X, self.W1) size is (N, hidden_size1),
        ## self.b1 size is (hidden_size1)
        ## so self.b1 will be broadcasted to the same shape as np.dot(X, self.W1)
        ## self.z1 size is (N, hidden_size1)

        self.z1 = np.dot(X, self.W1) + self.b1
        self.a1 = self.relu(self.z1)

        self.z2 = np.dot(self.a1, self.W2) + self.b2
        self.a2 = self.relu(self.z2)

        self.z3 = np.dot(self.a2, self.W3) + self.b3
        self.a3 = self.softmax(self.z3)
        return self.a3
```

1.3 ReLU layer

We add nonlinear activation functions after the neural layer using ReLU.

```
def relu(self, x):
    ## Note: 0 will be broadcasted to the same shape as x is (N, hidden_size)
    ## np.maximun is element-wise to compare which one is bigger
    ## reture size is the same as x
    return np.maximum(0, x)
```

1.4 Softmax output

The final layer is typically a Softmax function which outputs the probability of a sample being in different classes.

```
def softmax(self, x):
    ## Note1:
    ## x size is (N, output_size)
    ## we need to sum over the second dimension, so we set axis=1, size is (N, 1)
    ## and np.exp(x) size is (N, output_size)
    ## so np.exp(x) / np.sum(np.exp(x), axis=1, keepdims=True)
    ## size is (N, output_size)
    ## division here is broadcasted over the first dimension too
    ## Note2:
    ## subtracting the maximum value along the axis
    ## is to prevent overflow when exponentiating large values
    exp_x = np.exp(x - np.max(x, axis=1, keepdims=True))
    return exp_x / np.sum(exp_x, axis=1, keepdims=True)
```

1.5 Cross-entropy loss calculation

Use the output of a batch of data and their labels to calculate the CE loss.

```
def cross_entropy_loss(self, y_ture, y_pred):
    ## Note 1:
    ## dividing by y_true.shape[0] (the number of samples in the batch)
    ## is a normalization step to ensure that
    ## the loss is independent of the batch size.
    ## The loss function measures the average loss per sample in the batch.
    ## Note 2:
    ## original y size is (N, 1), y_pred size is (N, output_size),
    ## we need to change y to one-hot encoding.
    ## Note 3:
    ## np.sum(y_ture * np.log(y_pred)) at the beginning is nan
    ## because y_pred is nearly 0 at the beginning, and log(0) is not defined,
    ## so it is nan
    ## we need to add a small number to y_pred to avoid this problem
    epsilon = 1e-8
    return -np.sum(y_ture * np.log(y_pred + epsilon)) / y_ture.shape[0]
```

1.6 Backward propagation

Propagate the error backward and update each parameter.

```
def relu_derivative(self, x):
    return np.where(x > 0, 1, 0)

def backward(self, X, y_ture, lr=0.01):
    m = X.shape[0]

    # Output layer gradients
    ## Note: dL_dz3 is calculated by the derivative of cross entropy and softmax
    ## Note self.a3 size is (N, output_size), y_ture size is (N, output_size),
    ## dL_dz3 size is (N, output_size)
    dL_dz3 = (self.a3 - y_ture) / m
    ## Note: dL_dW3 is multiplied by backward and forward propagation
    ## self.a2.T size is (hidden_size2, N), dL_dz3 size is (N, output_size),
    ## dL_dW3 size is (hidden_size2, output_size)
    dL_dW3 = np.dot(self.a2.T, dL_dz3)
    dL_db3 = np.sum(dL_dz3, axis=0)

    # Hidden layer 2 gradients
    ## self.W3.T size is (output_size, hidden_size2), dL_dz3 size is
    ## (N, output_size), dL_dz2 and self.z2 size is (N, hidden_size2)
    dL_dz2 = np.dot(dL_dz3, self.W3.T) * self.relu_derivative(self.z2) / m
    # self.a1.T size is (hidden_size1, N),  dL_dz2 size is (N, hidden_size2)
    dL_dW2 = np.dot(self.a1.T, dL_dz2)
    dL_db2 = np.sum(dL_dz2, axis=0)

    # Hidden layer 1 gradients
    dL_dz1 = np.dot(dL_dz2, self.W2.T) * self.relu_derivative(self.z1) / m
    dL_dW1 = np.dot(X.T, dL_dz1)
    dL_db1 = np.sum(dL_dz1, axis=0)

    # Update weights and biases
    self.W3 -= lr * dL_dW3
    self.b3 -= lr * dL_db3
    self.W2 -= lr * dL_dW2
    self.b2 -= lr * dL_db2
    self.W1 -= lr * dL_dW1
    self.b1 -= lr * dL_db1
```

我們本次的Backward propagation 順序可以分成兩個部分分析，分別是最後一層的Layer，與其他層內容。

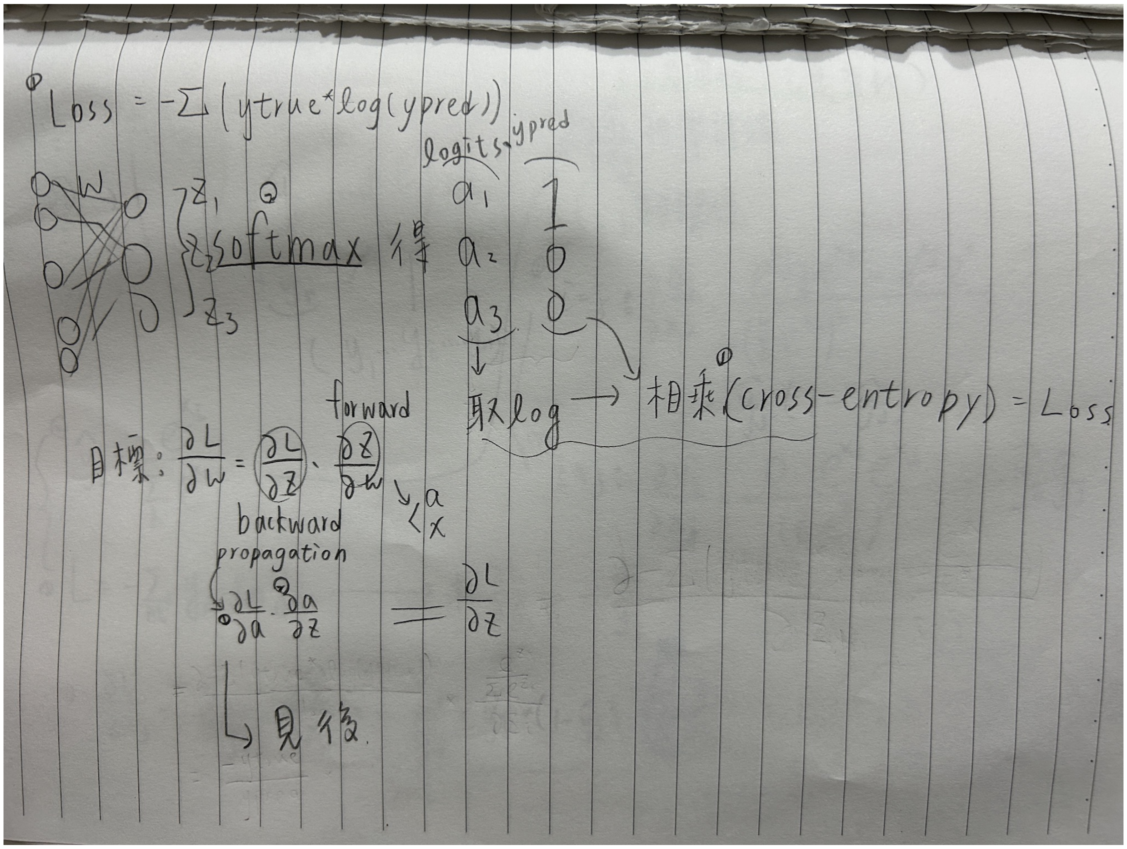
- Last layer：如圖1所示，這是我們最後一層的內容，我們經過Softax 之後，取log，並且利用 cross-entropy 計算出Loss value。而Partial derivative 的運算，可以見圖2，最後得出$\hat{y} - y$ 這樣的結果。
- Other layers：可以經過運算後，得出3的結果，我們就利用這個公式來實作Gradient descent。

2 Discussion

2.1 Model Architecture and score

Show the model architecture, implementation detail, and testing accuracy. Please describe the methods to achieve the final result in detail, e.g., epochs, batch size, etc.

我本次所用的模型是FFNN(Feedforward Neural Network) picture 4, table 1 一開始的input layer 是根據Dataset Fashion_MNIST 的大小$28 * 28$，所以input size 為784，接上的hidden layer 1 size 是$784/2 = 392$，hidden layer 2 size 是128，最後一層的output size 是根據class 數量，故設定為10，



① $\text{Loss} = -\sum(\text{ytrue} * \log(\text{ypred}))$

W
z_1
z_2
z_3
② softmax 得
logits
ypred
a_1
a_2
a_3
1
0
0

a_3 ↓ 取log → ① 相乘(cross-entropy) = Loss

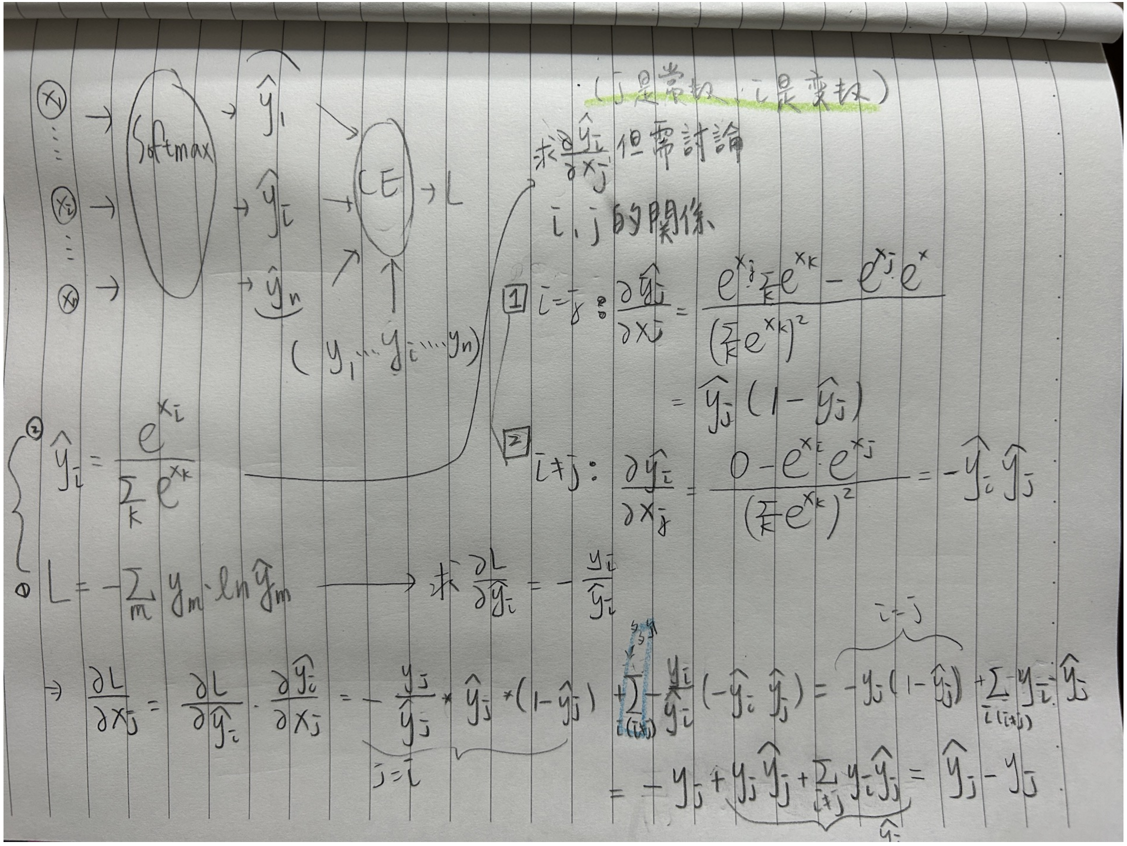
目標：$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial z} \cdot \frac{\partial z}{\partial w}$ forward ↘ $<\begin{matrix} a \\ x \end{matrix}$

backward propagation

① $\frac{\partial L}{\partial a} \cdot$ ② $\frac{\partial a}{\partial z} = \frac{\partial L}{\partial z}$

↳ 見後.

Figure 1: This is the last layer of FFNN.



x_1
x_i
x_n
Softmax
$\hat{y}_1$
$\hat{y}_i$
$\hat{y}_n$
CE
L
$(y_1 \cdots y_i \cdots y_n)$

(j是常數，i是變數)

求 $\frac{\partial \hat{y}_i}{\partial x_j}$ 但需討論 i, j 的關係

$\boxed{1}$ $i=j$：$\frac{\partial \hat{y}_i}{\partial x_j} = \frac{e^{x_j} \sum_k e^{x_k} - e^{x_j} \cdot e^{x}}{(\sum_k e^{x_k})^2} = \hat{y}_j(1-\hat{y}_j)$

$\boxed{2}$ $i \neq j$：$\frac{\partial \hat{y}_i}{\partial x_j} = \frac{0 - e^{x_i} e^{x_j}}{(\sum_k e^{x_k})^2} = -\hat{y}_i \hat{y}_j$

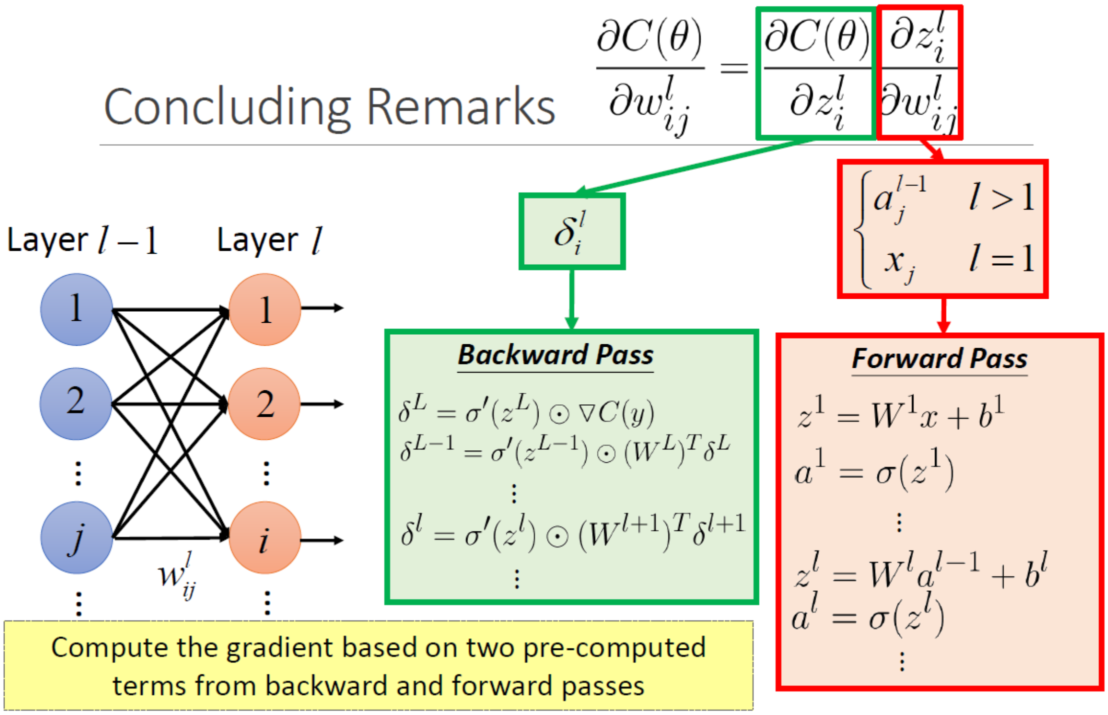
② $\hat{y}_i = \frac{e^{x_i}}{\sum_k e^{x_k}}$

① $L = -\sum_m y_m \ln \hat{y}_m \longrightarrow$ 求 $\frac{\partial L}{\partial \hat{y}_i} = -\frac{y_i}{\hat{y}_i}$

$$\Rightarrow \frac{\partial L}{\partial x_j} = \frac{\partial L}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial x_j} = \underbrace{-\frac{y_j}{\hat{y}_j} * \hat{y}_j * (1-\hat{y}_j)}_{j=i} + \sum_{i(\neq j)} -\frac{y_i}{\hat{y}_i}(-\hat{y}_i \hat{y}_j) = \overbrace{-y_j(1-\hat{y}_j)}^{i=j} + \sum_{i(\neq j)} y_i \hat{y}_j$$

$$= -y_j + \underbrace{y_j \hat{y}_j + \sum_{i \neq j} y_i \hat{y}_j}_{\hat{y}_j} = \hat{y}_j - y_j$$

Figure 2: Derivative of cross-entropy and softmax



Concluding Remarks

$$\frac{\partial C(\theta)}{\partial w_{ij}^l} = \boxed{\frac{\partial C(\theta)}{\partial z_i^l}}\boxed{\frac{\partial z_i^l}{\partial w_{ij}^l}}$$

$$\boxed{\delta_i^l}$$

$$\boxed{\begin{cases} a_j^{l-1} & l > 1 \\ x_j & l = 1 \end{cases}}$$

Backward Pass

$$\delta^L = \sigma'(z^L) \odot \nabla C(y)$$
$$\delta^{L-1} = \sigma'(z^{L-1}) \odot (W^L)^T \delta^L$$
$$\vdots$$
$$\delta^l = \sigma'(z^l) \odot (W^{l+1})^T \delta^{l+1}$$
$$\vdots$$

Forward Pass

$$z^1 = W^1 x + b^1$$
$$a^1 = \sigma(z^1)$$
$$\vdots$$
$$z^l = W^l a^{l-1} + b^l$$
$$a^l = \sigma(z^l)$$
$$\vdots$$

Compute the gradient based on two pre-computed terms from backward and forward passes

Figure 3: 其他layer 的運算[Che16]

	Value
Batch size	100
Epoch	1000
Learning rate	0.1
Precision	0.8732

Table 1: The parameter setting of model

每一層都是利用Weighted Matrix, bias 做線性轉換，並在前兩層做*RELU* activation function，來轉換空間，最後一層做softmax normalization，最後利用cross-entropy 來計算Loss，並陸續做gradient descent 更新參數。

而根據結論，最高分的設計為Batch size 為100, Epoch 為1000, Learning rate 為0.1 最後test dataset precision 為0.8732, training dataset precision 為0.97 如table 1。

2.2 Overfit

During testing, the model might overfit training data to achieve poor performance. How do you check overfitting and underfitting during training? Please describe the methods and experiments to prove that.

Check overfit 的方式為觀察training data 和testing data 的結果優劣，如果training data 的loss 持續下降且precision 也是持續的上升，那這樣的過程也就是gradient descent 去修改模型參數，但如果

	Value
Input size	784
Hidden layer 1 size	392
Hidden layer 2 size	128
Output size	10

Table 2: The layers' size

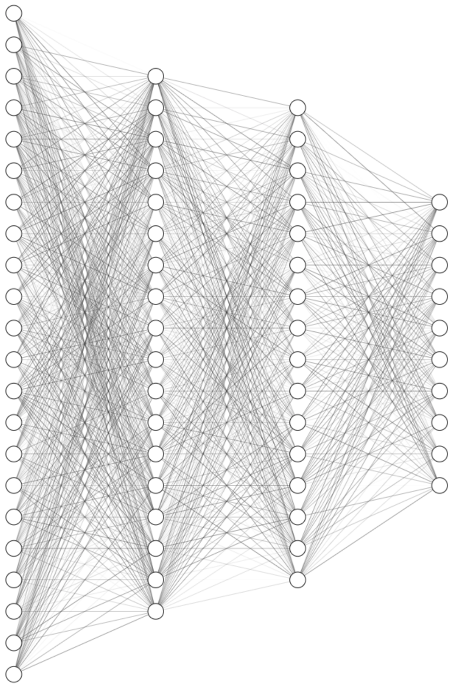


Figure 4: Feedforward Neural Network

此時的testing data 的loss 反而上升或accuracy 下降，那這個模型就是「過度符合」training data 的數值特性，這樣就是overfitting。

但在實驗overfit 時，我有幾組的參數，把epoch 調整到比較不符合常理的1000 如figure 5 或3000 如figure 9 ，但epoch 為1000 時，可以從圖形上稍許看到loss 有些微上升，就可以視為一點點overfit 或是即將overfit。

2.3 Hyperparameters

During training, you need to adjust the hyperparameters (epochs, batch size, ...) to achieve higher accuracy. How do you choose these hyperparameters? Please conduct experiments to show that these hyperparameters are suitable to achieve higher accuracy for the final test results. (Please use table to summarize your results)

我調整的hyperparameters 為epochs 與batch size 分別有同的組合，batch 有100, 1000，而epoch 有50, 300, 1000，而以下是各自的分數與細節。而如果batch size 較小的話，我們就會做比較多次iteration，所以參數的更新可以比較快，但結果可能較為不穩定，我們還需要根據本身的data size 與性質來做調整。而epoch 如果較小，那更新完的分數可能就不會很高，因為訓練的總次數不多，但如果太大的話，可能就會造成overfitting. 可見table 3.

Table 3: Models with different parameter sets

batch size	learning rate	activation function	epoch	train accuracy	test accuracy	fig #
100	0.1	RELU	50	0.880	0.8393	6
			300	0.910	0.8690	10
			1000	**0.970**	**0.8732**	5
			3000	0.900	0.8512	9
		Sigmoid	300	0.880	0.8210	11
1000	0.1	RELU	50	0.746	0.7446	8
			300	0.815	0.7986	7

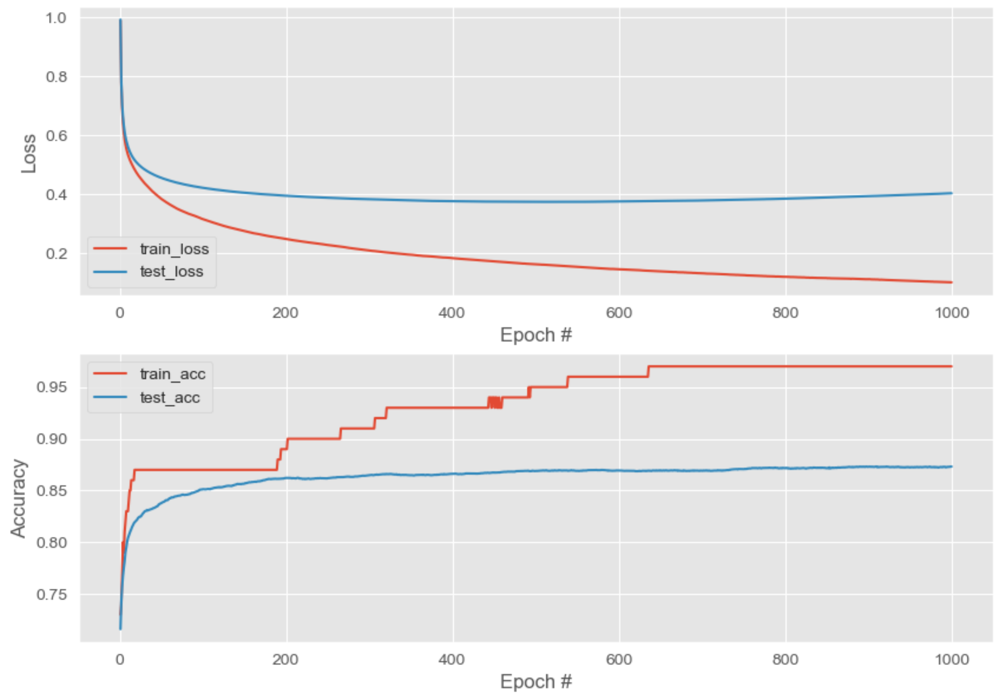


Figure 5: RELU, batch:100, epoch:300

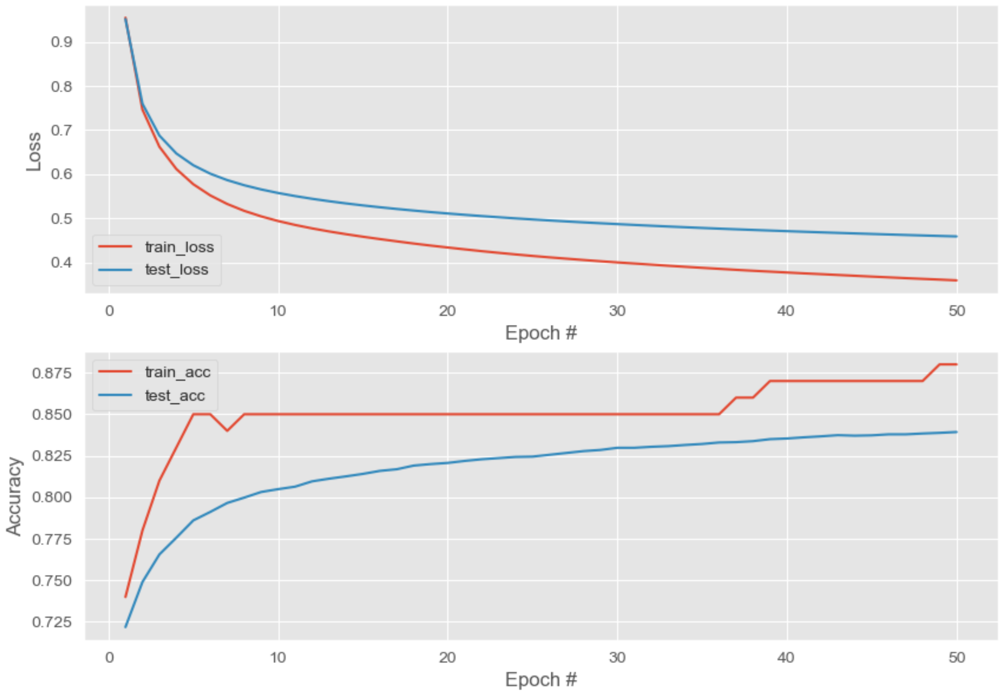


Figure 6: RELU, batch:100, epoch:50

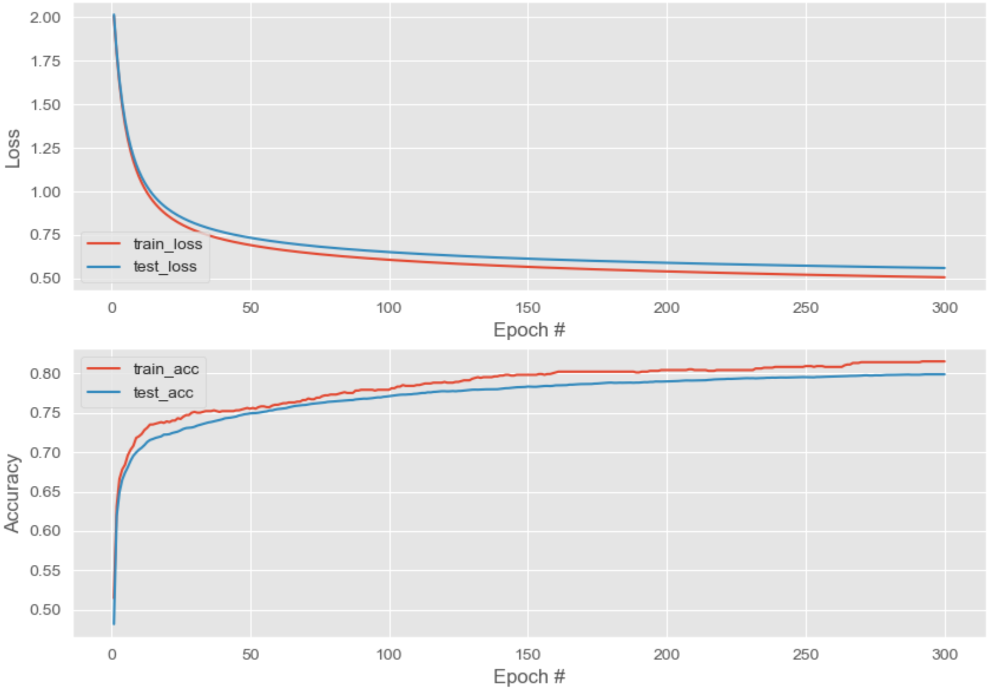


Figure 7: RELU, batch:1000, epoch:300

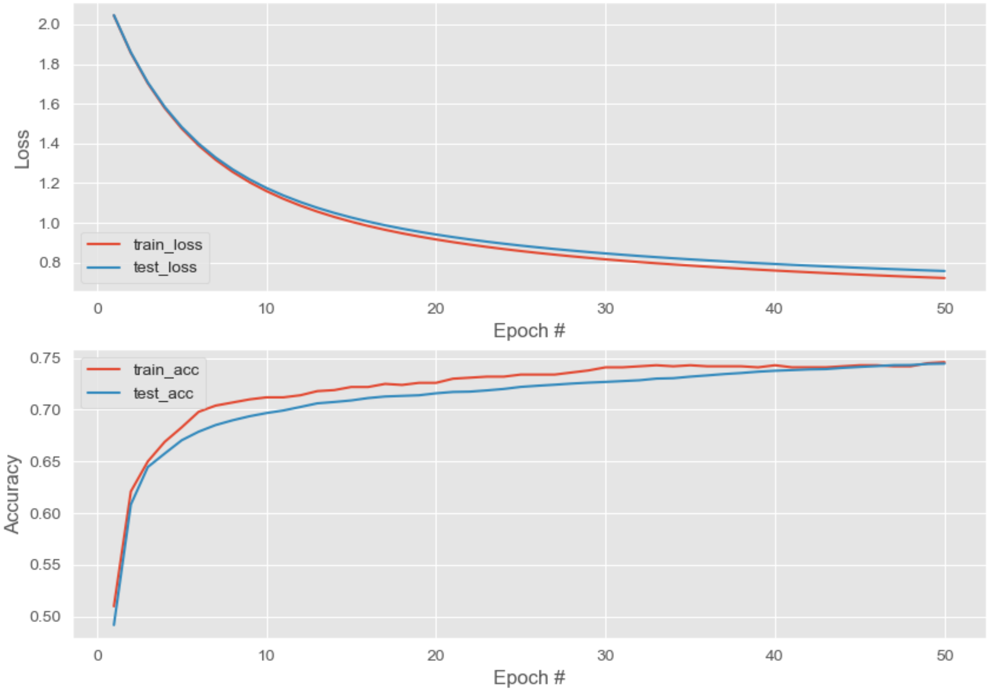


Figure 8: RELU, batch:100, epoch:300

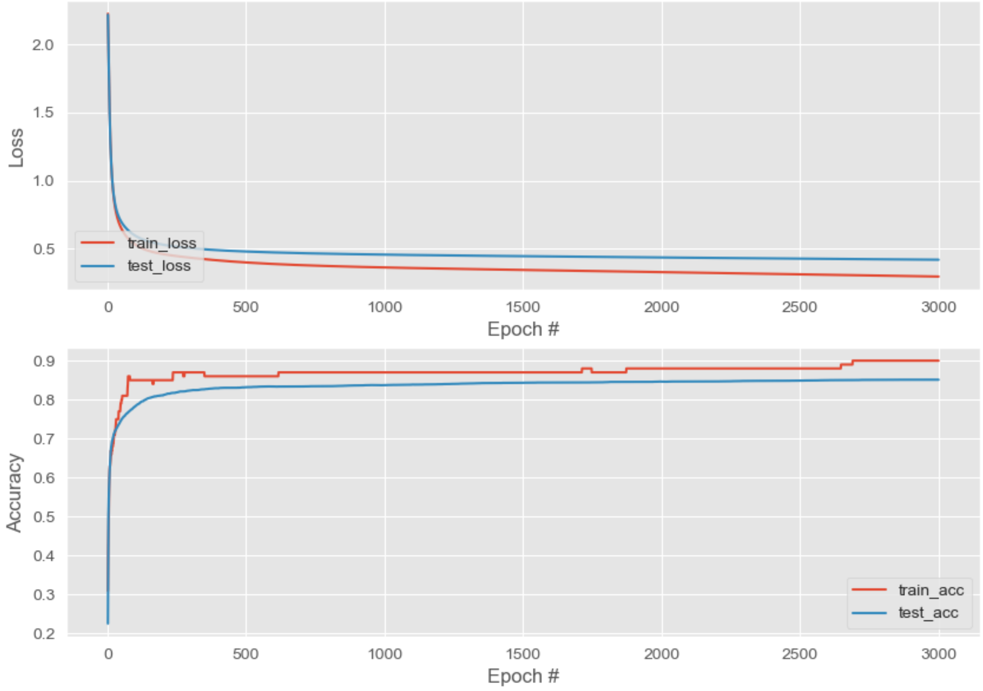


Figure 9: RELU, batch:100, epoch:3000

2.4 Activation Function

Please do experiments to compare at least other one activation function in the model. Analyze the activation function you used and show if these activation functions help improve the performance during testing.

2.4.1 RELU

$$f(x) = max(0, x) \tag{1}$$

我們原先使用的activation function 為Relu equation 1，在大於0 時，能夠維持原先的值，而在小於0時，直接將值變換成0，在運算上相當快速，不管是forward 或是backward 時，都不需要複雜的運算。但缺點是直接忽略掉負的狀況，也就是並不在乎負的情況來接續影響layer output，而是將其視為0。

```
def relu(self, x):
    ## Note: 0 will be broadcasted to the same shape as x is (N, hidden_size)
    ## np.maximun is element-wise to compare which one is bigger
    ## reture size is the same as x
    return np.maximum(0, x)

def relu_derivative(self, x):
    return np.where(x > 0, 1, 0)
```

2.4.2 Sigmoid

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

我採用的另外一個是sigmoid equation 2 梯度較為平穩。因為輸出介於0 1，因此不管怎麼樣的輸入，都不會造成blow up。缺點則是當輸入的絕對值很大時，輸入的輸出的影響會非常小。所以具有梯度消失的問題。更新速度也會比較慢。

```
def sigmoid(self, x):
    return 1 / (1 + np.exp(-x))

def sigmoid_derivative(self, x):
    return self.sigmoid(x) * (1 - self.sigmoid(x))
```

2.5 Comparison

- 根據上方的內容，我們可以從10、11 兩張圖看出，Loss 可以從y 軸數據的基準看出差異，此外在下降的波形上，也能觀察出RELU 的下降速度較快。
- 而在Accuracy 上，可以看出一開始RELU 約只需要5個epochs 即可達到0.75 的成績；但同時Sigmoid 需要約50 個epochs 才能達到，速度慢下許多。

References

[Che16] Yun-Nung Chen. Cs5431 - applied deep learning, backpropagation slide, 2016.

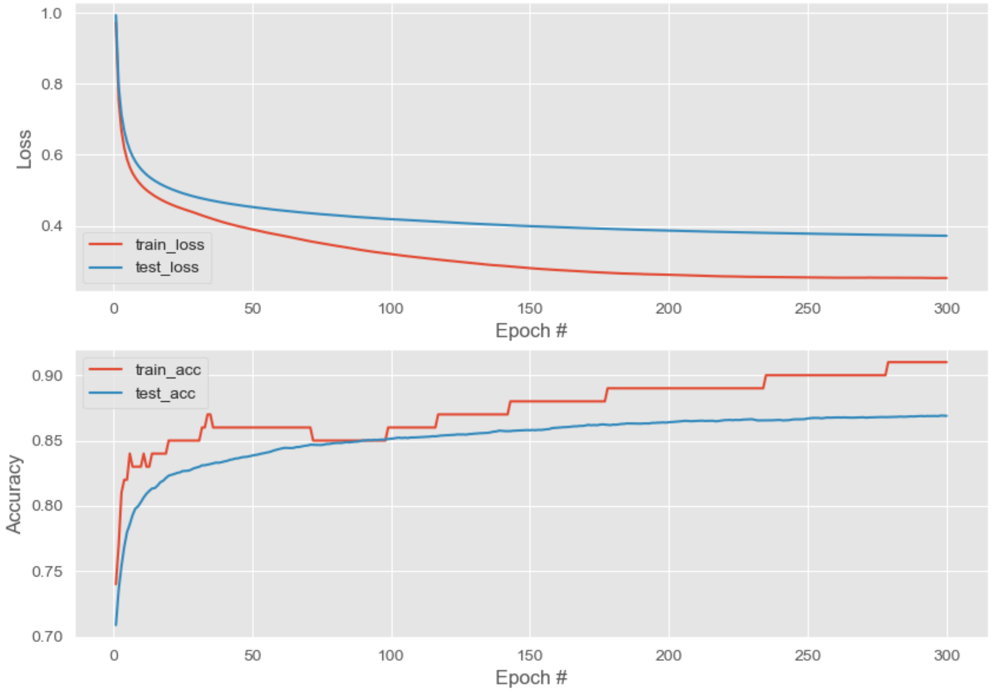


Figure 10: RELU, batch:100, epoch:300

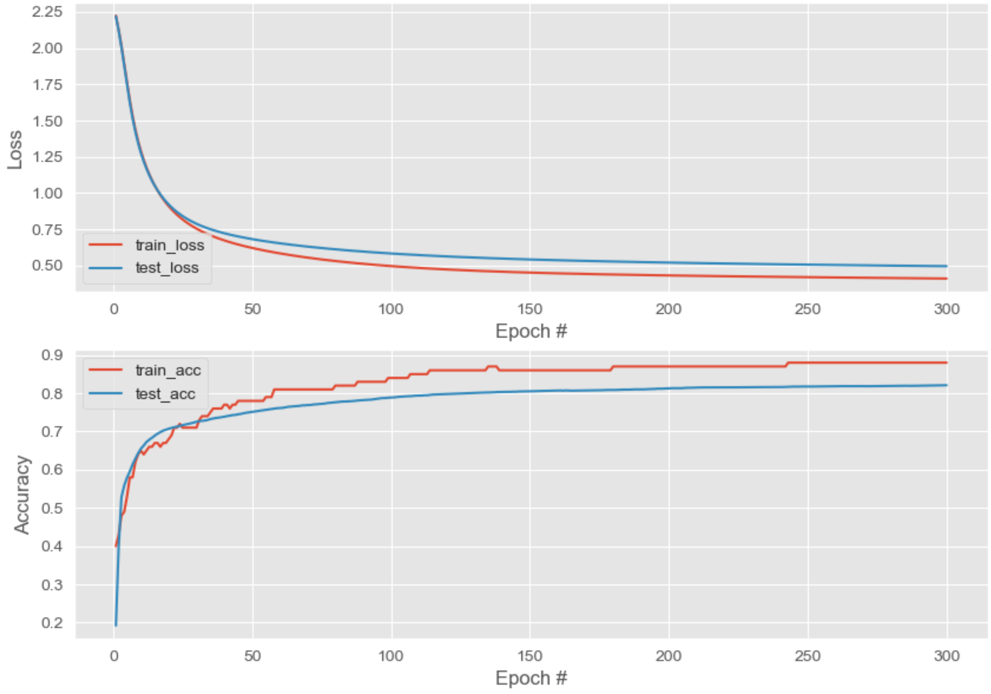


Figure 11: Sigmoid, batch:100, epoch:300