Overview

本次作業是要針對已經事先寫好的類神經網路骨幹做填空和修改，資料集是 28*28 的圖片對 47 個 class 進行判別。這次的作業剛好逢自己去開刀動手術，有點不剛好，所以 CNN 的部分就來不及鑽研做出來，但也慶幸助教讓我晚交不扣我分，所以在此「感恩助教！讚嘆助教！」，謝謝助教的寬宏大量 😊。

雖然如此，我還是很努力地試圖增進我的 performance，我對我的模型作了以下的改變：

Improvements & Tweaks

1. **在每個 epoch 進行之前將訓練資料重新洗牌：**

   為了減少 temporal data 對訓練的影響，我在每個 epoch 進行之前將 x、y training 資料綁起來，重新 shuffle 一次然後再拆開做 labels 的 one hot。當然，這樣的話很多 code 的寫法就要稍微修正，像是做 one hot 的地方，以及 validation set 在一開始就要切好，不能直接在 iteration 進行當中從 train data 取 subset，否則這樣的話因為 shuffle 的關係，epoch 與 epoch 間的 training data 很可能成為下一個 epoch 的 validation data，但這樣就會失去 validation 的意義，因為等於模型有對 validation set 進行訓練，loss 會不斷下降、accuracy 不斷上升，最終導致 overfitting。

2. **修改 validation set 的 batch size**

   原先設定 batch size 的意義就是怕在 training 的時候因為資料筆數太多，如果每跑完一筆資料就要更新 weights 會太沒有效率，所以將一塊一塊的資料綁在一起一次訓練更新，training data 是這樣但 validation data 就不一定了。因為 validation data 不是我們要訓練拿來更新我們權重的資料所以沒有更新效率的問題，會把 validation data 分成 batch 的原因就是怕在 validate 的時候記憶體一次塞不下，而我這次把 validation 的 batch size 另外做設定，讓他是 validation size 的因數，可以整除 validation set，這樣就不會有時因為 batch size 的整除問題導致一些資料沒有 validate 到，可以做更完善的 validation。

3. **改變網路層級數**

   本來的層數是 2，我在經過測試後將其改為 4，基本上，只要超過兩層的網路都可以有效模擬任何有限範圍的函數，在大於兩層就純粹靠試驗了，而我發覺 4 層的效果不錯。

4. **改變神經元個數**

   神經數的設定我在查資料的過程當中眾說紛紜，跟網路層一樣沒有確切的答案都要靠試驗以及嘗試，但因為神經數總不可能一個一個累加看模型的表現所以我抓住了幾個要點去設計自己的神經元個數：
   - 神經數藉由輸入層和輸出層大小之間
   - 神經數大概是輸入層大小的 2/3 加上輸出層

   所以在中間夾雜的兩層網路維度我分別取了先前提到的

$$\frac{2}{3} \times layer_{input} + layer_{output} = Number\ of\ Neurons\ \approx 570$$

以及

$$\frac{(layer_{input} + layer_{output})}{2} = Number\ of\ Neurons \approx 415$$

最後一層的 classifier 就用原本的 60。

5. 調 Learning Rate

5.1 The Traditional Method

LR 大小在我所有參數裏頭花最多心思也應該是影響最大的參數，首先我用了不同的 learning rate 去跑以上已經建出來的模型，結果發現因為 learning rate 愈小，所需要的 epoch 就自然愈多，但這樣一個一個訓練太沒有效率了，我就決定進一步增強自己模型的訓練方式，給它加了個 Adam optimizer！

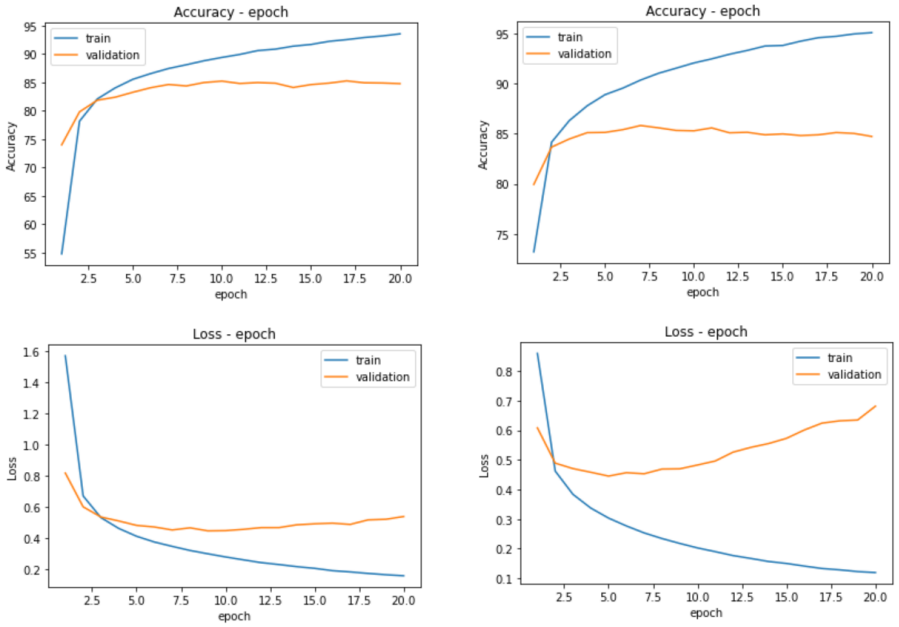
5.2 With Adam Optimizer

因為怕動到好不容易建起的模型，所以我先在未調過神經層、神經元個數的預設檔寫入 adam optimizer，參考 TensorFlow 還有當年發表 Adam 的 paper 預設的 learning rate、beta、epsilon，結果沒想到竟然跑不動，accuracy 一直卡在零，後來把 adam 移到四層的模型中就成功地跑出來了，可見神經網路的深度真的還是對 training 有影響，而且收斂速度真的很快，又不用調太多參數，很快的讓我 train 我的 model。這次配合先定義好的架構，我將自己的 adam optimizer 放在 Fully Connected 的 class 定義裏頭，這樣好分別存 weight 和 bias 的 w 和 v，設計如下：

```
def adam_weightGrad(self, iter, beta1=0.9, beta2=0.999, epsilon=1e-08):
    self.adam_m_weight = beta1 * self.adam_m_weight + (1 - beta1) * self.weight_grad
    self.adam_v_weight = beta2 * self.adam_v_weight + (1 - beta2) * (self.weight_grad)**2
    m_hat = (self.adam_m_weight / (1 - beta1**iter))
    v_hat = (self.adam_v_weight / (1 - beta2**iter))

    return m_hat / (np.sqrt(v_hat) + epsilon)

def adam_biasGrad(self, iter, beta1=0.9, beta2=0.999, epsilon=1e-08):
    self.adam_m_bias = beta1 * self.adam_m_bias + (1 - beta1) * self.bias_grad
    self.adam_v_bias = beta2 * self.adam_v_bias + (1 - beta2) * (self.bias_grad)**2
    m_hat = (self.adam_m_bias / (1 - beta1**iter))
    v_hat = (self.adam_v_bias / (1 - beta2**iter))

    return (m_hat / (v_hat**0.5 + epsilon))
```

```python
def update(self, lr, optimizer="SGD", curr_epoch=None):
    ##
    if optimizer == "SGD":
        self.fc1.weight -= lr*self.fc1.weight_grad
        self.fc1.bias -= lr*self.fc1.bias_grad

        self.fc2.weight -= lr*self.fc2.weight_grad
        self.fc2.bias -= lr*self.fc2.bias_grad

        self.fc3.weight -= lr*self.fc3.weight_grad
        self.fc3.bias -= lr*self.fc3.bias_grad

        self.classifier.weight -= lr*self.classifier.weight_grad
        self.classifier.bias -= lr*self.classifier.bias_grad

    elif optimizer == "adam":
        if(curr_epoch == None): raise Exception("Provide epochs for Adam Optimizer")
        self.fc1.weight -= self.fc1.adam_weightGrad(iter=curr_epoch)*lr
        self.fc1.bias -= self.fc1.adam_biasGrad(iter=curr_epoch)*lr
        self.fc2.weight -= self.fc2.adam_weightGrad(iter=curr_epoch)*lr
        self.fc2.bias -= self.fc2.adam_biasGrad(iter=curr_epoch)*lr
        self.fc3.weight -= self.fc3.adam_weightGrad(iter=curr_epoch)*lr
        self.fc3.bias -= self.fc3.adam_biasGrad(iter=curr_epoch)*lr
        self.classifier.weight -= self.classifier.adam_weightGrad(iter=curr_epoch)*lr
        self.classifier.bias -= self.classifier.adam_biasGrad(iter=curr_epoch)*lr

    else: raise Exception("Unrecognized Optimizer")
```

比較傳統的 learning rate 和加了 adam optimizer 果然收斂速度不一樣！



traditional gradient descent　　with adam optimizer

由上圖中可以看到傳統的 SGD 到了十幾個 epoch 才飽和，而 adam optimizer 在 5~7 個 epoch 就差不多飽和了。

把最終結果上傳到 Kaggle 上面可以得到 84.8%的準確度。