[[python/numpy tutorial]](http://link.zhihu.com/?target=http%3A//cs231n.github.io/python-numpy-tutorial)

Python語法的相關教學 大致跟普通程式語言相同 但有很強的library

可以支援多維度的計算(numpy)，大量模組化可以用簡單的語法撰寫

[[image classification notes]](http://link.zhihu.com/?target=http%3A//cs231n.github.io/classification)

人類能輕易地分辨出物體，然而電腦會因為很多其他因素，進而誤判物體的類別，假如我們有一些規則能訓練電腦判別(machine learning)，就能有辦法讓電腦認得input比較傾向於何種類型，而這些規則能粗糙的稱作演算法，不在是由人類的觀點出發，而是從每個影像的特徵做出分類。

流程大概是 輸入 -> 學習 -> 評價 ， 一般分為”訓練集” “測試集”

接下來介紹分類器

**Nearest Neighbor**

 **L1: 單純將測試的圖片與訓練集的圖片做矩陣相減**



**L2：將L1的相差距離計算出來，注意到因**

**為這個eq是取平方，所以對錯誤時的準確懲罰會很高，所以理論上準確率會低於L1**

K-Nearest Neighbor

<https://zh.wikipedia.org/wiki/%E6%9C%80%E8%BF%91%E9%84%B0%E5%B1%85%E6%B3%95>

這個方法就是找K個最相近的點來判斷，我應該是屬於哪種類別，所以K值的決定很重要，想當然，大家一定會想說那我取K趨近於能將所有測試分類越好，但是可能會發生over fitting的現象，這個現象指的就是你在訓練完後很難做預測，你的預測太過於緊密，導致函數不停地改變方向，也就是講義所說明的很難generalization (泛化)

Approximate Nearest Neighbor

由於上述要訓練的方法其實是趨近於暴力法才能解出來，在我看來有點像NP的問題，所以需要類似Approximate來協助我們預處理，講義有提及包括 k-d 樹(有點類似decision tree) k-means(下方)

K-means

假設預期的分群群聚數目為c，則根據上述觀察，我們可以經由下列步驟來進行k-means 分群法：

1. 隨機選取c 個資料點，將之分別視為c 個群聚的群中心，這就是Y。
2. 由固定的Y，產生最佳的U。
3. 計算目標函數J(X; Y, U)，如果保持不變，代表分群結果已經穩定不變，所以可以結束此疊代方法。
4. 再由固定的U，產生最佳的Y。

KD tree

相較傳統KNN的檢索，能較快搜尋到需要的data，因為每次是一分為二(以2D KD tree為例)也就是會加快至O(logN)，有點類似binary search(先決條件是需要排序好)本身不管影像之類的都會有先後順序，所以非常適合

[[linear classification notes]](http://link.zhihu.com/?target=http%3A//cs231n.github.io/linear-classify)

Linear classification

由於前面的方法所需要的不管是計算或者是暫存都需要大量的時間或空間，所以這裡自然延伸到nerve network ，包括score function 和 loss function

Score function



其中 W 被稱作 weight

b 被稱作 bias (?)

W相當於是將該圖片對每個類別做domain轉換的評分，也就是從高維度轉到低維度，假設要判斷是否是船，那藍色的weight就應該是正的，其他顏色就不該比藍色權重高(這是很粗糙的假設)，回想舉證的乘法，row \* col，換句話說W的col可以視為該類別對該圖片的評分

Loss function

SVM



其中\Delta是hyper parameter (可以tuning的值)

而這個eq想表達的是SVM的loss function想要正确分類類別y_i的分數比不正確類别分數高，而且至少要高\Delta

這種多類的SVM的中心思想應該是，希望在做所有比較後，相對較高分數的類別是正確的，而非絕對準確的類別歸屬，max的用意在於製造類似人類神經中的閾值，也就是必須超過某一個數值後，神經才開始會往前傳遞訊息，而這裡的閾值設定為0

Regulation penalty(通常使用 L2 penalty)

有可能有不同的W，作上述的Loss function卻得到相同的loss值，可是卻只偏向某些特徵(意即只偏重W的某些元素，造成over fitting現象(?))

SoftMax



將前面score function算出的值，經過softmax function，能將domain限制在 0~1之間，就能用機率的model與概念去解釋，這張圖片分屬各個類別的可能性，先用exp用來將所有數值轉為大於0的domain，在用normalize的技巧，將所有的值總合為1，接著用 –log 轉回該數值屬於的domain (但已經normalize)

SVM與SoftMax的差別

我覺得是SVM在意的是比較粗糙的一面，只要我的正確分類特別突出，並且超越變數\Delta，那我就能用max的函數達到loss value為0，但是在SoftMax卻是著重在所有類別可能性的排序，先後順序可能兩者都相同，但SoftMax表現出來可能會是某幾類類別可能性都蠻高，進而影響對W的信心

[[optimization notes]](http://link.zhihu.com/?target=http%3A//cs231n.github.io/optimization-1)

也就是尋找到最佳的W使的最後的Loss value minimize (並非最優化凸函數而是神經網路最優化)

1. 暴力法：將所有可能性嘗試過，記住loss value 最小的W
2. 隨機搜尋法：從一個隨機的W開始，並且生成一個隨機的值a，W+aW能使loss value降低的話，則更新W
3. 跟隨梯度法

：我們可以利用微積分的微分取得斜率最低的(向量這裡稱作gradient梯度)，就不在像是第二個方法盲目的猜測，而是跟著loss value而下降

第三個方法看起來是最好的，但是可能需要一些basic calculus concept，所以在實作上可能比前兩者困難，而這個方法在講義中又分成兩種 數值梯度法

梯度分析法

數值分析法使用微分定義

然而在一些分析上發現 這個表示式，大致上的原因是在估計上能更加準確estimate error 與 h^2 成正比，以及不包括first-order errors(?)

 另一種梯度分析法不再是利用上述的完整定義，而是利用微分的公式，加速運算，缺點是容易發生錯誤，所以實作方面可以兩個都算，一個是實驗組另一個是對照組，然而矩陣的微分我不是很懂意義何在，依照公式定義來說，是將W\*X後對X作微分，也就是結果會是1\*X^(1-1)\*W -> W，這跟講義最後的結果類似

有種加速的方法是**Mini-batch gradient descent，因為訓練集中的很多數據是有相關的，也就是假設有100萬張圖片的平均值會跟1000張圖片的平均值相近，那就不必訓練那麼多的圖片(這是一個粗糙的假設)**

[[backprop notes]](http://link.zhihu.com/?target=http%3A//cs231n.github.io/optimization-2)

概念是利用微積分中的偏微分，計算原本前向傳播，算完之後利用偏微分將所有的梯度(斜率)反向計算出來，進而能對輸出值進行調控

Sigmoid active function



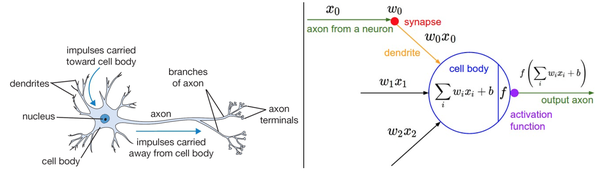
作用在於將函數值壓縮在0~1之間，有利於套用機率的model來做分析，在反向傳播上的微分也能用很簡單的數學式表示



矩陣、向量的反向傳輸，需要用到transpose的技巧，維度也需要這樣推測，假如有一個矩陣 D=X\*W X(3\*10) W(10\*5) D(3\*5)

結果dD為(3\*5) 想要反向得到的W為(10\*5) ，那麼需要X^T \* dD

[Neural Nets notes 1](http://link.zhihu.com/?target=http%3A//cs231n.github.io/neural-networks-1/)



神經網路的模型改成類似這樣，這是兩層的表示式，第一層在轉換時是”非線性”函數，第二層是普通矩陣相乘，當然會想說為何要要用非線性的函數當作第一層，原因在於如果使用非線性的話，那麼將第一層與第二層的矩陣相乘，如此一來就變成單層神經網路(多層也一樣)，無法做出更多的變化與學習



大致上的概念，都是模仿神經元的運作，**activation function就成了整個神經元最後的輸出，通常這個函數會使用**sigmoid function，可以將數值壓縮在0~1之間，前面有提過就不在贅述

當然人類的神經絕對沒這麼簡單，這是個非常粗糙的model，比如說W1x也可以是非線性函數，沒有說一定要用內積的方式呈現

Activation function介紹

Sigmoid ：前面已介紹過，優點是將數值壓縮在0~1之間，更能在生物神經上獲得相對應的解釋，1是激活，0是非激活，缺點包括 1. 由於在0或1時，梯度接近0，所以在反向傳播時，會將所有梯度歸0，所以無法再傳回數據 2. 並非是0中心，權重更新顯示為Z字形(?我想原因可能是你原本只要到達Z上面那個值就好，可是卻多了下面那一段的時間)

Tanh ：為放大版的sigmoid，將數值壓縮在-1~1之間，解決非0中心的問題

ReLU ：

簡單利用閾值的概念，模仿人類神經的函數，優點是計算簡單，且收斂速度比Tanh快，缺點是因為太簡單，所以很容易在訓練時就發生死亡的現象(無法被激活)，導致反向傳播時梯度皆為0

Leaky ReLU：閾值的值不在假設為0，而改為一個很接近0的數字，可以比上面的激活函數，死亡率下降，但還是不太穩定，也有人提出說讓每一個神經元的閾值不同，但這樣hyper parameter就變很多需要調控，增加實作困難

Maxout：

保留ReLU的優點，解決ReLU的缺點太容易死亡，唯一的缺點是hyper parameter激增，當你想要快速訓練時，可能會花太多時間在於參數的調控

[Neural Nets notes 2](http://link.zhihu.com/?target=http%3A//cs231n.github.io/neural-networks-2/)

數據預處理：**Original ->Mean subtraction -> Normalization**

**減去平均值，讓數據0中心化，接著再使用一般化技巧，讓數值介於-1~1之間**

**另一種預處理：Original ->Mean subtraction -> PCA->Whitening**

**其中PCA是指將矩陣降階，然後找出特徵矩陣，Whiten是將特徵矩陣上的特徵值做Normalization**

**權重初始化：**

1. **錯誤的方式(zero filled)：將導致數據無法更新**
2. **先隨機給一個小數字，接著在訓練完後，可能造成數據很分散，需要使用1/sqrt(n)的方式做校準，這跟高中學習線性回歸很像**
3. **Sparse initialization，看不太懂，為何先設成將所有W設為0，再隨機找幾個矩陣連接(利用高斯分布(常態分布)生成)**
4. **Batch Normalization，在數據進入學習前，先行進入另一個網路，將其轉換成常態分佈**

正歸化(防止over fitting)：

1. L2：讓整個W不會偏重於其中幾個feature，而讓Regulation penalty變重(向對於L1來說)，向函數增加
2. L1：Regulation penalty沒那麼嚴重，向函數增加
3. **L1+L2線性組合**
4. Max norm constraints：要求權重向量W的值必須小於一個數值C，C通常設置為3、4
5. **Dropout：以超過P為基準，來判斷是否要設為0，或者激活它**

[Neural Nets notes 3](https://link.zhihu.com/?target=http%3A//cs231n.github.io/neural-networks-3/)

**教學在training model所需要注意的部分以及提示(在實作時可以參考)**

[ConvNet notes](https://link.zhihu.com/?target=http%3A//cs231n.github.io/convolutional-networks/)

ConvNet大致與神經網路的精神類似，但採取三維方面思考，分別是長度 寬度 深度(指的是在神經網路中的層數)

大致分為 Conv layer -> ReLU layer ->Pooling layer -> full connected layer

**Conv layer**

**與input layer連接，每個神經元的小區域計算出與input的內積，假如有n個濾波器(receptive field)，那麼最後會有[A\*B\*n]的維度(這裡我想應該可以利用multithread的方式，平行加速運算，因為互相之間不影響結果，所以部會有data dependency)**

**ReLU layer**

**將每個元素經過激活函數，這裡就如其名使用ReLU函數**

**Pooling layer**

**這裡就是取樣的layer，利用downsampling的操作，將input做降維度的工作**

**Full connected layer**

**這裡是最後計算出評價的評價函數**