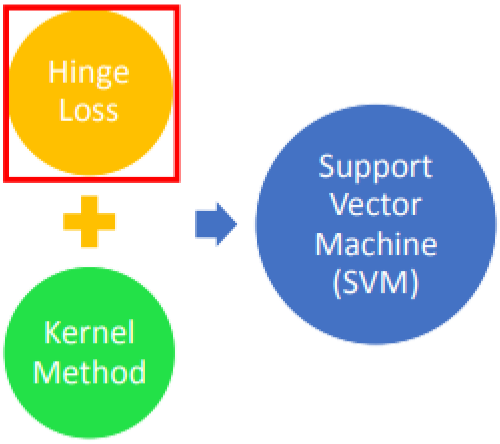
INDEX

Support Vector Machine

SVM 主要有兩個特色：Hinge Loss 與 Kernel Trick。



Binary Classification

處理 Binary Classification 的問題時，第一個 step 是假設一個 function g(x)，這個 g(x)裡面有另外一個 function f(x)。
當 f(x) > 0 時，output 為+1，代表某個 class；當 f(x) < 0，output 為-1，代表另外一個 class。

Step 1: Function set (Model)

$$g(x) = \begin{cases} f(x) > 0 & \text{Output = +1} \\ f(x) < 0 & \text{Output = -1} \end{cases}$$

由於該例為 supervised problem，因此每一筆 training data 有 label $\hat{y}$（此處用+1 跟-1 表示）。而最理想的 Loss Function 寫作：

$$L(f) = \sum_n \delta(g(x^n \neq \hat{y}^n))$$

其中 δ 表示裡面這件事情如果為真的話，delta function 的 output 就是 1；如果 g 不等於 y，δ 的 output 就是 1。
所以 loss 就變成 g 在 training set 上總共犯了幾次錯誤，因此我們會希望他犯的錯誤是越小越好。

但是在該 task 裡面要做 optimization 是相當困難的，因為其 loss 是不可微分，所以無法用 Gradient Descent 來解它。
因此此處 delta function 需要用其他 loss function 近似之。

一、Square Loss

它的 loss 定法是，希望當$\hat{y}^n = 1$的時候，f(x)跟 1 越接近越好；$\hat{y}^n = -1$時，f(x)跟-1 越接近越好。

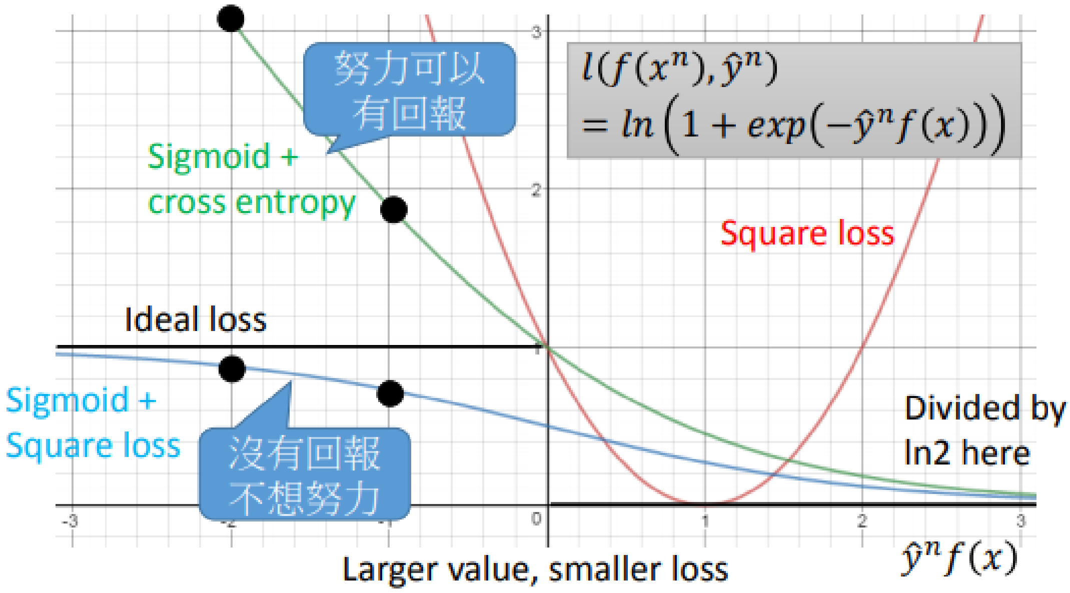
Square Loss: If $\hat{y}^n = 1$, $f(x)$ close to 1
If $\hat{y}^n = -1$, $f(x)$ close to -1

所以 square loss 可以寫成：

$$l(f(x^n), \hat{y}^n) = (\hat{y}^n f(x^n) - 1)^2$$

$$L(f) = \sum_n l(f(x^n), \hat{y}^n)$$

因此把 square loss 的 function 畫出來會呈現下圖紅線趨勢（不合理）。

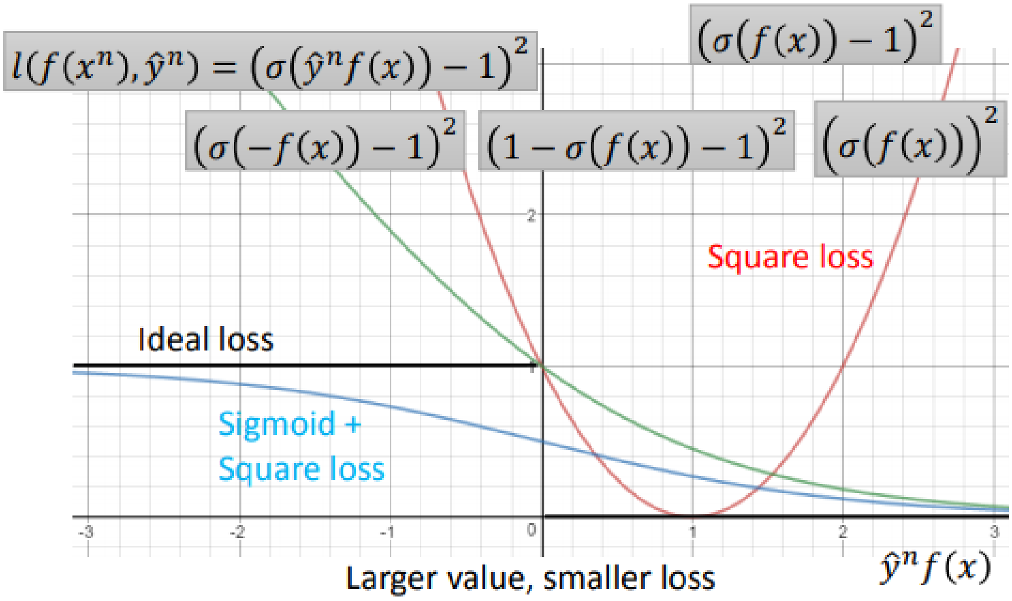


二、Sigmoid + Square Loss

此處 sigmoid function 就用來 σ 表示，我們希望$\hat{y}^n = 1$的時候，$\sigma(f(x))$會趨近於 1；等於-1 的時候，$\sigma(f(x))$會趨近於 0。

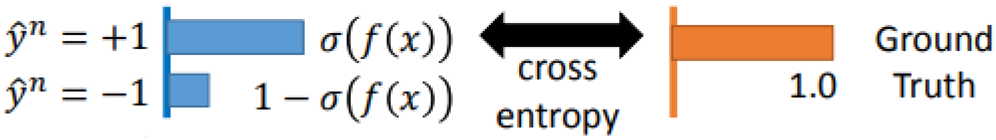
Sigmoid + Square Loss: If $\hat{y}^n = 1$, $\sigma(f(x))$ close to 1
If $\hat{y}^n = -1$, $\sigma(f(x))$ close to 0

其式子就是下圖藍線：



三、Sigmoid + Cross Entropy (Logistic Regression)

然而在做 Logistic Regression 的時候，使用 square loss 作為 loss 表現較為差勁。因此在一般情況下，我們會偏好使用 cross entropy。

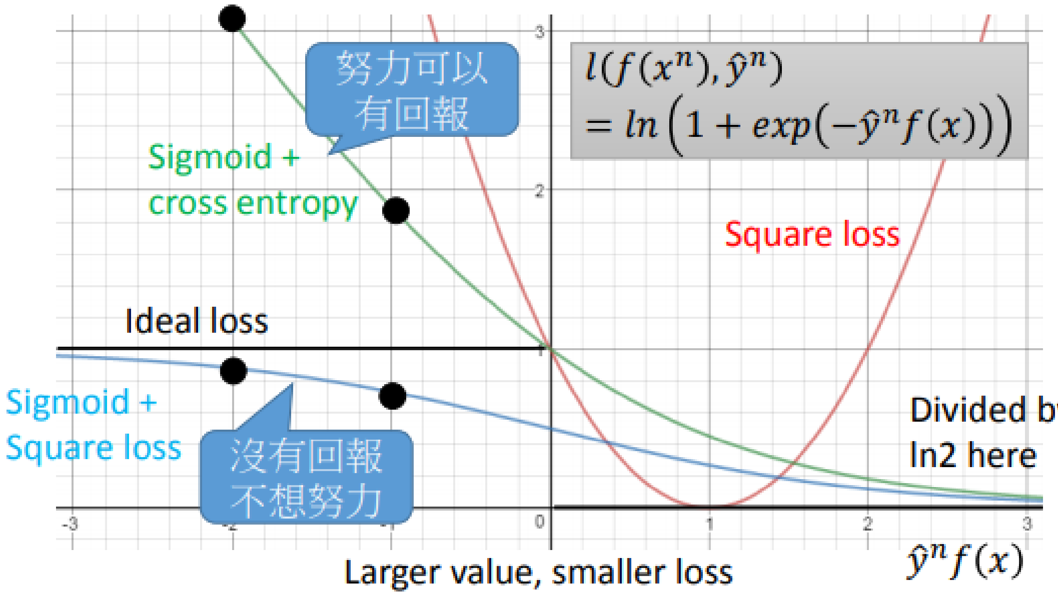


其$\sigma(f(x))$代表了一個 distribution，而 ground truth 是另外一個 distribution。這樣 distribution 之間的 cross entropy，就是需要去 minimize 的 loss。
這個 function 可以寫成：

$$l(f(x^n), \hat{y}^n) = ln(1 + \exp(-\hat{y}^n f(x^n)))$$

當$\hat{y}^n f(x^n)$趨近無窮大的時候，exponential 負的無窮大是 0，因此變成 ln(1+0)得到結果是 0；若$\hat{y}^n f(x^n)$為很大的負值的時候，得到結果還是無窮大。

其式子為下圖綠線。此處另除上一個 ln 2，讓它變成 ideal loss 的 upper bound。因此雖然沒有辦法 minimize ideal loss（黑線），但是可以去 minimize 它的 upper bound，就可以順便達到該目的。



另外比較 square error 與 cross entropy loss function，會發現假設把$\hat{y}^n f(x^n)$從-2 移到-1 的時候，前者的變化很小，但在後者的變化就非常大。

所以對 sigmoid function 來說，在這種極端的 case 下，其值為很大的負值時，照理說應該要有很大的 gradient，但是在 square error 中並不是如此。
在$\hat{y}^n f(x^n)$非常 negative 的時候，調整其值對最後 total loss 影響不大。
所以就算調整了 negative 的值，也沒有辦法得到太多的回報，因此會傾向不想調整那些非常 negative 的值。

然而對 cross entropy 來說，它的努力是可以得到回報的，所以會傾向把原來很 negative 的值往正的地方推。因此在實作上用 cross entropy 會比 square error 還更容易 training。

四、Hinge Loss

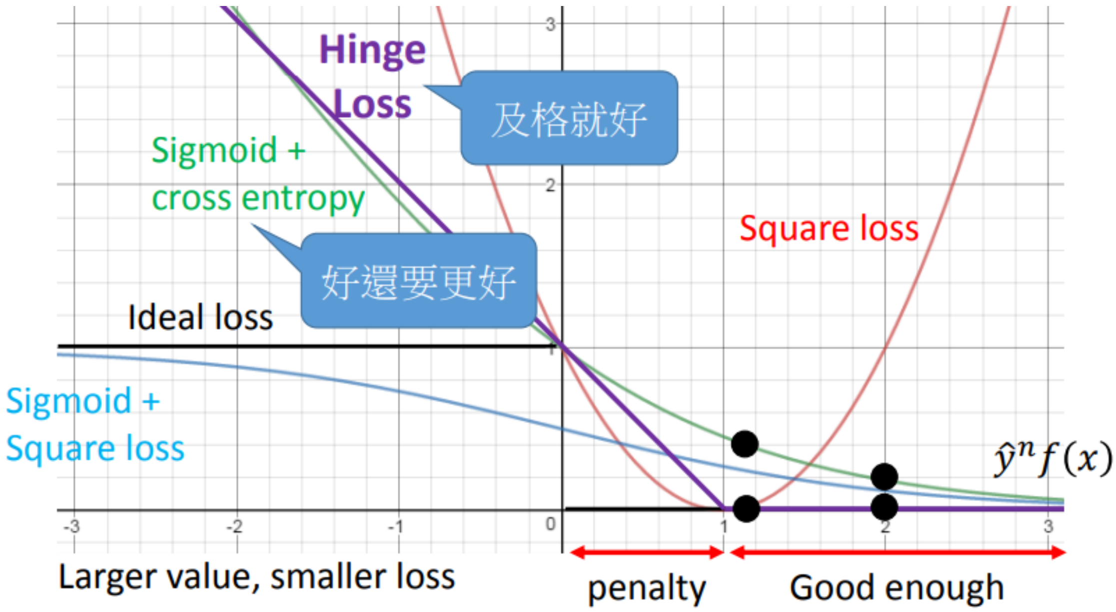
hinge loss 假設該處的 loss function 是 maximum。
當$\hat{y}^n = 1$的時候，loss function 就是 max(0, 1-f(x))，只要 1-f(x) < 0 則 loss function 即為 0；反之$\hat{y}^n = -1$時，loss function 訂為 max(0, 1+f(x))，當 1+f(x) < 0 則 loss function 為 0。

$$l(f(x^n), \hat{y}^n) = max(0, 1 - \hat{y}^n f(x))$$

If $\hat{y}^n = 1$, $max(0, 1 - f(x))$ $1 - f(x) < 0$ $f(x) > 1$

If $\hat{y}^n = -1$, $max(0, 1 + f(x))$ $1 + f(x) < 0$ $f(x) < -1$

所以用 hinge loss 做 training 的時候，對一個 positive 的 example 來說 f(x) > 1 就是完美的 case；對 negative example 來說，則希望 f(x) < -1。
將 hinge loss 作圖後將呈圖中紫線的趨勢。



在這條線上只要$\hat{y}^n f(x^n) > 1$的時候就已經夠好了，其 loss 已經為 0 再更大都沒有幫助；但如果$1 > \hat{y}^n f(x^n) > 0$，則 machine 在做 classification 的時候他已經可以得到正確的答案，但是對 hinge loss 來說這樣還不夠好，它必須比正確的答案還要好過一段距離（margin）。
也就是說當$\hat{y}^n f(x^n)$還沒有大於 1 的時候，仍會有 penalty 存在，促使 machine 讓$\hat{y}^n f(x^n) > 0$

另外假設 hinge loss 的基準點為 1 的情況在於，使其變成 ideal loss 的 upper bound。因此只要 minimize hinge loss，就可能可以得到 minimize ideal loss function 的效果。

Linear SVM

linear SVM 假設現在的 function 為 linear，而 f(x)寫作下列形式：

$$f(x) = \sum_i w_i x_i + b = \begin{bmatrix} w \\ b \end{bmatrix} \cdot \begin{bmatrix} x \\ 1 \end{bmatrix} = w^T x$$

接著我們把 w 跟 b 串起來的 vector 直接就用新的 w 來表示，代表需要透過 training data 找出來的 model 參數；而 x 跟 1 串起來的 vector 就當作一個新的 feature。

在 SVM 裡採用了 hinge loss 作為 loss function，通常還會另外加上 regularization term。這個 loss function 是一個 convex function（因為 hinge loss 與 L2 regularization 皆是 convex function）。

$$L(f) = \sum_n l(f(x^n), \hat{y}^n) + \lambda \|w\|_2$$

$$l(f(x^n), \hat{y}^n) = max(0, 1 - \hat{y}^n f(x))$$

所以 gradient descent 來說，convex function 很容易進行 optimization。因為不管從哪個地方做 initialization，最後找出來的結果都會是一樣的。

另外如果比較 Logistic Regression 跟 Linear SVM 的差別，唯一不同的地方在於怎麼定 Loss Function。若用 Hinge Loss 就是 Linear SVM；用 Cross Entropy 就是 Logistic Regression。因此在做 Deep Learning 的時候，如果不是用 Cross Entropy 當作 Loss Function 而是用 Hinge Loss 的話，其實就是 deep 版本的 SVM。

一、Gradient Descent

只要能夠對 model 裡的 loss 做 weight w_i 的偏微分，就能使用 Gradient Descent。首先 SVM 的 Loss function 寫作下列形式：

$$L(f) = \sum_n l(f(x^n), \hat{y}^n) \qquad l(f(x^n), \hat{y}^n) = max(0, 1 - \hat{y}^n f(x^n))$$

接著對 Loss function 做 w_i 的偏微分，如下式：

$$\frac{\partial l(f(x^n), \hat{y}^n)}{\partial w_i} = \frac{\partial l(f(x^n), \hat{y}^n)}{\partial f(x^n)} \frac{\partial f(x^n)}{\partial w_i}$$

由於$f(x^n)$就是一個 linear fnction，為兩個 vector 的 Inner Product，因此如果用 w_i 對$f(x^n)$做偏微分的話，得到的其實就是 x^n 的第 i 個 dimension 的值。

$$\frac{\partial f(x^n)}{\partial w_i} = x_i^n$$

接著前面對 Hinge Loss 做偏微分的解為：

$$\frac{\partial max\left(0, 1 - \hat{y}^n f(x^n)\right)}{\partial f(x^n)} = \begin{cases} -\hat{y}^n & \text{If } \hat{y}^n f(x^n) < 1 \\ 0 & \text{otherwise} \end{cases}$$

由於它有兩個 operation region，因此可以 operate 在 0 是 maximum 的 case，也可以 operate 在$1 - \hat{y}^n f(x^n)$是 maximum 的 case。而決定何時 operate 在哪一個 case 則 depend 現在的 model w^T的值是多少。

假設$\hat{y}^n f(x^n) < 1$則作用在下圖紫色斜線的 region，因此對$f(x^n)$做偏微分得到的值為$-\hat{y}^n$；若作用於其他 region，得到的偏微分值皆為 0。

$$\frac{\partial max\left(0, 1 - \hat{y}^n f(x^n)\right)}{\partial f(x^n)} = \begin{cases} -\hat{y}^n & \text{If } \hat{y}^n f(x^n) < 1 \\ 0 & \text{otherwise} \end{cases}$$

綜上所述，對 L(f)做偏微分以後得到的值為：

$$\frac{\partial L(f)}{\partial w_i} = \sum_n \underbrace{\underline{-\delta(\hat{y}^n f(x^n) < 1)\hat{y}^n} x_i^n}_{c^n(w)} \quad w_i \leftarrow w_i - \eta \sum_n c^n(w) x_i^n$$

亦即加總所有的 Training Data 之後，再看每一筆的$\hat{y}^n f(x^n)$是不是小於 1。

由於紅底線部分取決於現在的參數是什麼，因此可以把它寫作 $c^n(w)$。

Another Formulation

如果把 Hinge Loss 換成ε^n來取代它的話（$\varepsilon^n = \max\left(0, 1 - \hat{y}^n f(x^n)\right)$），同樣目標是要 minimize Total Loss。

因此從定義上來看，我們會取 0 跟$1 - \hat{y}^n f(x^n)$裡面取大的那一個當作ε^n，

在 minimize Total Loss 的情形下，等同於下列表示法：

$$\boxed{\begin{array}{l} \varepsilon^n \geq 0 \\ \varepsilon^n \geq 1 - \hat{y}^n f(x) \Rightarrow \hat{y}^n f(x) \geq 1 - \varepsilon^n \end{array}}$$

由於現在要去 minimize L(f)，因此只能選一個最小的ε^n讓 L(f)能夠最小，並同時符合上圖紅框的 constrain。所以最終ε^n必須等於 0 跟$1 - \hat{y}^n f(x^n)$裡面最大的一項。

該表示法即為一般常見的 SVM 型態。希望$\hat{y}^n f(x^n)$為同號的同時，必須另外滿足 margin 1。
然而在某些情況下無法滿足該 margin，因此會再放寬其條件，只須滿足減去一個 slack variable ε^n（必須為正值）之後的 margin 即可。
而上圖紅框則為 Quadradic programming problem，因此除 Gradient Descent 外亦可用 Quadratic Programming 的 solver 解之。

Kernel Method

實際上我們找出來可以 minimize Loss Function 的 weight w*，其實是 data 的 linear combination，寫作總和所有 training data 的 vector point x^n，並都乘上一個 weight α_n^*。

$$w^* = \sum_n \alpha_n^* x^n$$

Linear combination of data points

也就是說找出來的 model 其實 data point 的 Linear Combination。
若用 Gradient Descent 的方法說明之則為：

$$\left.\begin{aligned} w_1 &\leftarrow w_1 - \eta \sum_n c^n(w) x_1^n \\ &\vdots \\ w_i &\leftarrow w_i - \eta \sum_n c^n(w) x_i^n \\ &\vdots \\ w_k &\leftarrow w_k - \eta \sum_n c^n(w) x_k^n \end{aligned}\right\}$$

If w initialized as **0**

$$w \leftarrow w - \eta \sum_n c^n(w) x^n$$

$$c^n(w) = \frac{\partial l(f(x^n), \hat{y}^n)}{\partial f(x^n)}$$

Hinge loss: usually zero

c.f. for logistic regression, it is always non-zero

假設現在 w 有 k 維且 update 的式子都是一樣的，唯一不同的地方在於最後乘上去的 value x^n。
若把 w_1 到 w_k 串成一個 vector；x_1^n到x_i^n也串成一個 vector，得到的結果就是每次 update w 的時候，都是把 w 減掉 learning rate 乘上 x^n 的總和再乘上一個 weight。

假設在 initialize 的時候 w 是一個 zero vector，每次在 update w 時，都是加上 data points 的 Linear Combination，而最後得到的 solution 就是用 Gradient Descent 解出來的 w。

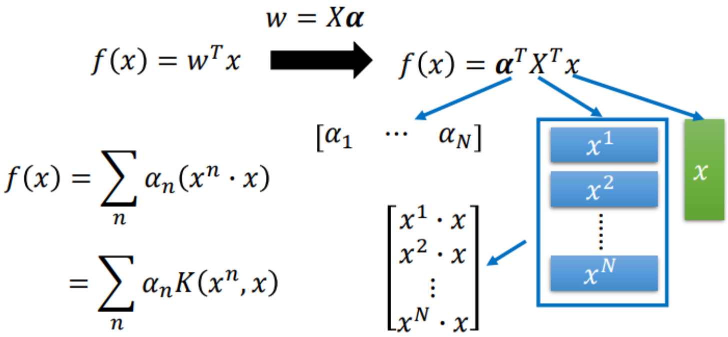
而 $c^n(w)$代表的是 f 對 Loss Function 的偏微分。此處選用 Hinge Loss 作為 Loss Function，因此該項往往都是 0（作用在 max = 0 的 region）。

故所以最後解出來的 w*，它的 Linear Combination weight 可能會是 sparse 的。意即可能有很多的 data point 它對應的 α*值等於 0，而其他 α*值不等於 0 的 x^n 就被稱作 support vector。這也是為何 SVM 相較於其他方法可能比較 robust 的原因。

而把 w 寫成 data point 的 Linear Combination 最大的好處在於可以使用 kernel trick。
我們已經知道 w 就是 data point 的 Linear Combination，因此可以把所有 data point x^1 到 x^N 排成一個 matrix X，而 α^1 到 α^N 就是一個 vector。
兩兩相乘後就會得到 X column 的 Linear Combination w。

$$w = \sum_n \alpha_n x^n = X\boldsymbol{\alpha} \qquad X = \boxed{\begin{matrix} x^1 & x^2 & \cdots\cdots & x^N \end{matrix}} \qquad \boldsymbol{\alpha} = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_N \end{bmatrix}$$

接著可以藉此改寫 function，本來的 function 寫作$w^T x$，而上述又知$w = X\alpha$，所以$f(x) = \alpha^T X^T x$。



最後得到的結果如下：

$$f(x) = \sum_n \alpha_n (x^n \cdot x)$$

此外由於 α 是 sparse 的，因此計算 inner product 時只需考慮 α 不等於 0 的 vector 就好。
接著會將$x^n \cdot x$這個式子寫作另一個 function $k(x^n, x)$，稱作 Kernel function。因此此處的問題就變成找一組最好的α_n使 total loss 最小。
在原本的 total loss 中，有兩個 input 分別為$\hat{y}^n$與$f(x^n)$，而後者就可以使用上式替換之，記為：

$$L(f) = \sum_n l(\underline{f(x^n)}, \hat{y}^n) = \sum_n l\left(\underline{\sum_{n'} \alpha_{n'} K(x^{n'}, x^n)}, \hat{y}^n\right)$$

所以我們不再需要真的知道 vector x 是多少，只需要得知它與另一個 vector 的內積即可（Kernel function）。

一、Kernel Trick

假設有一筆二維的 data x，對它進行 Feature Transform 以後其結果為ϕ(x)。同理對 z 做完 Feature Transform 之後，x 與 z 的 Kernel function 計算如下：

$$K(x,z) = \phi(x)\cdot\phi(z) = \begin{bmatrix} {x_1}^2 \\ \sqrt{2}x_1x_2 \\ {x_2}^2 \end{bmatrix} \cdot \begin{bmatrix} {z_1}^2 \\ \sqrt{2}z_1z_2 \\ {z_2}^2 \end{bmatrix}$$

$$= {x_1}^2{z_1}^2 + 2x_1x_2z_1z_2 + {x_2}^2{z_2}^2$$

$$= (x_1z_1 + x_2z_2)^2 = \left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \cdot \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}\right)^2$$

$$= (x\cdot z)^2$$

故知若 x 與 z 為 k 維的 vector，將其投影到一個更高維的平面，該空間會考慮所有 feature 倆倆之間的關係，投影後的 vector 就記作ϕ(x)與ϕ(z)。

$$x = \begin{bmatrix} x_1 \\ \vdots \\ x_k \end{bmatrix} \quad z = \begin{bmatrix} z_1 \\ \vdots \\ z_k \end{bmatrix} \qquad \phi(x) = \begin{bmatrix} {x_1}^2 \\ \vdots \\ {x_k}^2 \\ \sqrt{2}x_1x_2 \\ \sqrt{2}x_1x_3 \\ \vdots \\ \sqrt{2}x_2x_3 \\ \vdots \end{bmatrix}$$

因此使用 Kernel Trick 可以直接算出 Feature Transform 之後，ϕ(x)與ϕ(z)的內積值，減少運算量。

二、Radial Basis Function Kernel (RBF Kernel)

該方法旨在處理無窮多維的平面上運算 Kernel Trick 的問題，即ϕ()為無限多維的 vector。然而在該情況容易出現 overfitting 的情形，因此使用時須多注意。

$$K(x,z) = exp\left(-\frac{1}{2}\|x-z\|_2\right) = \phi(x)\cdot\phi(z)?$$

$$= exp\left(-\frac{1}{2}\|x\|_2 - \frac{1}{2}\|z\|_2 + x\cdot z\right)$$

$\phi(*)$ has inf dim!!!

$$= exp\left(-\frac{1}{2}\|x\|_2\right) exp\left(-\frac{1}{2}\|z\|_2\right) exp(x\cdot z) \quad = C_xC_z exp(x\cdot z)$$

$$= C_xC_z \sum_{i=0}^{\infty} \frac{(x\cdot z)^i}{i!} = C_xC_z + C_xC_z(x\cdot z) + C_xC_z\frac{1}{2}(x\cdot z)^2 \cdots$$

$$[C_x]\cdot[C_z] \qquad \begin{bmatrix} C_xx_1 \\ C_xx_2 \\ \vdots \end{bmatrix} \cdot \begin{bmatrix} C_zz_1 \\ C_zz_2 \\ \vdots \end{bmatrix} \qquad \frac{1}{\sqrt{2}}\begin{bmatrix} C_x{x_1}^2 \\ \vdots \\ \sqrt{2}C_xx_1x_2 \\ \vdots \end{bmatrix} \cdot \frac{1}{\sqrt{2}}\begin{bmatrix} C_z{z_1}^2 \\ \vdots \\ \sqrt{2}C_zz_1z_2 \\ \vdots \end{bmatrix}$$

三、Sigmoid Kernel

Sigmoid Kernel 式子為 K(x, z) = tanh(x ‧ z)。當要把 x 拿來做 testing，帶到 f 裡面的時候，其實是計算 x 與所有 training data 裡面 x^n 的 Kernel function，然後再乘上 α_n。

如果用的是 Sigmoid Kernel 的話，就是把 data 裡面所有的 x^n 跟 x 做內積，套上 Hyperbolic Tangent 後再乘 α_n，最後總和之後的結果。

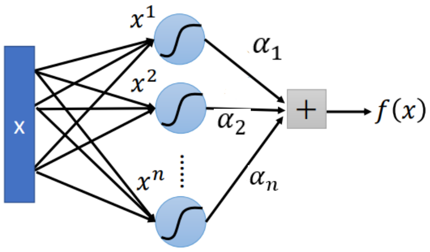
$$f(x) = \sum_n \alpha_n K(x^n, x) = \sum_n \alpha_n \tanh\left(x^n \cdot x\right)$$

而這個 f(x)就可以想成其實是一個只有一層 hidden layer 的 Neural Network，把 x 拿進來之後會跟所有 x^n 做內積，再通過 Hyperbolic Tangent (activation function)，乘上 α_n 後取其總和。

因此問題變為，找出這些 α_n 把它 Weighted Sum 起來可以得到最後的 f(x)。

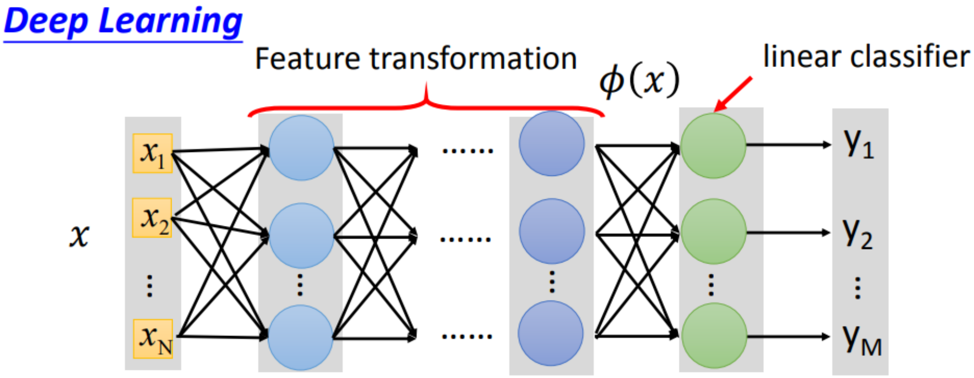
The weight of each neuron is a data point

The number of support vectors is the number of neurons.



四、Deep Learning VS. SVM

在 Deep Learning 中，前幾層 layer 可以看作是 Feature Transformation，而最後一層 layer 則是 Linear Classifier。



SVM 做的也是很類似的事情。

它前面先 apply 一個 Kernel Function，把 feature 轉到 high dimension 上後，就可以 apply Linear Classifier。而在 SVM 裡面 Linear Classifier 一般都會用 Hinge Loss。另外，事實上 SVM 的 kernel 是 learnable，因此可做到把不同 kernel combine 起來，他們中間的 weight 是可以 learn 的。