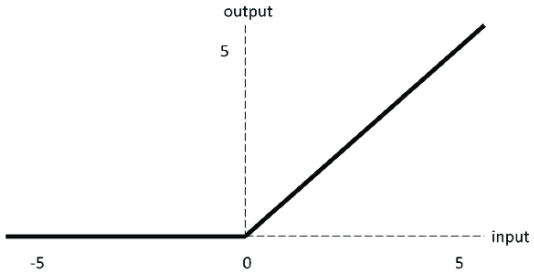
Neural Networks and CIFAR10 Classification
B083040029 邱品諺

在作業 4 我們實作了 Logistic Regression，其為單層的 Network，且在作業 4 只利用 1 項 Feature（也就是一個 Neuron）訓練，也就是線性方程（f=Wx）再通過一個 Non-linearity（Sigmoid）。但 Deep Learning 之所以叫 Deep，表示 Network 通常不會只有一層，而本次作業的目的即為以 CIFAR10 為資料集建立一個多層的神經網路。

1. Neural Network Models
    a. Modularization
    訓練多層神經網路需要透過 Loss Function 對 Model Parameters 的微分得知各參數的更新方向，以持續更新至模型收斂。由於多層的神經網路有許多 Model Parameters，因此在實務操作上需要在 forward 的時候記錄神經網路中各個 Model Parameters，以方便在 Backward 時方便計算 Gradient。

    b. Layer Example: Non-Linearities
    除了作業 4 使用的 Sigmoid 外，ReLU 為現今 Deep Learning 最常用的 non-linearity，下圖為 ReLU 的函式圖：

$$ReLU(x) = max(0, x)$$



    c. Affine Layers
    此定義了每一層 Network 的 Forward 及 Backward 方式。Forward 主要透過以下方程式處理 Neuron 間的計算：

$$z = Wx + b$$

    而 Backward 主要為計算 Loss Function 對各 Model Parameters 的微分，核心概念主要為 Chain Rule，以上式為例：需先計算 Loss Function 對 z 的微分，再計算 z 對 Model Parameters 的微分。以下為計算式：

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial z} \cdot \frac{\partial z}{\partial W} \qquad \frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} \cdot \frac{\partial z}{\partial x}$$

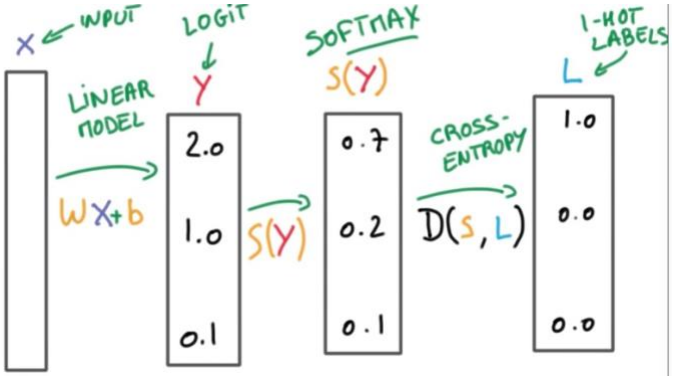
    但在電腦運算中，資料多以矩陣儲存，下列為矩陣的表示方式：

$$\frac{\partial L}{\partial W} = X^T \frac{\partial L}{\partial z} \qquad \frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} W^T$$

d. N-layer Classification Network
classification_net 此函式主要包含了 Linear layer 的 Forward 及 Backward、Activation Function 的 Forward 及 Backward，並紀錄儲存每一層的 Model Parameters 以及 Backward 後 Loss Function 對各個 Model Parameters 的微分。另外，可透過輸入參數決定 Input Layer、Hidden Layer、輸出類別的數量，以及神經網路的層數和 Activation Function 的種類。

2. CIFAR10 Dataset
CIFAR10 為 50000 張 32x32x3 訓練資料的資料集，而本作業中取 1%（500 張）作為欲建立模型之訓練資料，且 Batch 大小為 8，因此如果 drop last 的話會有 62 個 batch，而 input layer 的 neuron 數為 32x32x3=3072。

3. Cross-Entropy/Softmax Loss From Logits



首先，Softmax 為將輸出結果轉換為[0,1]機率分佈的函式，先透過 exponential 將輸出轉換為保證>=0 的數值，再透過 Normalization 轉換為[0,1]間的機率分佈。另外，Softmax 其實是 Sigmoid 的 multi-class、廣義版本，而 Sigmoid 為 two-class 的版本。以下為 Softmax 的函式：

$$softmax(x) = \sigma(x) = \frac{e^{x_i}}{\sum_{i=1}^{n} e^{x_i}}$$

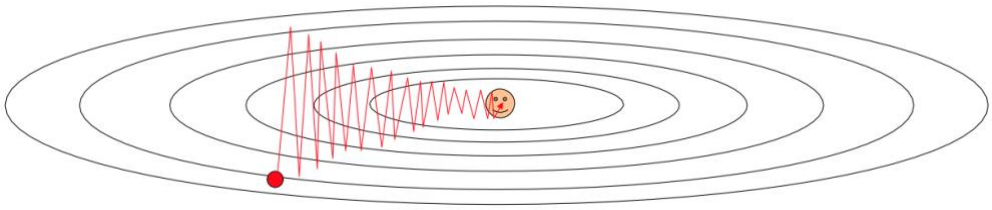
另外，Softmax 會有 Numerical Stability 的問題，也就是當輸入至 Softmax 函式的數值太大時進行 exponential 運算時會 Overflow，因此需要減去所有的輸入減去輸入中的最大值，以確保 Softmax 不會 Overflow。

接著，Logit 為進入 Softmax 前的 layer，與其他 layer 不同的地方為其沒有通過 Activation Function。最後，Cross-Entropy 為一種計算分類問題的 Loss Function，E 透過計算 Cross-Entropy 可得知模型輸出的預測結果與目標相差多少，以下為 Cross-Entropy 的函式：

$$CE(\hat{y}, y) = \frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{C}[-y_{ik}log(\hat{y}_{ik})]$$

4. Optimization
   a. Gradient Descent vs Stochastic Gradient Descent
   如果以 Gradient Descent 的方式更新 Network，將以整個資料集作為輸入進行訓練，這樣進行訓練雖然可得到很好的結果，但是訓練的速度十分緩慢且需佔用非常大的記憶體空間，因此不是一個具有效益的方法。

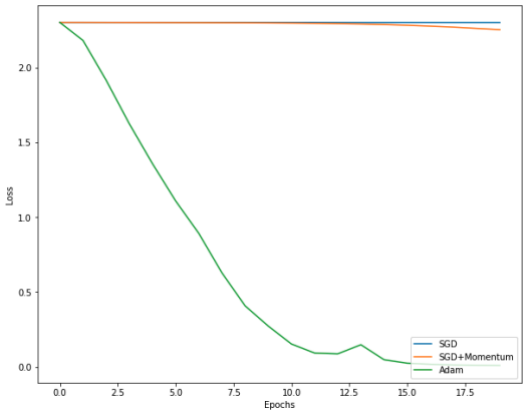
   b. SGD
   SGD 由於其每次輸入進 Model 的資料為隨機取點或 mini-batch 以進行訓練，因此可透過平行化改善 Gradient Descent 訓練速度緩慢的問題，但訓練結果可能較為 noisy，由於資料的隨機性因此可能造成在梯度下降時的方向可能都不一樣而導致下圖的情形，而我們想要的是一個既訓練速度快又可以穩定向最低點下降的方式。



   c. SGD + Momentum
   增加動量的方式改善了上述問題，因為在每次更新時多考慮了前一次的下降速度，如此不僅可修正下降的方向外，又可保留前一次的速度以適應當時的地形進行不同的更新方式。以下為 SGD + Momentum 的函式：

$$v^{k+1} = \beta v^{k} - \alpha\nabla_{\theta}L(\theta^{k})$$
$$\theta^{k+1} = \theta^{k} + v^{k+1}$$

   d. Adam
   Adam 主要結合了 1st 及 2nd order 的 momentum。在訓練執行 20 個 epochs 後，相較於上述的更新方式，Adam 有較快的更新速度，由下圖可見更新速度的差異：



5. 最後，Neural Network 更新的一個 epoch 為下列步驟：
    a. Affine layer 通過每一層的 Activation Function 直到 Logit 層
    b. Softmax 轉換成[0, 1]機率分佈
    c. Cross Entropy 計算 Data Loss
    d. 由 Weights 計算 Regularization Loss
    e. 由 Total Loss 對 Model Parameters 進行更新