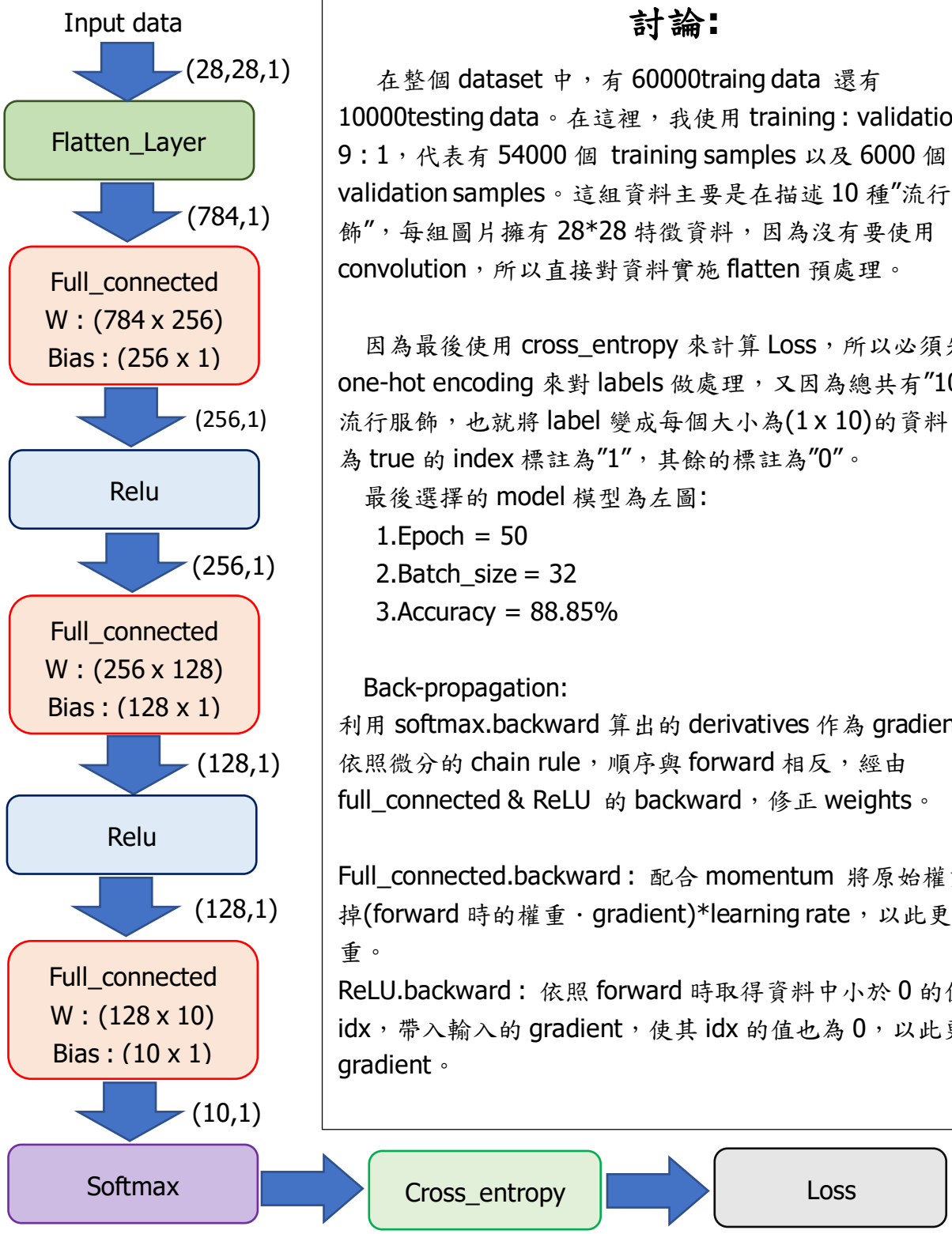
HW1 neural network reports

電機系碩士班 111061562 林淞祐

1. model architecture



討論:

在整個 dataset 中，有 60000traing data 還有 10000testing data。在這裡，我使用 training : validation 為 9 : 1，代表有 54000 個 training samples 以及 6000 個 validation samples。這組資料主要是在描述 10 種"流行服飾"，每組圖片擁有 28*28 特徵資料，因為沒有要使用 convolution，所以直接對資料實施 flatten 預處理。

因為最後使用 cross_entropy 來計算 Loss，所以必須先用 one-hot encoding 來對 labels 做處理，又因為總共有"10"種流行服飾，也就將 label 變成每個大小為(1 x 10)的資料，在為 true 的 index 標註為"1"，其餘的標註為"0"。

最後選擇的 model 模型為左圖:

1.Epoch = 50
2.Batch_size = 32
3.Accuracy = 88.85%

Back-propagation:

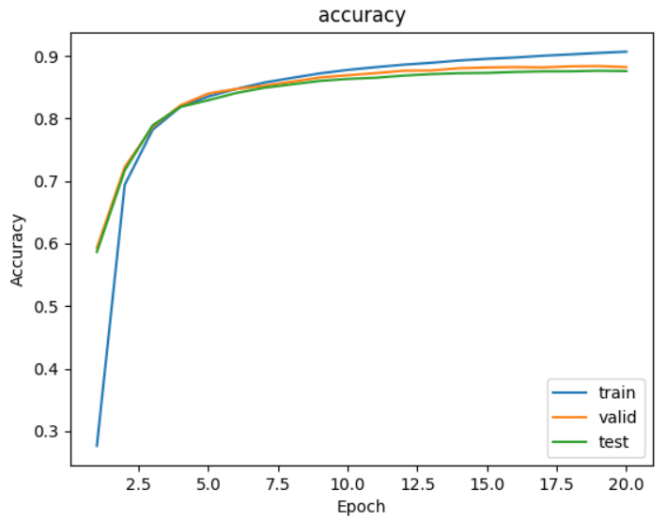
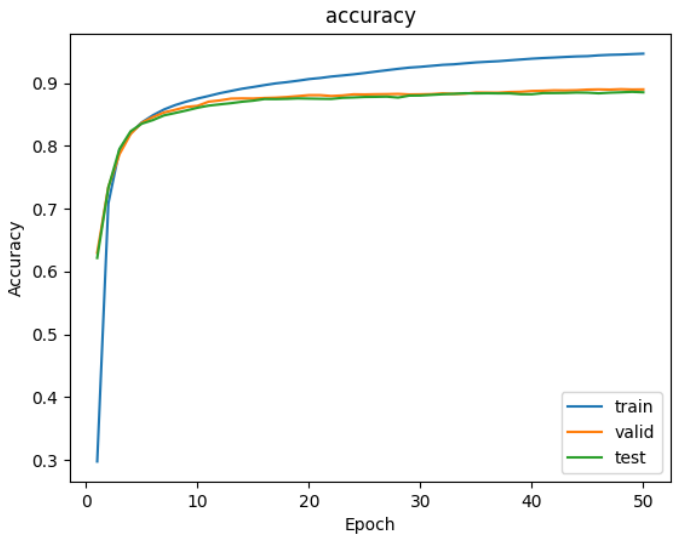
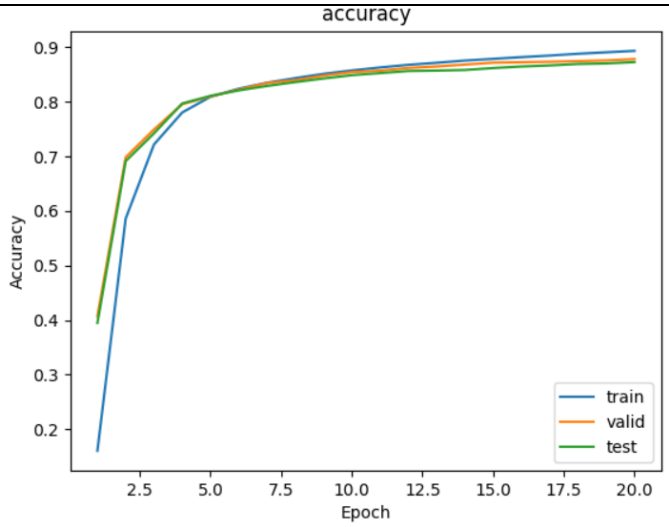
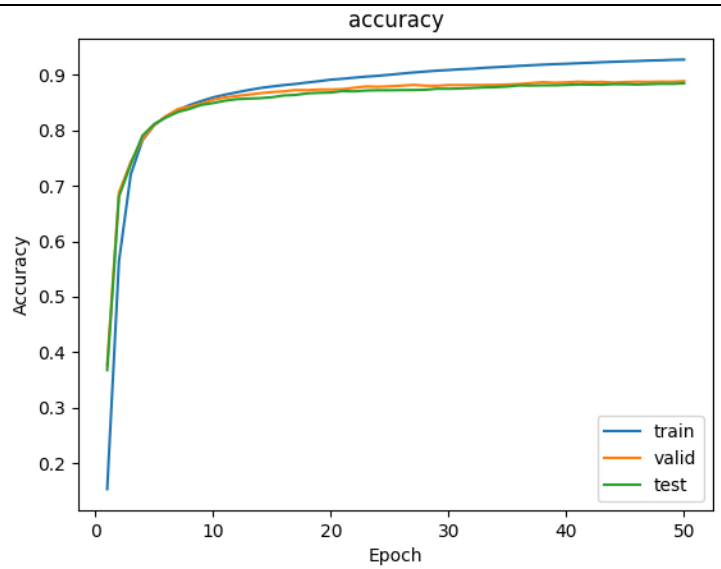
利用 softmax.backward 算出的 derivatives 作為 gradient，依照微分的 chain rule，順序與 forward 相反，經由 full_connected & ReLU 的 backward，修正 weights。

Full_connected.backward : 配合 momentum 將原始權重減掉(forward 時的權重 · gradient)*learning rate，以此更新權重。

ReLU.backward : 依照 forward 時取得資料中小於 0 的值的 idx，帶入輸入的 gradient，使其 idx 的值也為 0，以此更新 gradient。

2.3. overfitting discusion & hyperparameters comparison

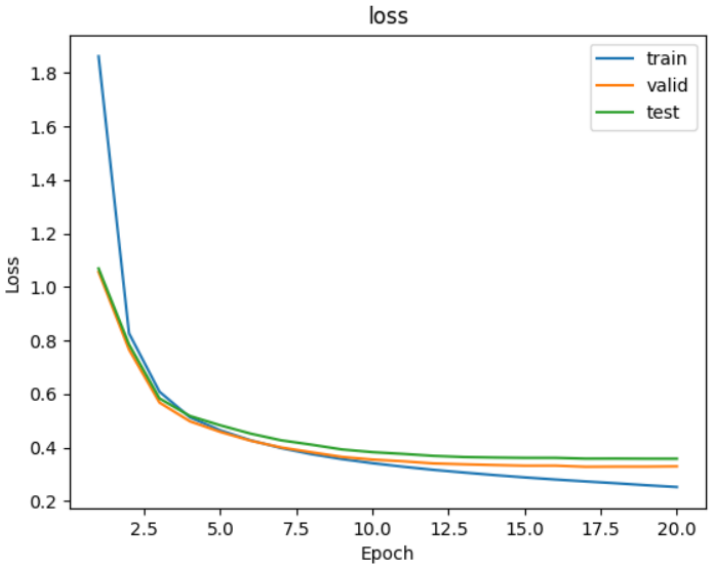
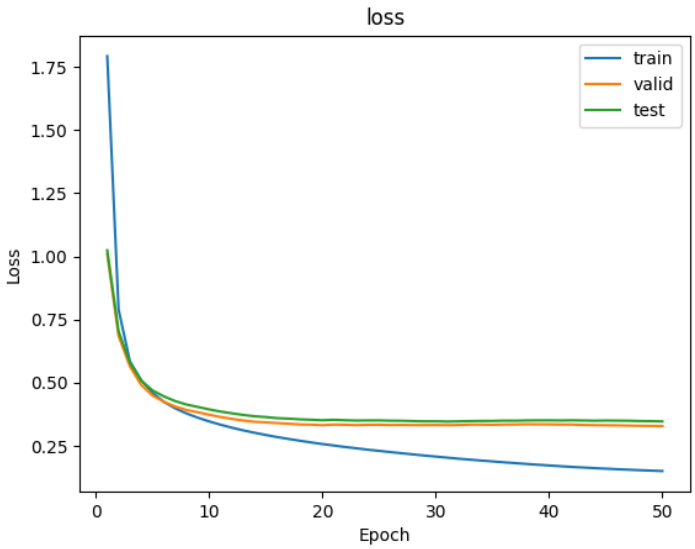
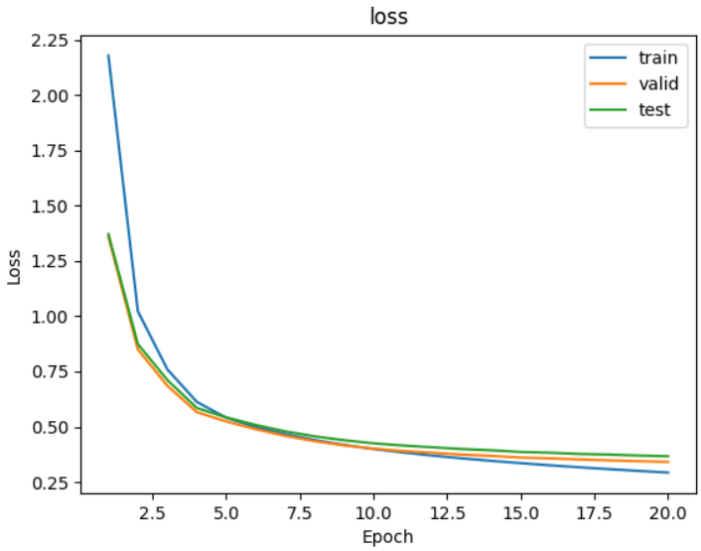
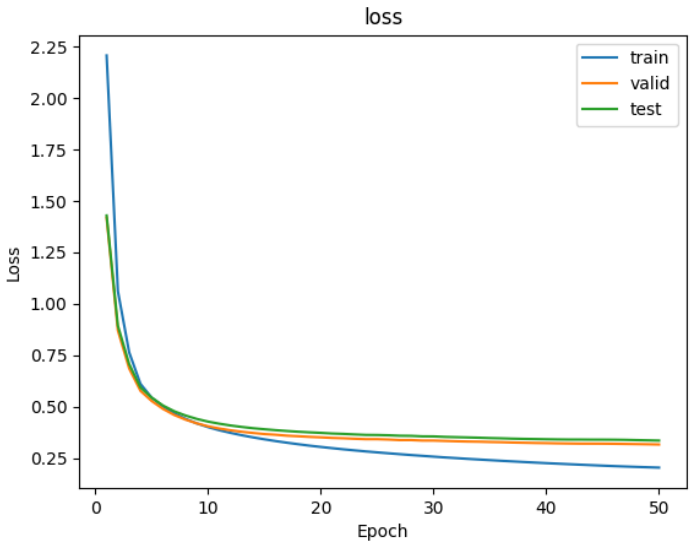
Accuracy

	Epoch = 20	Epoch = 50
Batch_size = 32		
Batch_size = 50		

討論:

由上圖，可以發現與 epoch=20 比較，在 epoch=50 的時候，accuracy 和 Loss 已經接近收斂完成的趨勢(88.8%的準確率)，繼續的增加迭代數並不會對 accuracy(testing)造成影響，反而在 validation samples 的 epoch=20 開始，模型開始出現 overfitting 的現象(traing_acc – test_acc > 3%)，因此將 learning_rate 在第 20 次迭代之後開始利用線性遞減做調整，計算每次遞減大小所用的公式為 ：distance = (LR – LR/10) / (epoch - 20)。同時可以觀察到 validation 的準確率收斂程度可以反應 testing 的收斂程度，這點可以透過上圖來得知。由 Batch_size 的比較可以看出，能夠影響 Loss 最後的收斂空間能夠繼續改善。所以最後的選擇的參數為 ：epoch = 50 ; batch_size = 32。

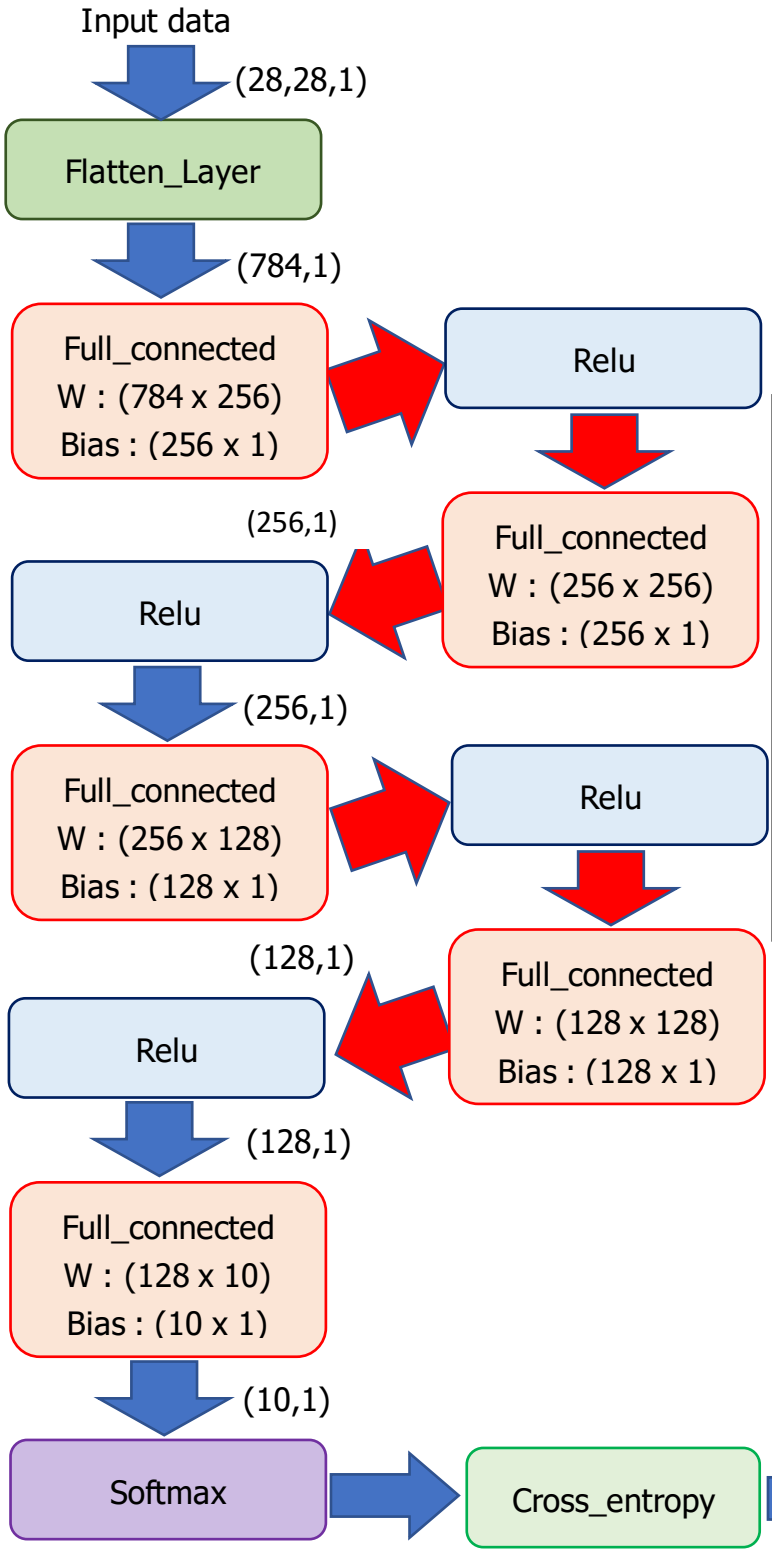
Loss

	Epoch = 20	Epoch = 50
Batch_size = 32		
Batch_size = 50		


```
Epoch 45 | Train acc. 0.9524 | Train loss 0.1368 | Val acc. 0.8879 | Val loss 0.3442 | test_acc 0.8866 | test loss 0.3628
Epoch 46 | Train acc. 0.9533 | Train loss 0.1344 | Val acc. 0.8882 | Val loss 0.3420 | test_acc 0.8873 | test loss 0.3605
Epoch 47 | Train acc. 0.9537 | Train loss 0.1320 | Val acc. 0.8885 | Val loss 0.3419 | test_acc 0.8869 | test loss 0.3601
Epoch 48 | Train acc. 0.9549 | Train loss 0.1299 | Val acc. 0.8899 | Val loss 0.3402 | test_acc 0.8879 | test loss 0.3578
Epoch 49 | Train acc. 0.9554 | Train loss 0.1277 | Val acc. 0.8892 | Val loss 0.3389 | test_acc 0.8885 | test loss 0.3568
Epoch 50 | Train acc. 0.9559 | Train loss 0.1263 | Val acc. 0.8900 | Val loss 0.3370 | test_acc 0.8885 | test loss 0.3552
```

(epoch = 50 , batch size = 32 , activation function = ReLU)

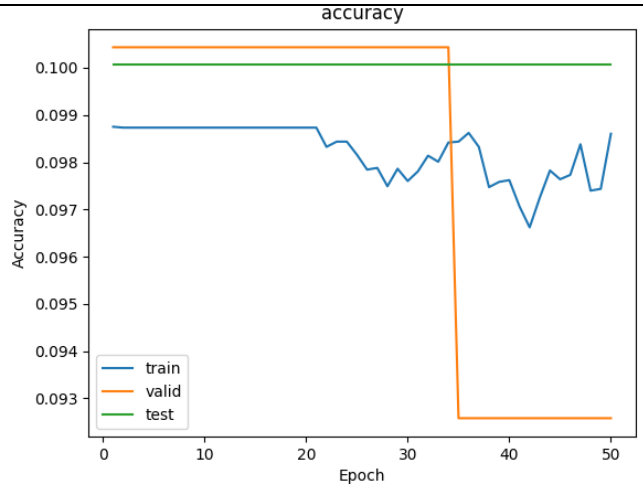
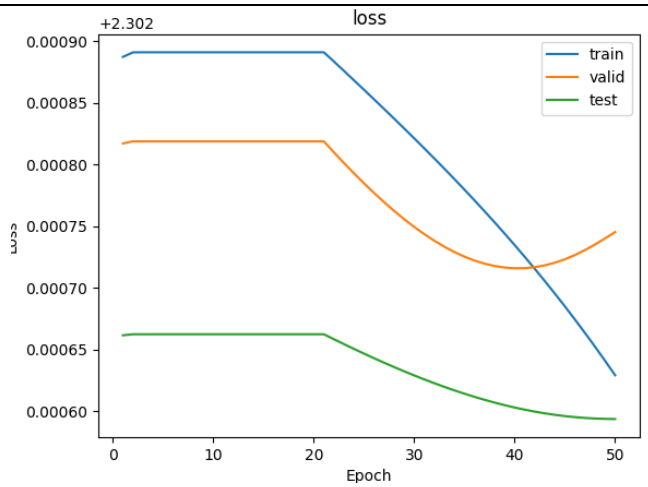
4. compare with deep NN



討論:

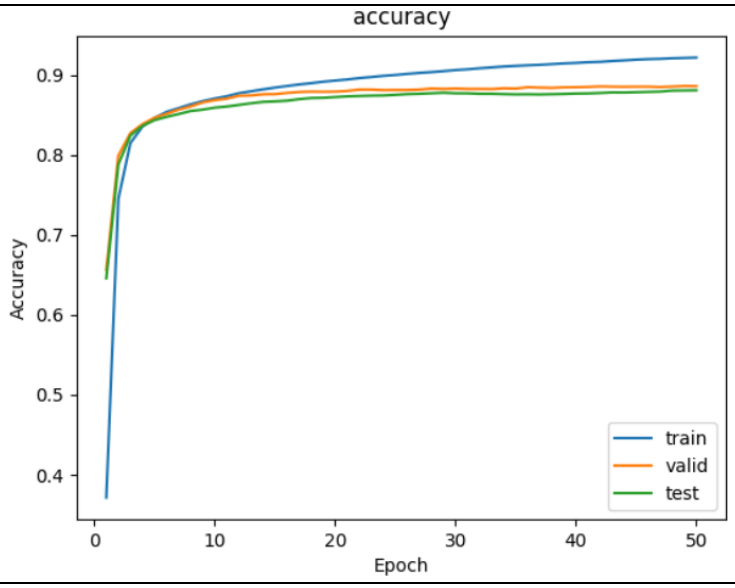
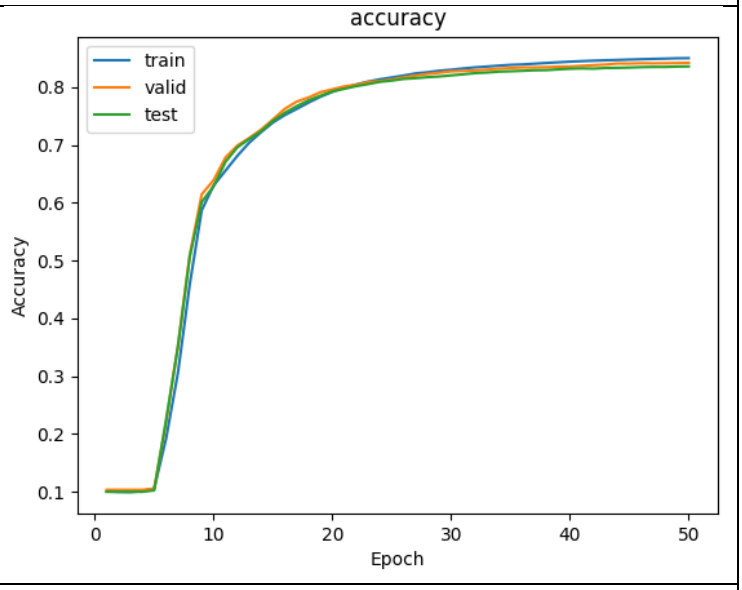
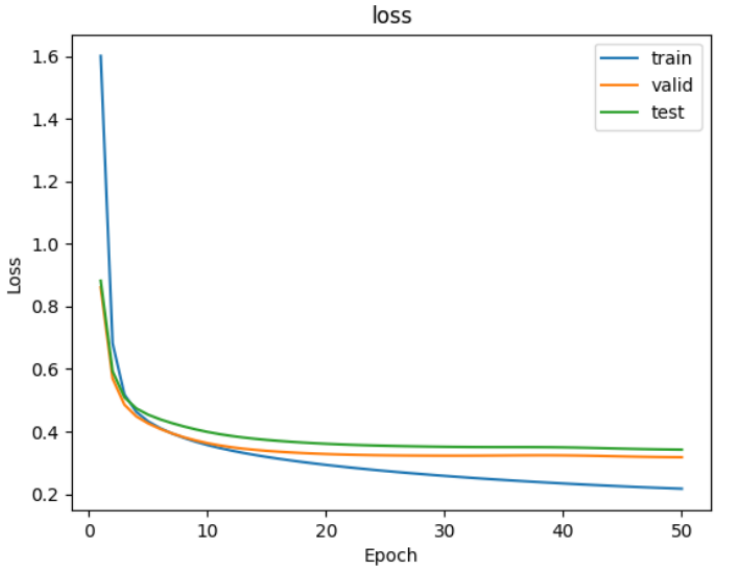
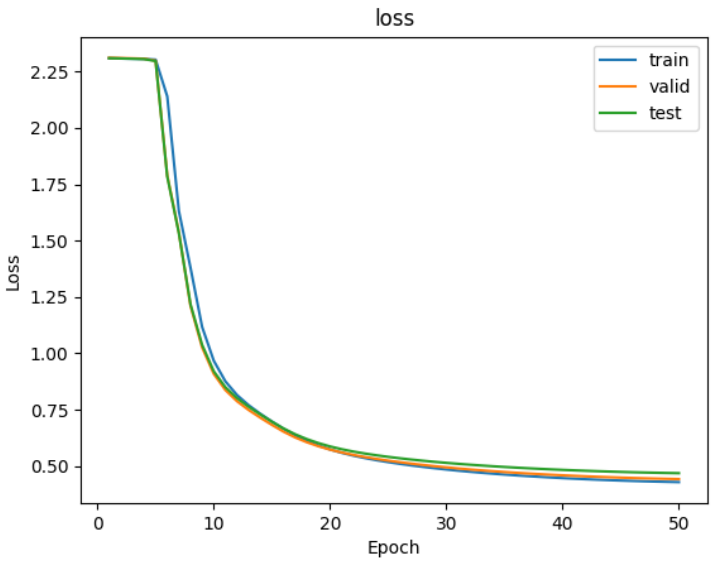
多加了兩層的 Full_connected layers，這兩層皆不改變 data 的維度，純粹加深模型的複雜度。可以發現模型的表現相當差，可見 NN 越深，不見得越好，因為 NN 太深時，會無法將資料的 feature 留住，導致 data 消散過大，因此深度應該視 dataset 的複雜度而做出相對應的設計。

```python
def pass_forward(data_input , label_input):
    output = flatten.forward(data_input)
    output = fc_layer1.forward(output)
    output = relu1.forward(output)
    # output = sigmoid_1.forward(output)
    # output = tanh_1.forward(output)

    output = fc_layer4.forward(output)      #deep
    output = relu3.forward(output)


    output = fc_layer2.forward(output)
    output = relu2.forward(output)
    # output = sigmoid_2.forward(output)
    # output = tanh_2.forward(output)

    output = fc_layer7.forward(output)      #deep
    output = relu6.forward(output)

    output = fc_layer3.forward(output)
    output = softmax.forward(output)
    accuracy , loss , predict = crs_etp.forward(output, label_input)
    return accuracy , loss , predict
```

	Deep NN (epoch = 50 , batch_size = 32)
Accuracy	
Loss	

5. compare to other two activation functions(sigmoid , tanh)

Condition : Epoch = 50 , batch_size = 32

	tanh	sigmoid
Accuracy		
Loss		

討論:

由上圖可得，activation function 為 tanh 時，accuracy=87.5%，為 sigmoid 時，accuracy=84.7%，兩者相較於 ReLU 來說，表現較差。因為 sigmoid function 有 vanishing gradient problem,導致淺層的權重變化越來越少，且函數收斂的速度也比較慢，造成相同 condition 下，accuracy 相對低於另外兩者。而 tanh 相較於 sigmoid，在 zero center 的地方擁有較陡峭的斜率，但在 back propagation 時一樣會有梯度消失的問題，只是會較輕微，另外 tanh 函式因為包含大量的 Exponetial 計算，需耗費大量的運算資源。相反的，ReLU function 並不會有梯度消失的問題，且因為擁有固定的斜率，在微分之後的計算相對容易，因此收斂速度比另外兩者都還要快。