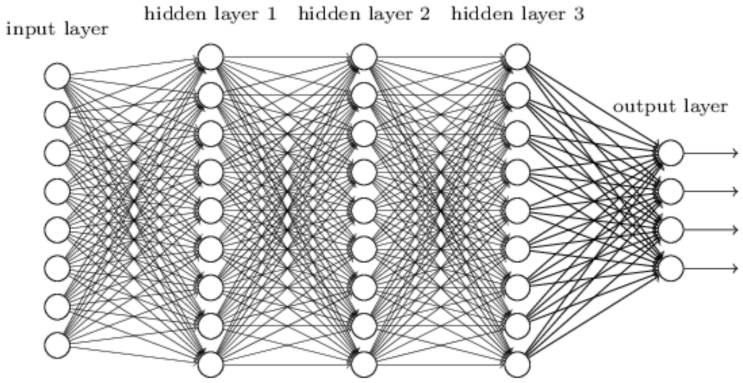
Deep Learning

深度學習 (deep learning) 是機器學習的分支，是一種以人工神經網路為架構，對資料進行表徵學習的演算法。在隨後討論神經網路的原理時，我們會對表徵學習 (representative learning) 這個名詞有更深刻的體會。

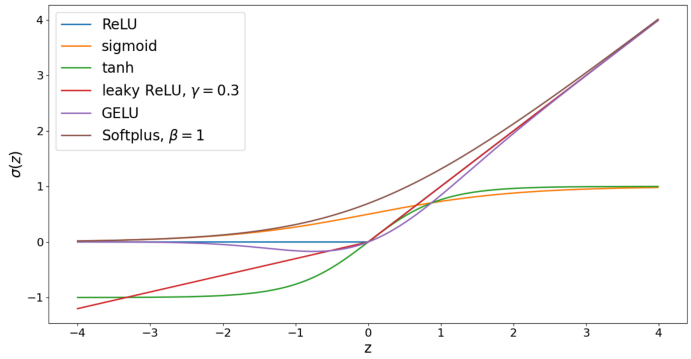
Introduction to Neural Networks

我想用與一般教科書不同的方式介紹神經網路。神經網路的結構是經過不斷更新的理論與無數次實驗得到的，而不是直接被推導出來的，它的運作原理非常抽象、複雜，而教科書在一開始為了方便讀者理解而給出的解釋，容易讓人產生誤解。具體來說，首先我會直接給出神經網路的結構，隨後介紹優化算法——反向傳播 (backpropagation)，再從這些結構與算法出發，回頭解釋神經網路實際上在做什麼。
神經網路的目的是什麼？神經網路具有非常強大的學習能力，能夠適應各種非線性分佈的數據，因此廣泛應用於迴歸任務與圖像辨識（將每個像素視為輸入特徵，學習出不同圖像的決策邊界）。
它的構造如下：



圖一：神經網路

第一層稱為輸入層 (input layer)，它的每個節點對應到輸入特徵向量 x 的一個分量。如果總共有 m 個特徵，則輸入層就會有 m 個節點，分別代表 $x_1, \ldots, x_m$。我們定義這裡的 x 不像 GLM 那樣預設含有 $x_0 = 1$ 的偏差項。輸入層本身不進行任何運算，它只負責將數據傳遞給下一層。
接下來是中間的隱藏層 (hidden layers)，這些層中的每個節點稱為神經元 (neuron)，神經元會用前一層的特徵產生線性預測器，並加以非線性化，通過一個非線性的激活 (activation) 函數 $\sigma(\cdot)$，目的是起到類似 perceptron 的作用，待會有更深的解釋。這樣的操作在每層中重複進行，前一層的輸出作為下一層的輸入，每個神經元都在執行 $\sigma(\theta^T x)$ 的操作。這樣一層層堆疊下來，我們可以把輸出層之前的神經網路看作是在學習一個複雜的非線性預測器表示 $\theta^T \phi(x)$。
最後一層稱為輸出層 (output layer)，可以理解成根據任務而決定的 hypothesis。例如圖一的輸出層有四個輸出，代表神經網路處理的可能是多分類任務，hypothesis 為 softmax 函數。



圖二：激活函數 $\sigma(\cdot)$

作為示例，接下來在這篇筆記中，激活函數指的是 ReLU (rectified linear unit)，其定義為：

$$\text{ReLU}(z) = \max(0,\ z)$$

下面的 $w^T x + b$ 只是 $\theta^T x$ 的另一種表示法，代表了權重 (weights) 跟偏置 (bias)（因為 x_0 已經不是 1）。
在後續討論中，神經網路的層數將只包含那些有參數（權重與偏差）並帶有非線性處理的層，也就是排除輸入層。因此，r 個隱藏層的神經網路做的事是：

$$x = a^{[0]},\ a^{[1]} = \text{ReLU}(W^{[1]}a^{[0]} + b^{[1]})$$

$$a^{[2]} = \text{ReLU}(W^{[2]}a^{[1]} + b^{[2]})$$

$$\vdots$$

$$a^{[r-1]} = \text{ReLU}(W^{[r-1]}a^{[r-2]} + b^{[r-1]})$$

$$\bar{h}_\theta(x) = W^{[r]}a^{[r-1]} + b^{[r]}$$

其中 $W^{[j]}a^{[j-1]} + b^{[j]}$ 是：

$$\begin{bmatrix} — W_1^{[j]T} — \\ — W_2^{[j]T} — \\ \vdots \\ — W^{[j]T} — \end{bmatrix} \begin{bmatrix} a_1^{[j-1]} \\ a_2^{[j-1]} \\ \vdots \\ a^{[j-1]} \end{bmatrix} + \begin{bmatrix} b_1^{[j]} \\ b_2^{[j]} \\ \vdots \\ b^{[j]} \end{bmatrix}$$

即為遞迴：

$$a^{[k]} = \text{ReLU}(W^{[k]}a^{[k-1]} + b^{[k]}),\ \forall k \in \{1,\ \cdots,\ r-1\}$$

注意神經網路的倒數第二層（最後一層隱藏層）通常不再是非線型的，而是普通的線性預測器。例如，因為模型的預測範圍也可以是負數，但 ReLU 是非負的。另外，因為在這之前所有神經元的輸出都是非線性的，所以最後一層隱藏層可以視為：

$$\bar{h}_\theta(x) = W^{[r]}\phi_\beta(x) + b^{[r]}$$

我們能用符號 MM (matrix multiplication) 代表矩陣乘法如下：

$$\text{MM}_{W,b}(z) = Wz + b$$

因此神經網路所做的操作（叫做 multi-layer perceptron, MLP）可以寫成：

$$\text{MLP}(x) = \text{MM}_{W^{[r]},b^{[r]}}(\sigma(\text{MM}_{W^{[r-1]},b^{[r-1]}}(\sigma(\cdots \text{MM}_{W^{[1]},b^{[1]}}(x)))))$$

或為了方便簡寫成：

$$\text{MLP}(x) = \text{MM}(\sigma(\text{MM}(\sigma(\cdots \text{MM}(x)))))$$

總結目前到手的資訊：已經知道每個神經元都會用上一層的輸出作為輸入，產生線性預測器，並做一個非線型的 activation。思考：所有神經元裡都有參數，數量非常多之外，又分散在不同的層。我們要如何優化這些參數？

Backpropagation

即使神經網路的結構這麼複雜，但優化方法還是一樣的，對於一筆輸入而言依然存在真實數據 $y^{(i)}$ 以及神經網路的輸出 $h_\theta(x)$。我們要處理的都是一些 GLM 模型的常見任務，所以不再從機率角度重新推導損失函數，而是直接定義單筆數據的 loss function 為：

$$J^{(i)}(\theta) = \frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2$$

cost function 為：

$$J(\theta) = \frac{1}{n}\sum_{i=1}^{n} J^{(i)}(\theta)$$

乘上一個常數不改變函數的凸性，還能讓多分類任務的成本函數直接變成 cross entropy。基本上，對於迴歸任務、二分類任務與多分類任務，都是完全一樣的算法，只是現在的 hypothesis 是由神經網路產生而已。

有了以上認識，就能直接進入反向傳播 (backpropagation)。反向傳播並不困難，它就是多元函數求導的連鎖律 (chain rule)。我們都知道，對於以下的對應關係而言：

$$z \in \mathbb{R}^m \xrightarrow{g:\mathbb{R}^m \to \mathbb{R}^n} u \in \mathbb{R}^n \xrightarrow{f:\mathbb{R}^n \to \mathbb{R}} J \in \mathbb{R},$$

$$\forall i \in \{1, \cdots, m\}, \frac{\partial J}{\partial z_i} = \sum_{j=1}^{n} \frac{\partial J}{\partial u_j} \cdot \frac{\partial g_j}{\partial z_i}$$

或以梯度表示（符號的部分是遵從教科書，雖然看起來像是一個標量）：

$$\frac{\partial J}{\partial z} = \begin{bmatrix} \frac{\partial g_1}{\partial z_1} & \cdots & \frac{\partial g_n}{\partial z_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial g_1}{\partial z_m} & \cdots & \frac{\partial g_n}{\partial z_m} \end{bmatrix} \cdot \frac{\partial J}{\partial u} = \left[\frac{\partial (g_1, \cdots, g_n)}{\partial (z_1, \cdots, z_m)}\right]^T \cdot \frac{\partial J}{\partial u}$$

由此知欲從 $\partial J/\partial u$ 求 $\partial J/\partial z$ 只需要 $g,\ z$ 的資訊。定義此 Jacobian 轉置為一個映射函數 $\mathcal{B}(\cdot)$，使得：

$$\frac{\partial J}{\partial z} = \mathcal{B}[g,\ z]\left(\frac{\partial J}{\partial u}\right)$$

開始處理優化的問題。因為對 cost function 求梯度滿足線性性質，所以我們能計算 loss function 再求和得之：

$$\nabla_\theta J(\theta) = \nabla_\theta \left(\frac{1}{n}\sum_{i=1}^{n} J^{(i)}(\theta)\right) = \frac{1}{n}\sum_{i=1}^{n}\left(\nabla_\theta J^{(i)}(\theta)\right)$$

為了系統化 loss function 的求導過程，將其視為一連串模組 (module) 的組合：

$$u^{[0]} = x$$

$$u^{[1]} = M_1(u^{[0]},\ \theta)$$

$$u^{[2]} = M_2(u^{[1]},\ \theta)$$

$$\vdots$$

$$J = u^{[k]} = M_k(u^{[k-1]},\ \theta,\ y) = M_k(M_{k-1}(\cdots M_1(x)),\ \theta,\ y)$$

例如一個二分類 MLP 的 loss function 會有 $M_1 = \text{MM}_{W^{[1]},b^{[1]}}$，$M_2 = \sigma$ ... $M_{k-1} = \text{MM}_{W^{[r]},b^{[r]}}$，$M_k = l_{\text{logistic}}$。

反向傳播分成兩個步驟：forward pass 與 backward pass。forward pass 做的事情僅僅是讓電腦去計算 $u^{[i]}$ 並記憶起來。backward pass 則是 backpropagation 的重點：利用連鎖率計算梯度。

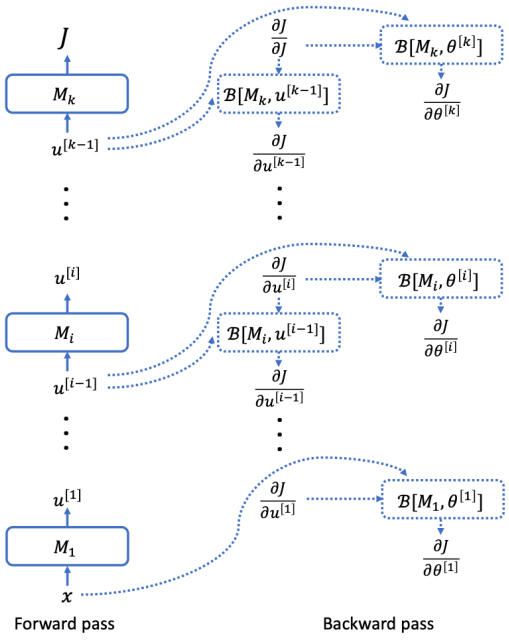
以最後一層（第 k 層）到第一層的順序計算，連鎖率確保我們每次都能得到：

$$\frac{\partial J}{\partial u^{[i-1]}} = \mathcal{B}[M_i,\ u^{[i-1]}]\left(\frac{\partial J}{\partial u^{[i]}}\right)$$

因此，對參數的梯度計算也可以從最後一層一直延續到第一層，因為我們總是能得到上一層的 $\partial J/\partial u^{[i-1]}$：

$$\frac{\partial J}{\partial \theta^{[i]}} = \mathcal{B}[M_i,\ \theta^{[i]}]\left(\frac{\partial J}{\partial u^{[i]}}\right)$$

連鎖率確保我們能夠一步步推回上一層的資訊。



圖三：反向傳播

接著就能進行梯度下降，開始訓練神經網路。