**AI training HW3**

STuser19 賴昱凱

Best accuracy: 90.11%

**Optimizer:**

原先的training skills使用的是

**Normalization: None, Activation function: ReLU, Dropout: 0.2**

**Optimizer: SGD (lr=0.0001)**

**Scheduler: None**

可以看到accuracy為55.57%，十分不準確，因此我的第一個想法就是更改Optimizer，因為原先使用的SGD並沒有考慮先前的梯度，只使用當下梯度進行，因此非常容易被局部的梯度影響導致不容易降到local minimum，因此我考慮了幾個不同的Optimizer，包括SGD with weight、Adam、AdamW、Lamb等，更改之後的accuracy如下:

**SGD with weight (momentum=0.9, weight\_decay=0.0005): 85.72%**

**Adam (lr = 0.0001): 88.11%**

**AdamW (lr = 0.0001): 87.51%**

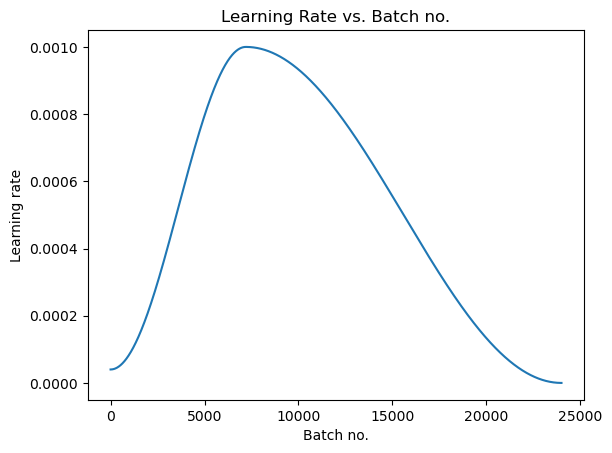
**Lamb (lr = 0.0001): 82.96%**

其中Lamb的準確率最低，因為Lamb適合的是batch size很大的情況，可是在本模型中batch size預設在20，是一個很小的batch，因此表現不佳也是可預期的。而SGD with weight也因為架構相對簡單，相較Adam來說，比較無法靈活改變weight。因此由比較的結果，後續我皆使用Adam來當作Optimizer。

**Scheduler:**

而第二步我把重點放在Learning rate的變化，由於Optimizer的Learning rate都是固定不變的，因此我們需要自己設定Scheduler來適當改變training過程中的Learning rate。

一開始我的想法是讓Learning rate有個warning up的趨勢，讓梯度先在初始點附近尋找適當的方向再加大步伐，因此我選擇了OneCycleLR，並先都使用預設的參數設定，Max\_lr則設定成0.001，得到結果accuracy為89.03%



**Dropout:**

Dropout的比例對於accuracy也有很大影響，由於這個模型小，架構也簡單，因此將dropout的比例降低是有利於避免在訓練模型時過度依賴某些樣本導致overfitting，進而使訓練結果不佳，而我多次嘗試也發現的確有此項趨勢。

**Dropout(0.5): 88.06%**

**Dropout(0.2): 89.03%**

**Dropout(0.1): 89.05%**

因此後續使用Dropout(0.1)進行訓練。

**Normalization:**

更改相關參數無法再有效增加accuracy，因此我把改進方向轉至Normalization上，我在前兩層計算結束後使用了Batch Normalization，accuracy結果為88.88%，反而變低了，我推測是由於會需要計算Batch的mean與variance，需要一定的樣本數量，因此原先的batch size = 20太少了，比統計學中對於大樣本數的定義30個還少，因此我將Batch size增加至100，使每一次Batch的樣本分佈更接近母體，結果提升到了89.38%，證明我的猜想是正確的。

**Activation function:**

接下來我想了解不同的activation function的效果如何，測試結果如下:

ReLU: 89.52%

PReLU: 88.48%

GELU: 89.57%

Softmax: 88.05%

Sigmoid: 85.44%

ReLU及GELU在此模型上表現差不多，且ReLU計算量明顯小於GELU，因此之後就繼續使用ReLU。

至於Softmax表現差的原因應該是因為當我們選擇使用CrossEntropyLoss當作loss function時，就已經在該函數中使用過Softmax了，因此再多使用一個並不會讓結果更好。而Sigmoid則是適合用在二分法，不適合用在此題目中，因此表現特別差。

**調整參數:**

在此時的情況下，我發現warning up並不會增加accuracy，反而增加初始的learning rate可以讓loss有更大幅度的下降，因此我將參數調整並尋找最佳的組合，最後使用的learning rate如下:

torch.optim.lr\_scheduler.OneCycleLR(optimizer, max\_lr=0.0035, steps\_per\_epoch=len(train\_loader), epochs=n\_epochs, div\_factor=0.7, final\_div\_factor=100, pct\_start=0.4)

而最後測試出來最好的training skills:

**Normalization: Batch Normalization**

**Activation function: ReLU**

**Dropout: 0.1**

**Optimizer: Adam**

**Scheduler: OneCycleLR(optimizer, max\_lr=0.0035, steps\_per\_epoch=len(train\_loader), epochs=n\_epochs, div\_factor=0.7, final\_div\_factor=100, pct\_start=0.4)**

其中我也有嘗試的調整Adam中的betas，但最後測試出來還是預設的表現最佳，因此皆使用預設的參數。

**Best result:**

