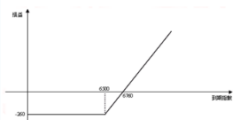
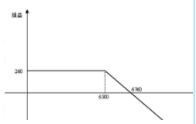
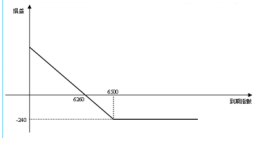
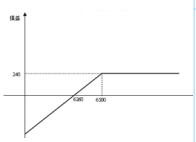
買權(Call)

情況	買進	賣出
	預期大漲時	預期小跌時
圖形	買進一口4月到期，履約價為6500，權利金為260點的買權	賣出一口4月到期，履約價為6500，權利金為260點的買權

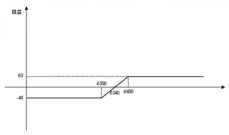
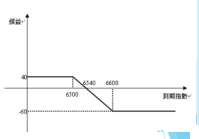
賣權(Put)

情況	買進	賣出
	預期大跌時	預期小漲時
圖形	買進一口4月到期，履約價為6500，權利金為240點的賣權	賣出一口4月到期，履約價為6500，權利金為240點的賣權

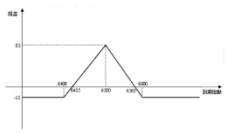
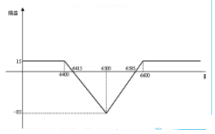
價差(Spread)

情況	多頭 預期小漲， 但僅願承擔有限風險	空頭 預期小跌， 但想獲取權利金收入
圖形	1. 買進一口履約價為6500，權利金為260點的買權 2. 賣出一口履約價為6600，權利金為220點的買權	1. 買進一口履約價為6600，權利金為220點的買權 2. 賣出一口履約價為6500，權利金為260點的買權

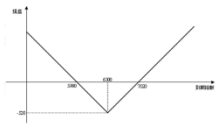
蝶式價差(Butterfly Spread)

情況	買進 預期市場盤整時 (風險有限)	賣出 預期標的物小漲或小跌時 (獲利有限)
圖形	1. 買進一口履約價為6400的賣權 2. 買進一口履約價為6600的賣權 3. 賣出二口履約價為6500的賣權	1. 賣出一口履約價為6600的賣權 2. 賣出一口履約價為6400的賣權 3. 買進二口履約價為6500的賣權

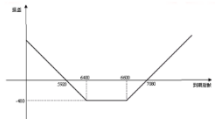
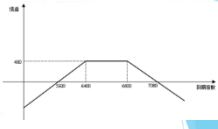
跨式(Straddle)

情況	買進 預期標的物大漲或大跌時	賣出 預期市場盤整時
圖形	1. 買進一口履約價為6500，權利金為260點的買權 2. 買進一口履約價為6500，權利金為260點的賣權	1. 賣出一口履約價為6500，權利金為260點的買權 2. 賣出一口履約價為6500，權利金為260點的賣權



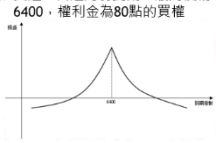
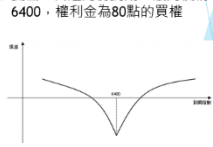

勒式(Strangle)

情況	買進 預期標的物大漲或大跌時	賣出 預期市場盤整時
圖形	1. 買進一口履約價為6600，權利金為240點的買權 2. 買進一口履約價為6400，權利金為240點的賣權	1. 賣出一口履約價為6600，權利金為240點的買權 2. 賣出一口履約價為6400，權利金為240點的賣權

時間價差(Time Spread)

情況	買進 預期市場盤整時	賣出 預期標的物小漲或小跌時 (獲利有限)
圖形	1. 賣出一口近月份到期，履約價為6400，權利金為50點的買權 2. 買進一口遠月份到期，履約價為6400，權利金為80點的買權	1. 買進一口近月份到期，履約價為6400，權利金為50點的買權 2. 賣出一口遠月份到期，履約價為6400，權利金為80點的買權

買賣平權公式 $Call + Xe^{-rT} = Put + S$

Black-Scholes Formulas

Call: $c = S_0N(d_1) - Ke^{-rTN}(d_2)$; Put: $p = Ke^{-rTN}(d_2) - S_0N(-d_1)$

$$d_1 = \frac{\ln\left(\frac{S_0}{K}\right)+\left(r+\frac{\sigma^2}{2}\right)T}{\sigma\sqrt{T}};$$

$$d_2 = \frac{\ln\left(\frac{S_0}{K}\right)+\left(r-\frac{\sigma^2}{2}\right)T}{\sigma\sqrt{T}} = d_1 - \sigma\sqrt{T}.$$

K: the execution price;

r: the risk free rate;

T: the life of an option (in years, usually measured using trading days, ie. 252 days);

σ: the volatility of the stock price per year;

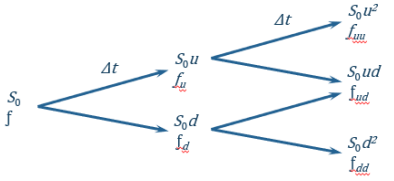
N(x) is the cumulative probability distribution function for a standardized normal distribution;

Monte Carlo Stimulation

$$S(t+\Delta t) = S(t)e^{(\mu-0.5\sigma^2)\Delta t+\sigma\varepsilon\sqrt{\Delta t}} \quad \text{——— (5.7)}$$

```
def MCsim(S, T, r, vol, N, M): # N time slices, M trials
    dt = T/N
    St = np.zeros((M, N+1)) # M row, N+1 col
    St[:, 0] = S # row, col
    for i in range(N):
        St[:,i+1] = np.multiply(St[:,i],np.exp((r-0.5*vol*vol)*dt+vol*np.random.normal(0,1,(M,))*math.sqrt(dt)))
    return St
```

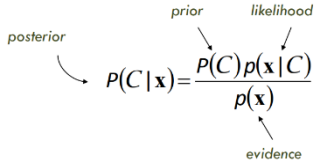
n-step Binomial Model



$$p = \frac{e^{r\Delta t} - d}{u - d}; \quad u = e^{\sigma\sqrt{\Delta t}}; \quad d = e^{-\sigma\sqrt{\Delta t}}$$

```
def bitcall(S0,K,T,r,vol,N):
    dt = T/N
    u = math.exp(vol*math.sqrt(dt))
    d = math.exp(-vol*math.sqrt(dt))
    p = (math.exp(r*dt)-d)/(u-d)
    STree = np.zeros((N+1,N+1))
    PTree = np.zeros((N+1,N+1))

    STree[0][0] = S0
    for c1 in range(N):
        STree[0][c1+1] = STree[0][c1] * u
        for r1 in range(c1+1):
            STree[r1+1][c1+1] = STree[r1][c1]*d

    PTree[0][0] = 1
    for c1 in range(N):
        for r1 in range(c1+1):
            PTree[r1][c1+1] += PTree[r1][c1]*p
            PTree[r1+1][c1+1] += PTree[r1][c1]*(1-p)

    call = 0
    for r1 in range(N+1):
        if(STree[r1][N]>=K):
            call += (STree[r1][N]-K)*PTree[r1][N]

    return call*math.exp(-r*T)
```

Bayes' Rule



choose $\begin{cases} C=1 \text{ if } P(C=1|x_1,x_2) > 0.5 \\ C=0 \text{ otherwise} \end{cases}$

or

choose $\begin{cases} C=1 \text{ if } P(C=1|x_1,x_2) > P(C=0|x_1,x_2) \\ C=0 \text{ otherwise} \end{cases}$

Decision Tree

$$H(S) = -\sum_{i=1}^{N} P_i log_2 P_i$$

```
def entropy(p1, n1):
    if p1==0 and n1==0:
        return 1
    elif p1==0 or n1==0:
        return 0
    else:
        pp = p1/(p1+n1)
        np = n1/(p1+n1)
        return -pp*math.log2(pp) - np*math.log2(np)

#print(entropy(5, 5))
#print(entropy(0, 10))
```

```
def IG(p1, n1, p2, n2):
    pr = p1 + p2
    nr = n1 + n2
    num = pr + nr
    num1 = p1 + n1
    num2 = p2 + n2
    return entropy(pr, nr) - num1/num*entropy(p1, n1)-num2/num*entropy(p2, n2)

#print(IG(3, 4, 6, 1))
#print(IG(6, 2, 3, 3))
```

```
def ID3DTtrain(feature,target):
    node = dict()
    node['data'] = range(len(target))
    tree = []
    tree.append(node)
    t = 0
    while(t<len(tree)):
        idx = tree[t]['data']
        if(sum(target[idx])==0):
            tree[t]['leaf']=1
            tree[t]['decision']=0
        elif(sum(target[idx])==len(idx)):
            tree[t]['leaf']=1
            tree[t]['decision']=1
        else:
            bestIG = 0
            for i in range(feature.shape[1]):
                pool = list(set(feature[idx,i]))
                pool.sort()
                for j in range(len(pool)-1):
                    thres = (pool[j]+pool[j+1])/2
                    G1 = []
                    G2 = []
                    for k in idx:
                        if(feature[k][i]<thres):
                            G1.append(k)
                        else:
                            G2.append(k)
                    p1 = sum(target[G1]==1)
                    n1 = sum(target[G1]==0)
                    p2 = sum(target[G2]==1)
                    n2 = sum(target[G2]==0)
                    thisIG = IG(p1,n1,p2,n2)
                    if(thisIG>bestIG):
                        bestIG = thisIG
                        bestG1 = G1
                        bestG2 = G2
                        bestthres = thres
                        bestf = i
            if(bestIG>0):
                tree[t]['leaf'] = 0
                tree[t]['selectf'] = bestf
                tree[t]['threshold'] = bestthres
                tree[t]['child']=[len(tree),len(tree)+1]
                node = dict()
                node['data'] = bestG1
                tree.append(node)
                node = dict()
                node['data'] = bestG2
                tree.append(node)
            else:
                tree[t]['leaf']=1
                if(sum(target[idx]==1)>sum(target[idx]==0)):
                    tree[t]['decision'] = 1
                else:
                    tree[t]['decision'] = 0
        t+=1
    return tree
```

```
p = 10000 # population
r = 0.01 # survival_rate
m = 1000 # mutation_rate
g = 10 # generation
survive = round(p*r)
pop = np.random.randint(0,2,(p,40))
fit = np.zeros((p,1))
for generation in range(g):
    for i in range(p):
        gene = pop[i,:]
        A = (np.sum(2**np.array(range(10))*ger
        B = (np.sum(2**np.array(range(10))*ger
        C = np.sum(2**np.array(range(10))*gene
        D = (np.sum(2**np.array(range(10))*ger
        fit[i] = E(b2,A2,A,B,C,D)
    sortf = np.argsort(fit[:,0])
    pop = pop[sortf,:]
    for i in range(survive,p):
        fid = np.random.randint(0,survive)
        mid = np.random.randint(0,survive)
        while mid==fid:
            mid = np.random.randint(0,survive)
        mask = np.random.randint(0,2,(1,40))
        son = pop[mid,:].copy()
        father = pop[fid,:]
        son[mask[0,:]==1] = father[mask[0,:]==
        pop[i,:]=son
    for i in range(m):
        mr = np.random.randint(survive,p)
        mc = np.random.randint(0,40)
        pop[mr,mc] = 1-pop[mr,mc]

for i in range(p):
    gene = pop[i,:]
    A = (np.sum(2**np.array(range(10))*gene[:10])-511)/100
    B = (np.sum(2**np.array(range(10))*gene[10:20])-511)/100
    C = np.sum(2**np.array(range(10))*gene[20:30])-511
    D = (np.sum(2**np.array(range(10))*gene[30:40])-511)/100
    fit[i] = E(b2,A2,A,B,C,D)
sortf = np.argsort(fit[:,0])
pop = pop[sortf,:]

gene = pop[0,:]
A = (np.sum(2**np.array(range(10))*gene[:10])-511)/100
B = (np.sum(2**np.array(range(10))*gene[10:20])-511)/100
C = np.sum(2**np.array(range(10))*gene[20:30])-511
D = (np.sum(2**np.array(range(10))*gene[30:40])-511)/100
print('A:',A,' B:',B,' C:',C,' D:',D)
```

線性回歸:自變數因,依變數果, $y=\beta 0+\beta 1x$,$\beta 0$:截距,

$\beta 1$:斜率為x變動一個單位y變動的量,loss function 最小平方法=$\sum_1^n (y-\hat{y})^2$

```
# Regression
n = 1000
# shape (row, col)
b1 = np.zeros((n,1))
A1 = np.zeros((n,5))
for i in range(n):
    # np.random.random() => [0, 1)
    t = np.random.random()*100 # generate t fro
    b1[i] = F1(t)
    A1[i,0] = t**4
    A1[i,1] = t**3
    A1[i,2] = t**2
    A1[i,3] = t
    A1[i,4] = 1
x = np.linalg.lstsq(A1,b1)[0] # A1 x = b1
print(x)
```

Genetic Algorithms

定義基因

- For example, 在我們的問題中使用一組40 bits code來代表4個變數

初代

- 利用亂數或expert knowledge產生一群初始的族群

複製 (reproduction)

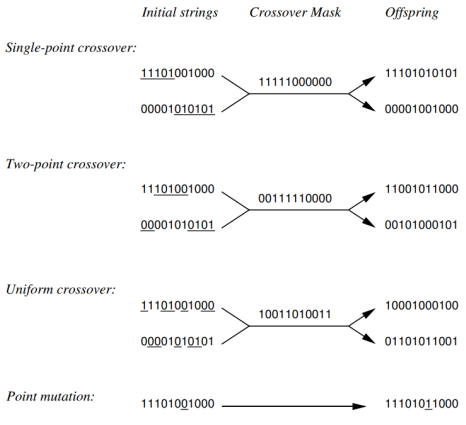
- 計算fitness
- 利用fitness決定適者生存
- 輪盤式選擇 (roulette wheel selection)
  - 依照fitness分割輪盤大小，面積比例越大越容易被選中
- 競爭式選擇 (tournament selection)
  - 只留fitness最高的一小群人survive，淘汰適應不佳的

交配 (crossover)

- 單點交配
  - 此點以後的基因互換
- 雙點交配
  - 兩點間的基因互換
- 遮罩交配
  - 產生一個0/1 mask或filter，mask為1的bit互換

突變

- 少數bit 0->1或1->0



;Delta Hedge:一種投資策略,旨在對沖或中和資產價格變動的風險,特別是期權交易中,德爾塔是一個衡量期權價格變動對期權價值變動的敏感性的指 Delta hedge 通常涉及建立相反的或相關聯的頭寸,以對抗價格波動的影響,以確保整體風險是可控的

(1) ROD :當日有效(Rest of Day)

常見的預設掛單方式，想要讓掛單持續到收盤都有效，就使用ROD。一般來說，身邊的朋友交易最常用到此條件呢～

(2) IOC：立即成交否則取消(Immediate-or-Cancel)

允許部分成交，而沒有成交的部分就取消。通常是市價單指令會搭配IOC。

(3) FOK：全部成交否則取消(Fill-or-Kill)

一定要全部成交，否則就全部取消。

AdaBoost

Input : sequence of N labeled examples $\langle (x_1,y_1),\ldots,(x_N,y_N)\rangle$
Distribution D over N examples
Weak learning algorithm **WeakLearn**
integer T specifying number of iterations

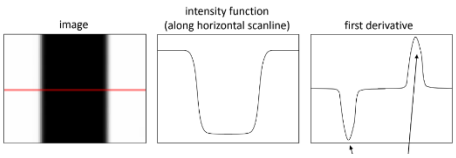
Initialize the weight vector $w_i^1 = D(i)$ for i=1...N

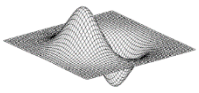
Do for t = 1,2,...,T

1. Set $p^t = \frac{w^t}{\sum_{i=1}^{N} w_i^t}$
2. Call **WeakLearn**, providing it with the distribution p^t; get back a hypothesis h_t: X→[0,1]
3. Calculate the error of h_t: $\varepsilon_t = \sum_{i=1}^{N} p_i^t |h_t(x_i) - y_i|$
4. Set $\beta_t = \varepsilon_t / (1-\varepsilon_t)$
5. Set the new weight s vector to be $w_i^{t+1} = w_i^t \beta_t^{1-|h_t(x_i)-y_i|}$

Output the hypothesis

$$h_f(x) = \begin{cases} 1 & if \sum_{t=1}^{T} (\log\frac{1}{\beta_t}) h_t(x) \geq \frac{1}{2}\sum_{t=1}^{T} \log\frac{1}{\beta_t} \\ 0 & otherwise \end{cases}$$





derivative of Gaussian (x)

$$\frac{\partial}{\partial x} h_\sigma(u,v)$$

The Sobel operator

Common approximation of derivative of Gaussian

$$\nabla f = \left[\frac{\partial f}{\partial x}, 0\right] \quad \nabla f = \left[0, \frac{\partial f}{\partial y}\right] \quad \nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

$$\cos(\theta) + \sin(\theta) =$$

$$\frac{1}{8}\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad s_x \qquad \frac{1}{8}\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \quad s_y$$

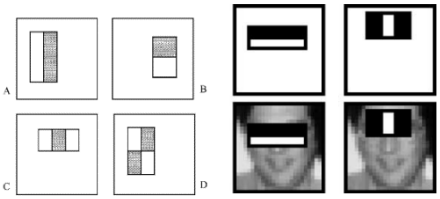
		真實值		總數
		p	*n*	
預測輸出	*p'*	真陽性 (TP)	偽陽性 (FP)	P'
	n'	偽陰性 (FN)	真陰性 (TN)	N'
總數		P	N	

陽性 (P, positive)
陰性 (N, Negative)
真陽性 (TP, true positive)
正確的肯定。又稱：命中 (hit)
真陰性 (TN, true negative)
正確的否定。又稱：正確拒絕 (correct rejection)
偽陽性 (FP, false positive)
錯誤的肯定，又稱：假警報 (false alarm)
偽陰性 (FN, false negative)
錯誤的否定，又稱：未命中 (miss)
真陽性率 (TPR, true positive rate)
又稱：命中率 (hit rate)、敏感度(sensitivity)
TPR = TP / P = TP / (TP+FN)
偽陽性率(FPR, false positive rate)
又稱：錯誤命中率，假警報率 (false alarm rate)
FPR = FP / N = FP / (FP + TN)

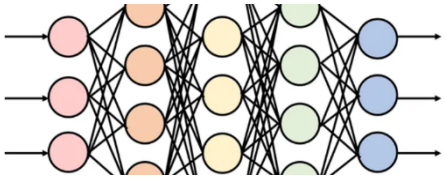
準確度 (ACC, accuracy)
ACC = (TP + TN) / (P + N)
即：(真陽性+真陰性) / 總樣本數
真陰性率 (TNR)
又稱：特異度 (SPC, specificity)
SPC = TN / N = TN / (FP + TN) = 1 - FPR
陽性預測值 (PPV)
PPV = TP / (TP + FP)
陰性預測值 (NPV)
NPV = TN / (TN + FN)
假發現率 (FDR)
FDR = FP / (FP + TP)
F1評分
F1 = 2TP/(P+P')

Robust Real-Time Face Detection

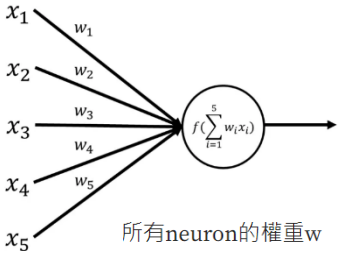
24 × 24, the exhaustive set of rectangle features



邊緣檢測是一種在數位影像處理中常用的技術,目標是找出影像中物體與背景之間的界限或邊緣.對於物體識別.圖像分割和特徵提取等應用非常重要. Sobel 運算子的作用是通過檢測影像中亮度的變化,找出邊緣的位置.是通過卷積（Convolution）的方式來實現的,使用了一個 3x3 的卷積核（模板或遮罩）.這個卷積核分別對影像進行水平和垂直方向的卷積,計算出每個像素點的梯度值.梯度值的大小和方向可以用來指示該點的邊緣強度和方向.$G = \sqrt{Gx^2 + Gy^2}$若 G 的值高於門檻,則把輸出中相對應的像素設成白色(像素值 255),否則設成黑色(像素值 0);

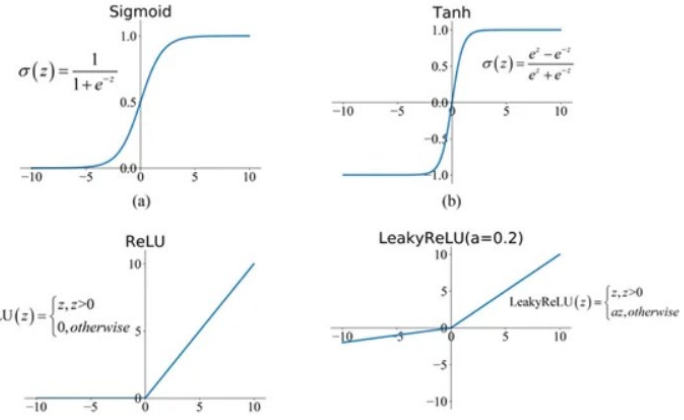


可以看到每一個圓圈代表的就是一個neuron，然後一定數量的neuron就會構成一個layer，當資料輸入進來後，就會通過第一層layer，然後輸出的結果再輸入到第二層layer，以此類推直到得到我們最後的結果。
而接收輸入資料的layer叫做input layer，輸出結果的layer叫做output layer，中間連接的layer叫做hidden layer。



所有neuron的權重w

而這裡的f就是所謂的activation function，像比較常見的activation function



梯度下降（Gradient descent）

$$\Delta w = -\eta \nabla E[w]$$

更新的程度會由 ∇E（Gradient）乘上一個「η（Learning rate）」來決定，而這邊的 ∇E 會受到一次丟給神經網路做運算的資料量影響，所以 Batch size 會影響神經網路收斂過程中，往 Minima 前進的速度與方向。

Epoch

訓練模型過程裡，演算法完整使用過資料集每筆
假設我們有一個很小的數據集，只有 10 筆資料，把它們全丟進神經網路運算完，那就是完成了 1 個 Epoch 的訓練。

Batch size

Batch 中文直翻的話意思是「批次」。

假設剛剛的 10 筆資料，每筆數據量都很大，電腦因為記憶體不夠用所以無法一次跑 10 筆。這時我們只好把 10 筆資料**分批**送進神經網路，這裡**分批的數量**就是 Batch size，今天我一次放 2 筆資料進神經網路運算，那 Batch size 就等於 2。

Iteration

Iteration 意思是「迭代」，這個概念與 Batch size 息息相關，畢竟我們把資料分批送進神經網路的時候，在程式語言的寫法上會透過迴圈來實踐。

延續剛剛的例子，我們如果一次送 2 筆資料進神經網路，那就要跑 5 次迴圈才算 1 個 Epoch，這時就稱訓練過程迭代了 5 次。

Batch size 大的時候，因為考量了比較多的數據，修正的方向會比較準確，但因為每次 Iteration 要算比較久，往 Minima 前進的次數就相對小。
Batch size 相對小的時候，因為只有考量局部數據，所以修正的方向會出現偏差，但因為每次 Iteration 要計算的資料量比較少，就有機會在同樣的時間成本下，修正比較多次。

批次學習（Batch learning）：先將整個訓練數據集分為多個大小相等的批次，然後模型通過處理每個批次來進行權重的更新。可以想像成填鴨式教育，一次塞進一大包資訊。

📍優點：由於每個批次中包含多個樣本，批次學習能夠提供對整個訓練數據集的更全面的信息。這有助於減少訓練過程中的隨機性，使得模型更穩定並

📍缺點：批次學習要求將整個訓練數據集加載到內存中，這對於大型數據集可能會產生內存壓力。此外，對於非平穩的數據分佈，批次學習可能會收斂到局部最優解，而無法及時適應變化。