CIFAR10 Classification
B083040029 邱品諺

1.1 Dataset and Dataloader

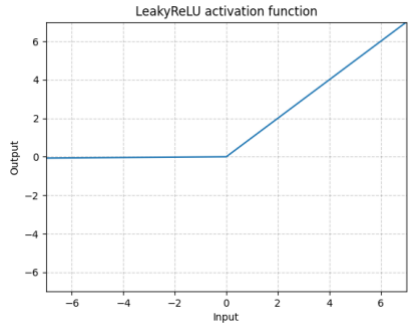
資料集處理及載入的部分和過去作業不同在於本次作業使用了 FlattenTransform，也就是將以多維陣列表示的圖片展開成一維陣列表示，此方式為給予神經網路訓練最簡單的圖片儲存方式，但其無法表示圖片中各物件的空間關係。

1.2 Data Augmentation

模型的訓練需要大量的資料，但人為產生的資料再多也有限，再加上需要人類標注，因此資料數量無法滿足訓練的需求，因此需要 Data Augmentation 來增加資料的數量及廣度，如此模型可更加的 general 及 robust。Data Augmentation 的方式有以下幾種：圖片翻轉、旋轉、調整飽合度、亮度、Cropping 等等。

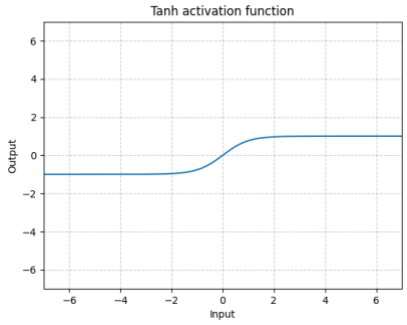
1.3 Layers

Activation Function 的部分除了 Sigmoid 及 ReLU 外，本次將介紹 LeakyReLU 和 Tanh。首先，LeakyReLU 和 ReLU 最大的差別在於小於 0 的部分不為 0，因此不會有 ReLU 只有右半邊能計算 Gradient 的情況，且能保有 ReLU 可快速收斂且不會飽和的優點。以下為 LeakyReLU 的計算式及函式圖：



$$f(x) = \mathbb{1}(x < 0)(\alpha x) + \mathbb{1}(x >= 0)(x)$$

而 Tanh 和 Sigmoid 相似，但 Tanh 較 Sigmoid 好的一點是他的中心點為 0，輸出範圍為 0~1，但仍然有和 Sigmoid 一樣會有飽和的問題，也就是在較大及較小的輸入值中，通過此 Activation Function 後所產生的輸出無法進行微分，亦或是輸出值已趨近於 0。以下為 Tanh 的計算式及函式圖：



$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

1.4 Regularization loss

計算 Loss Function 除了計算預測值與 Ground Truth 的差距外，還會加上 Regularization Loss 以防止模型 Overfitting， 也就是訓練的結果過度地符合訓練資料。而 Regularization Loss 之所以可以放止 Overfitting 是因為其可使模型變得較簡單、較 General，像是在 L2 Regularization，由於計算方式為將每個 Weight 取平方相加，因此如果有較大的 Weight 便會使得整體的 Loss 較大，因此經過更新後會使得 weight 間的變異較小，模型也不會為了要過度地學習 Training set 中某項偏差的資料而學到較極端、不符合整體情形的 weight。

2. An overview of hyperparameters

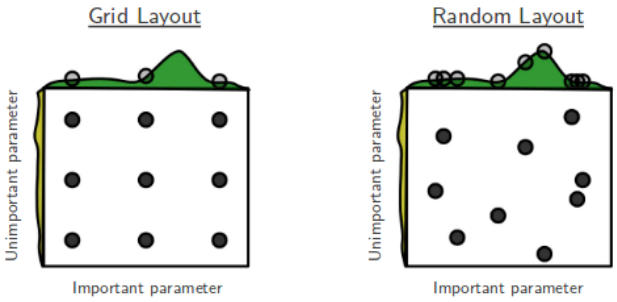
訓練神經網路除了讓 model 自己學習 model parameters，像是 weights, bias，另外，有些參數像是 hyperparameters 是在訓練前需要我們自己決定，並透過一次次的訓練結果來調整的，而這些 hyperparameters 包含網路的架構像是 Activation Function、網路的層數、每一層要有幾個 Neurons，Learning rate、epochs、Batch size、Regularization strength 等等。

而在訓練網路一開始，應該從少量的資料開始訓練，從 1、10、1000 慢慢增加資料的數量，一開始我們會發現在訓練資料極少的情形下訓練的準確率都是 100%，那是因為模型並沒有學到東西，他只是背下來而已，因此在 Validation set 的準確率非常差。資料量越來越多後，會發現 training 和 validation 的 loss 都開始下降，但總在半路上就開始反升，此時一大原因是因為資料量的不足，因此除了需要增加資料量外，其實我們可以觀察訓練的結果找出規律，像是通過哪些 Activation Function 較好、幾層的神經網路結果較好等等，以調整 Hyperparameters，但是手動調整這些參數實在是一個不太有效率的做法，因此本作業介紹了兩種除了手動挑整參數外的參數調整方式。

3. Hyperparameter Tuning

首先為 Grid Search，透過指定每個參數的可能組合進行訓練，像是使用者指定三種 learning rate，regularization strength 有兩種，因此將有 3x2=6 種訓練組合，如此我們可以觀察訓練的結果來調整 hyperparameters。

但有時候我們無法準確的知道確切的參數進行訓練並比較各個參數組合的好壞，我們只約略知道大概的範圍，因此有了 Random Search 的方法。Random Search 使得我們可以決定各參數的範圍，在每次訓練隨機抽樣範圍中的數值進行訓練，並觀察多次的訓練結果以調整參數。



4. Let’s find the perfect model
   依照上述訓練的方式，資料量由小到大進行訓練，並觀察參數及 loss 的變化以調整 hyperparameters。

   從 1000 筆開始訓練增加到 10000 筆，再到 20000 筆，在 1000 筆時發現 learning rate 在 1e-3~1e-5 時的訓練情形較好，太大的話會 loss 有可能會不降反升，而太小的話可能會下降的太慢。另外，神經網路的層數在 2、3 層的訓練結果較好，層數再多結果非但沒有比較好且耗時。而在 Activation Function 的部分，ReLU 及 LeakyReLU 的表現較好。而由於資料量的不足加上 early stopping 的機制，因此增加 epoch 數也無法透過觀察出 loss 的變化調整參數。

   資料數增加後，會發現 learning rate 應該在小一點，限縮在 1e-4~1e-5 間，另外，層數在 2 層時，且 neuron 數在 200 上下時普遍表現較好，因此也將 epoch 數量加大以觀察 loss 的變化。

```
model_class = ClassificationNet

best_model, results = random_search(
    dataloaders['train_small'], dataloaders['val_500files'],
    random_search_spaces = {
        "learning_rate": ([1e-3, 1e-5], 'log'),
        "reg": ([1e-4, 1e-5], "log"),
        "hidden_size": ([150, 400], "int"),
        "num_layer": ([2, 3], "int"),
        "activation": ([Relu(), LeakyRelu()], "item"),
        "loss_func": ([CrossEntropyFromLogits()], "item")
    },
    model_class=model_class,
    num_search = 10, epochs=10, patience=3)
```

```
[ ] model_class = ClassificationNet

best_model, results = random_search(
    dataloaders['train_small'], dataloaders['val_500files'],
    random_search_spaces = {
        "learning_rate": ([2e-5, 7e-5], 'log'),
        "reg": ([1e-4, 1e-5], "log"),
        "hidden_size": ([180, 220], "int"),
        "num_layer": ([2], "int"),
        "activation": ([Relu(), LeakyRelu()], "item"),
        "loss_func": ([CrossEntropyFromLogits()], "item")
    },
    model_class=model_class,
    num_search = 10, epochs=15, patience=3)
```

(由於中間 10000 筆資料的訓練過程被 20000 筆的覆蓋因此未顯示)

```
from exercise_code.networks import MyOwnNetwork
from exercise_code.hyperparameter_tuning import random_search
best_model = ClassificationNet()

################################################################################
# TODO:                                                                        #
# Implement your own neural network and find suitable hyperparameters  #
# Be sure to edit the MyOwnNetwork class in the following code snippet #
# to upload the correct model!                                                 #
################################################################################
best_model, results = random_search(
    dataloaders['train_20000files'], dataloaders['val_4000files'],
    random_search_spaces = {
        "learning_rate": ([3e-05], 'log'),
        "reg": ([1e-4, 1e-5], "log"),
        "hidden_size": ([200], "int"),
        "num_layer": ([2], "int"),
        "activation": ([LeakyRelu()], "item"),
        "loss_func": ([CrossEntropyFromLogits()], "item")
    },
    model_class=ClassificationNet,
    num_search = 1, epochs=20, patience=5)
################################################################################
#                            END OF YOUR CODE                                  #
################################################################################
```

```
Train Accuracy: 60.456730769230774%
Validation Accuracy: 49.168669871794876%
```

```
Test Accuracy: 48.617788461538478%
```

由訓練結果可以發現已經 overfitting，因此或許需要增加 regularization 的強度，亦或是利用 Data Augmentation 增加訓練資料，或是增加訓練及驗證的資料數量加以調整參數，才可改進準確率。