HW5 成果分析

資料集狀況

CIFAR10 dataset 的資料是數筆 32*32 pixels 的彩色影像，並且有數個類別。

建立模型

和上週一樣，建立全連接層，但是這次可以設定不只一層：

```
for layer_i in range(len(dim_hidden)-1):
      fc_i += 1
      self.params[f'w{fc_i}'] =
torch.from_numpy(np.random.randn(dim_hidden[layer_i], dim_hidden[layer_i +
1])*0.2).to(self.dv)
      self.params[f'b{fc_i}'] = torch.zeros(dim_hidden[layer_i +
1]).to(self.dv)
      self.net.append(ReLU())
      self.net.append(FullyConnectedLayer(device=self.dv))
```

重點演算法和上次一樣，包含向前傳播、向後傳播、活化函數。
只是這次，我們使用三種梯度下降法，並且比較其結果。

梯度下降法

SGD

SGD是最簡易的梯度下降法，但是下降路徑並不是最佳的，所以會容易被雜訊影響。

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial \mathbf{L}}{\partial \mathbf{W}}$$

我們在反向傳播時已經算出 $\frac{\partial L}{\partial W}$ 了，所以只要再讀入，乘上學習率，這就是最簡單的 SGD。

```
params[key] -= self.lr * params_grad[key]
```

SGD+momentum

將 SGD 加上一階變數動量，讓梯度變化有加速度的概念，若是相同梯度，速度就會增加，反之減少。

$$v_t \leftarrow mv_{t-1} - \eta\frac{\partial \mathbf{L}}{\partial \mathbf{W}}$$

$$\mathbf{W} \leftarrow \mathbf{W} + v_t$$

```
self.velocity[key] = self.momentum * self.velocity[key] - self.lr *
params_grad[key]
params[key] += self.velocity[key]
```

Adam

然而，當梯度找到極值時仍會因為有速度繼續往前而導致 Overshoot 。所以必須新增一個能約束速度的調整項。

$$m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1)\frac{\partial \mathbf{L_t}}{\partial \mathbf{W_t}}$$

$$v_t \leftarrow \beta_2 v_{t-1} + (1-\beta_2)(\frac{\partial \mathbf{L_t}}{\partial \mathbf{W_t}})^2$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

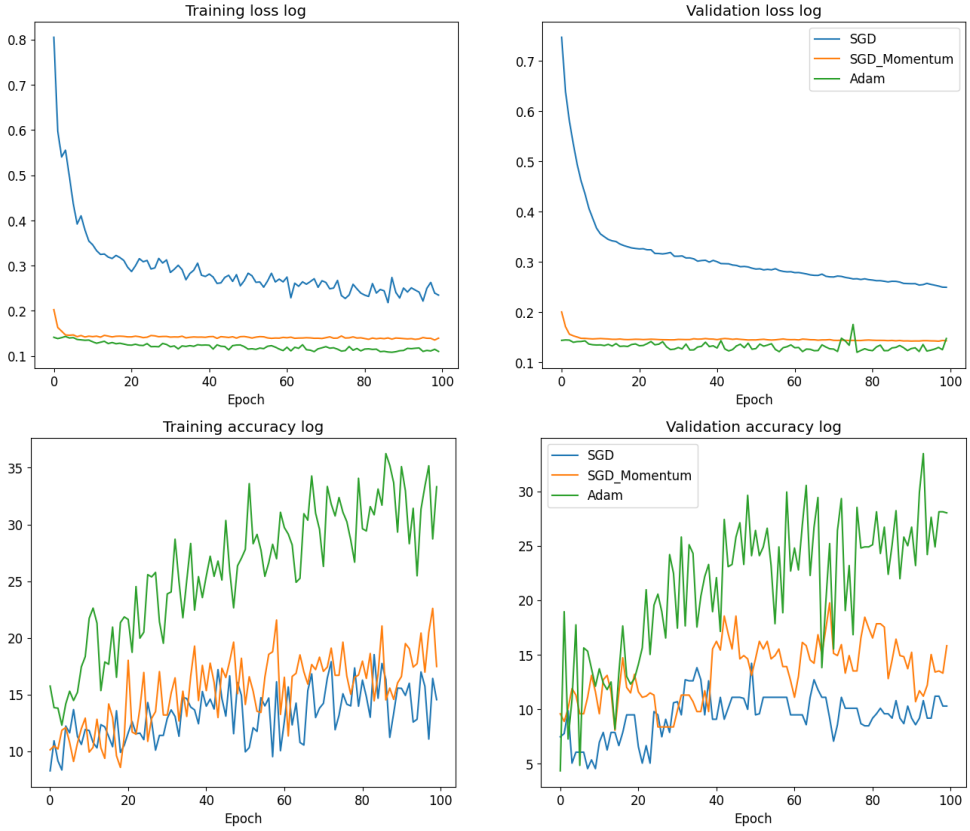
$$\mathbf{W} \leftarrow \mathbf{W} - \eta\frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\epsilon}$$

新增一個二次項，可以控制學習率的大小（也就是 Adaptive Learning Rate）。當累積梯度愈多（即跌代愈多次）時，會讓整體係數變小，進而達到收斂的結果。

```
self.momentum[key] = self.beta1 * self.momentum[key] + (1 - self.beta1) *
params_grad[key]
self.velocity[key] = self.beta2 * self.velocity[key] + (1 - self.beta2) *
params_grad[key] ** 2
m_hat = self.momentum[key] / (1 - self.beta1 ** self.t)
v_hat = self.velocity[key] / (1 - self.beta2 ** self.t)
params[key] -= self.lr * m_hat / (torch.sqrt(v_hat) + self.epsilon)
```

比較

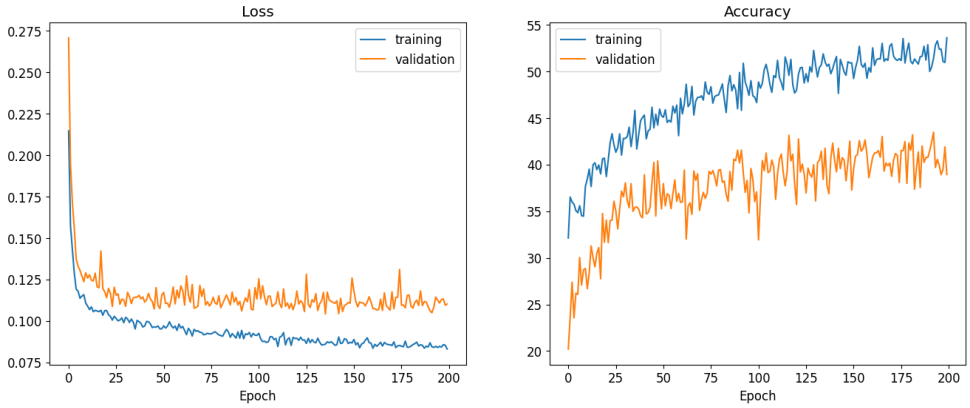
我們發現，Adam 確實有更好的效果。



執行結果與分析

多次嘗試之後，發現以下的設定可以達成準確率 40% 以上的目標：

```
BATCH_SIZE = 16
LR = 1e-4
DIM_HIDDENS = [180]
REG = 1e-6
OPTIMIZER = Adam(learning_rate=LR)
```



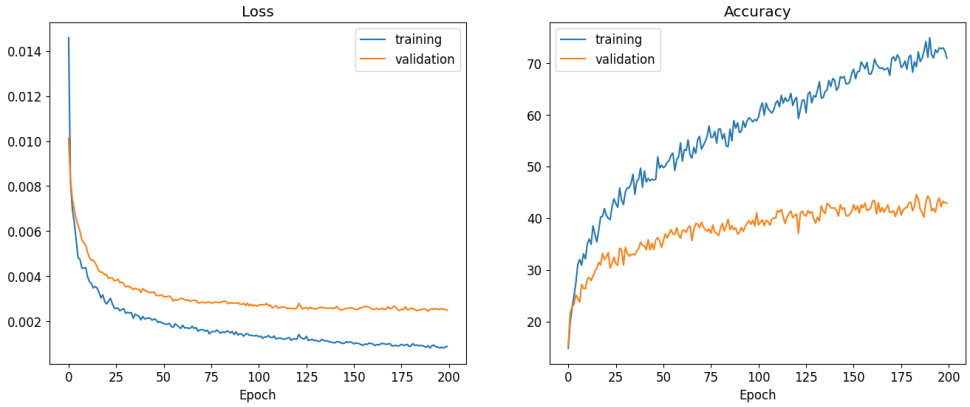

```
Got 4370 correct prediction in 10000 test data, accuracy: 43.70%
```

根據圖表，可以發現有些許的 Overfitting ，所以以下將持續改良模型。

改良

持續更改參數進行實驗。調整神經元數目可以讓模型的 capacity 增強，所以試著增加模型神經元數目。
然而，這會導致嚴重的 Overfitting ，所以也必須跟著提高正則項。
但我發現，以這樣的規則調整到一定的程度便無法再讓模型更好。此時，我無意間拿掉正則項，並且提高 Batch size ，發現有蠻好的表現：

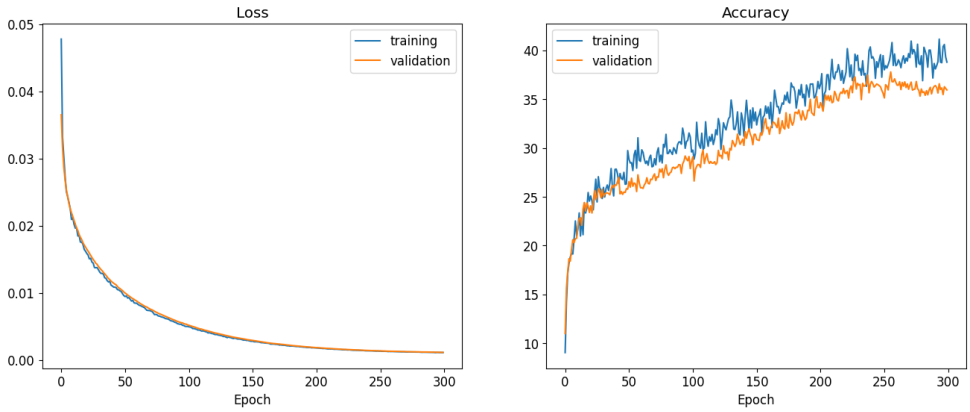
```
BATCH_SIZE = 1024
LR = 1e-4
DIM_HIDDENS = [1024]
REG = 0
OPTIMIZER = Adam(learning_rate=LR)
```




```
Got 4184 correct prediction in 10000 test data, accuracy: 41.84%
```

模型變穩定了，且仍有持續上升的趨勢。得出結論：
模型的 capacity 愈大，訓練時就需要愈大的 Batch size 來支持它。
然而，這個模型出現了嚴重的 Overfitting 現象，所以我重新加入正則項，並且提高訓練代數到300，想測試準確率有沒有繼續提高的可能：

```
BATCH_SIZE = 1024
LR = 1e-4
DIM_HIDDENS = [1024]
REG = 1e-4
OPTIMIZER = Adam(learning_rate=LR)
EPOCH = 300
```




```
Got 3702 correct prediction in 10000 test data, accuracy: 37.02%
```

可以發現第250個 epoch 前 Overfitting 的現象小了很多，雖然到最後還是有些微的 Overfitting 現象發生，且這個作法沒有如第一次的參數優秀，再加上訓練時間非常久(2hr)。
日後的嘗試應該會試著調整 Learning Rate ，讓梯度下降的速度加快。