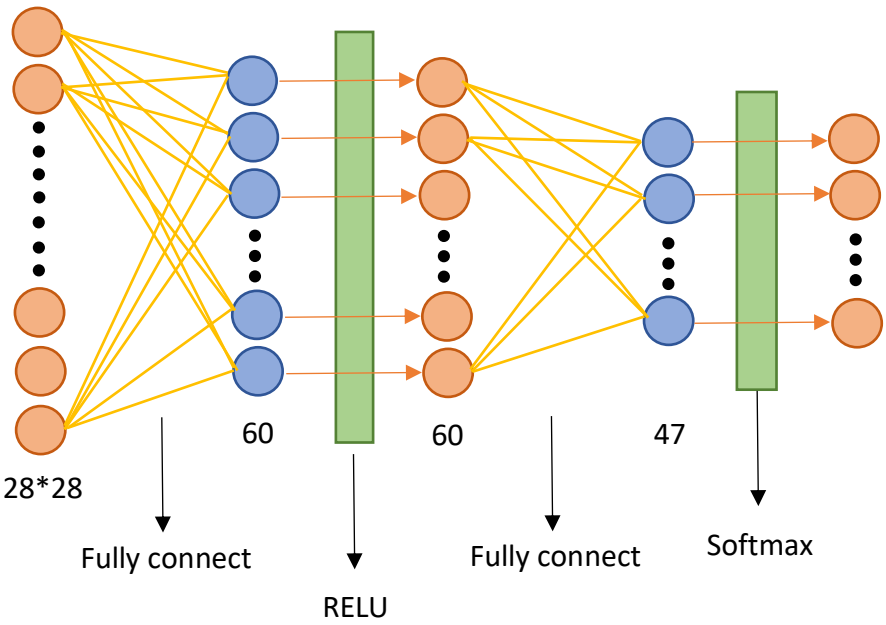
LAB3 BACK PROPAGATION

一、架構



二、Fully connect

1.forward

```
29      def forward(self, input):
30          ###useful at backward
31          self._save_for_backward = input
32          output = np.dot(input, self.weight)+self.bias
33
34          ######
35
36          return output
```

根據公式$Y = X * W + b$

其中 Y 為 output，X 為 input，W 為 fully connect 權重，b 為 bias

2.backward

```
38      def backward(self, output_grad):
39          ############################
40
41          #p = np.tile(output_grad.sum(axis=1, keepdims=True),(1,60))
42          self.weight_grad = np.matmul((self._save_for_backward).T, output_grad)
43          self.bias_grad = output_grad.sum(axis=0, keepdims=True)
44          output = np.matmul(output_grad, (self.weight).T)
45          return output
```

分別計算 weight 和 bias 的更新斜率

根據公式，使用 forward 過程中預先存取的 input 值，乘以從 activation function backward 函式回傳的值，得到 weight 更新斜率

再使用從 activation function backward 函式回傳的值，乘以 weight 矩陣，回推得到對下一層 layer backward 輸入值，當作 fully connected backward 函式輸出

三、RELU

```
class ReLU(_Layer):
    def __init__(self):
        pass

    def forward(self, input):
        self._save_for_backward = input

        output = np.maximum(input, 0)

        return output

    def backward(self, output_grad):
        input_grad = output_grad.copy()
        input_grad[self._save_for_backward < 0] = 0

        return input_grad
```

1.forward

RELU activation function 會讓大於 0 的數字表現原本的大小，讓小於 0 的數字以 0 表現，故將 input 與 0 做比較，輸出經過比較後較大的結果

2.backward

根據 forward 儲存下來的 input 值，將從 fully connected backward 回傳的梯度值，其中當初 input 為負值的項的梯度改為 0

四、Softmax with cross entropy

1.forward

```
class SoftmaxWithCE(_Layer):
    def __init__(self):
        pass
    #input : train_data[it*Batch_size:(it+1)*Batch_size], target : train_label_onehot[it*Batch_size:(it+1)*Batch_size]
    def forward(self, input, target):
        self._save_for_backward_target = target
        '''Softmax'''
        maximum = np.max(input)
        input_predict = np.exp(input - maximum)
        predict = [row / sum(row) for row in input_predict]

        predict = np.array(predict)

        self._save_for_backward_predict = predict

        '''Average Cross Entropy'''
        loss = 0
        for i in range(predict.shape[0]):
            for j, k in zip(predict[i], target[i]):
                loss = loss - k * np.log(j)
        ce = loss / input.shape[0]

        return predict, ce
```

根據 softmax 公式$\mathrm{y} = \frac{e^a}{e^a+e^b+e^c}$

將 input 取 exponential 後，取得每一項對總和所占比重，結果以機率形式表示

根據 cross entropy 公式$\mathrm{L} = -\sum_i t_i \log y_i$

將 predict 結果取 log 跟正確結果做比較，計算 loss 值

2.backward

```
    def backward(self):
        #########
        input_grad = self._save_for_backward_predict - self._save_for_backward_target

        return input_grad
```

五、network

```
    def forward(self, input, target):
        ##example
        h1 = self.fc1.forward(input)
        a1 = self.relu1.forward(h1)
        h2 = self.fc2.forward(a1)
        a2 = self.relu2.forward(h2)
        out = self.classifier.forward(a2)
        pred, loss = self.smce.forward(out, target)

        return pred, loss

    def backward(self):
        input_gradient_smce = self.smce.backward()
        input_gradient_relu2 = self.classifier.backward(input_gradient_smce)
        input_gradient_fc2 = self.relu2.backward(input_gradient_relu2)
        input_gradient_relu1 = self.fc2.backward(input_gradient_fc2)
        input_gradient_fc1 = self.relu1.backward(input_gradient_relu1)
        input_gradient = self.fc1.backward(input_gradient_fc1)

```

為了提升表現，根據上課老師提到的 neural network 層數越多，深度越深，學習效果越好，所以我多加了一層的 fully connected layer 和一層 RELU 在 classifier 之前，結果明顯提升 acc

```
    def update(self, lr):
        ##
        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.classifier.weight -= lr*self.classifier.weight_grad
        self.classifier.bias -= lr*self.classifier.bias_grad

```

每做完一次 batch 後，就利用 fully connected backward 函式裡存下來的梯度值，乘以設定好的 learning rate，更新一次 weight 和 bias 的數字

六、設定參數

Hyperparameters

```
EPOCH = 50
Batch_size = 200
Learning_rate = 5e-6

```

一開始 learning rate 設太大，所以 validation 的 acc 一直在很低的數值間來回震盪，後來試著調小 learning rate，並且根據網路上的資料建議，當 learning rate 調小時，batch size 也必須跟著變小，調整參數的結果對於提升 acc 也有一定幫助