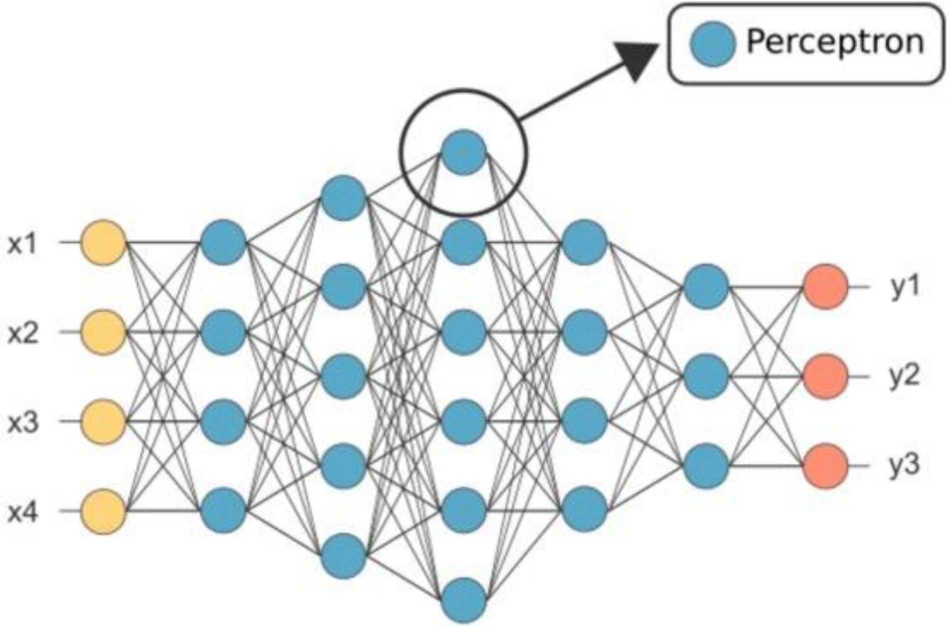
Artificial Neural Networks

Nicktming.

神经网络:



导数(derivatives):该函数曲线在这一点上的切线斜率

$$f'(x_0)=\lim_{\Delta x\to 0}\frac{\Delta y}{\Delta x}=\lim_{\Delta x\to 0}\frac{f(x_0+\Delta x)-f(x_0)}{\Delta x}$$

常函数（即常数）	$y=C$（C为常数）	$y'=0$
指数函数	$y=a^x$ $y=e^x$	$y'=a^x\ln a$ $y'=e^x$
幂函数	$y=x^n$	$y'=nx^{n-1}$
对数函数	$y=\log_a x$ $y=\ln x$	$y'=\frac{1}{x\ln a}$ $y'=\frac{1}{x}$

$f(x)=3x$

$f'(x)=3$

$$f'(x_0)=\lim_{\Delta x\to 0}\frac{\Delta y}{\Delta x}=\lim_{\Delta x\to 0}\frac{f(x_0+\Delta x)-f(x_0)}{\Delta x}=\lim_{\Delta x\to 0}\frac{3(x_0+\Delta x)-3x_0}{\Delta x}=3$$

$f(x)=3x^2+4x+5$

$f'(x)=\frac{df(x)}{dx}=6x+4$

$$f'(x_0)=\lim_{\Delta x\to 0}\frac{\Delta y}{\Delta x}=\lim_{\Delta x\to 0}\frac{f(x_0+\Delta x)-f(x_0)}{\Delta x}$$

$$=\lim_{\Delta x\to 0}\frac{3(x_0+\Delta x)^2+4(x_0+\Delta x)+5-(3x_0^2+4x_0+5)}{\Delta x}$$

$$=\lim_{\Delta x\to 0}\frac{6x_0\Delta x+3\Delta x^2+4\Delta x}{\Delta x}$$

$$=\lim_{\Delta x\to 0}6x_0+4+3\Delta x$$

$$=6x_0+4$$

复合函数导数:链式法则(Chain Rule)

复合函数对自变量的导数，等于已知函数对中间变量的导数，乘以中间变量对自变量的导数

$$f(x)=e^{-x}$$

$$f'(x)=e^{-x}\cdot(-x)'$$

$$=-e^{-x}$$

sigmoid: $\sigma(x)=\dfrac{1}{1+e^{-x}}$

$$\sigma'(x)=\left((1+e^{-x})^{-1}\right)'$$

$$=-1\,(1+e^{-x})^{-2}\cdot(1+e^{-x})'$$

$$=-1\cdot(1+e^{-x})^{-2}\cdot(-e^{-x})$$

$$=\frac{e^{-x}}{(1+e^{-x})^{2}}$$

$$=\left(1-\frac{1}{1+e^{-x}}\right)\cdot\frac{1}{1+e^{-x}}$$

$$=\big(1-\sigma(x)\big)\,\sigma(x)$$

$$\boxed{\begin{aligned}&\sigma(h)=\frac{1}{h}=h^{-1}\\&h(x)=1+e^{-x}\\&\sigma'(x)=\sigma'(h)\cdot h'(x)\end{aligned}}$$

偏导数:存在多个变量

每个变量的导数(对于结果有多大影响)

- $\frac{\partial}{\partial x}(x^2y^2) = 2xy^2$
- $\frac{\partial}{\partial y}(-2y^5 + z^2) = -10y^4$
- $\frac{\partial}{\partial \theta_2}(5\theta_1 + 2\theta_2 - 12\theta_3) = 2$
- $\frac{\partial}{\partial \theta_2}(0.55-(5\theta_1 + 2\theta_2 - 12\theta_3)) = -2$

梯度:多变量微分的一般化每个变量的导数(对于结果有多大影响)

梯度：对于可微的数量场 $f(x,y,z)$，以 $\left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}\right)$ 为分量的向量场称为f的梯度或斜量。

顾名思义，梯度下降法的计算过程就是沿梯度下降的方向求解极小值（也可以沿梯度上升方向求解极大值）。

其迭代公式为 $a_{k+1} = a_k + \rho_k \bar{s}^{(k)}$,其中 $\bar{s}^{(k)}$ 代表梯度负方向，ρ_k 表示梯度方向上的搜索步长。梯度方向我们可以通过对函数求导得到，步长的确定比较麻烦，太大了的话可能会发散，太小收敛速度又太慢。一般确定步长的方法是由线性搜索算法来确定，即把下一个点的坐标看做是a_{k+1}的函数，然后求满足$f(a_{k+1})$的最小值的a_k+1即可。

例子:

f(x) = 3x^2 + 4x + 5 找到一个x使得f(x)变小   x := x - a * f’(x)

J(w, b) want to find w,b that minimize J(w, b)   w := w - a * dw; b := b - a * db.

例子1: $f(x) = 3x^2 + 4x + 5$

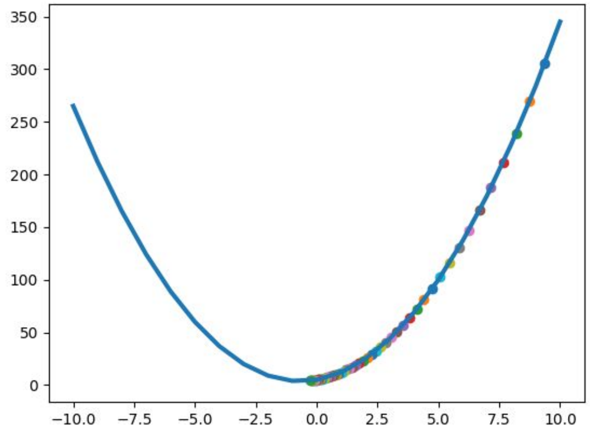
```
(loop_count, current_x, current_y)
(0, 10, 345)
(1, 3.5999999999999996, 58.27999999999994)
(2, 1.0399999999999996, 12.404799999999996)
(3, 0.015999999999999792, 5.064767999999999)
(4, -0.39360000000000006, 3.8903628799999996)
(5, -0.55744, 3.7024580607999997)
(6, -0.6229760000000001, 3.6723932897280003)
(7, -0.6491904000000001, 3.66758292635648)
(8, -0.6596761600000001, 3.666813268217037)
(9, -0.663870464, 3.666690122914726)
```

给出一个起点(start_x) 和步长(step)

如何利用**梯度下降法**去寻找到可以使f(x)**变小**？

导数: $dx = 6x + 4$ $x := x - a * f'(x)$ gradient_test_1.py

```python
start_x = 10
step = 0.1
current_x = start_x
current_y = 3 * current_x * current_x + 4 * current_x + 5
print("(loop_count, current_x, current_y)")
for i in range(10):
    print(i, current_x, current_y)
    derivative_f_x = 6 * current_x + 4
    current_x = current_x - step * derivative_f_x
    current_y = 3 * current_x * current_x + 4 * current_x + 5
```



例子2: $f(x, y) = 3x^2 + 4y^2 + 5$

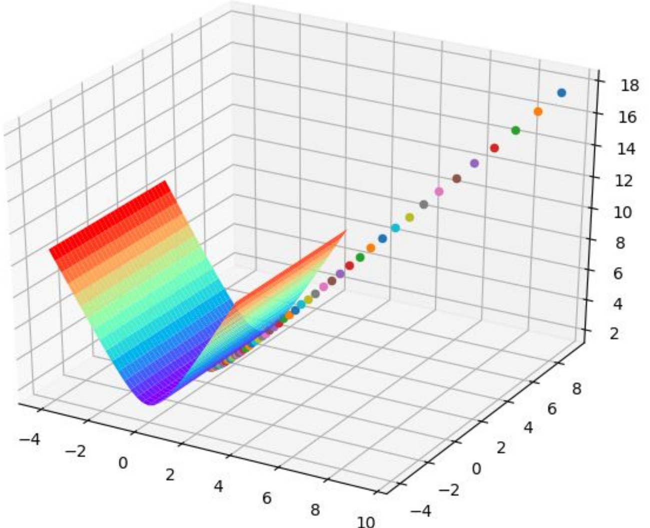
给出一个起点(start_x, start_y) 和步长(step)

如何利用**梯度下降法**去寻找到可以使f(x, y)**变小**？

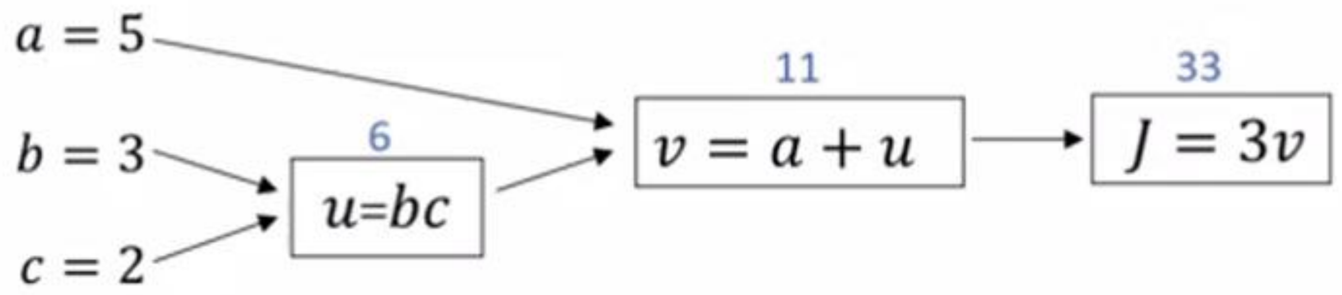
导数: dx = 6x　dy = 8y　gradient_test_3.py

```
start_x = 10
start_y = 10
step = 0.01
current_x = start_x
current_y = start_y
current_f = 3 * current_x * current_x + 4 * current_y + 5
print("(loop_count, current_x, current_y, current_f)")
for i in range(100):
    print(i, current_x, current_y, current_f)
    ### derivatives of x and y
    derivative_f_x = 6 * current_x
    derivative_f_y = 8 * current_y
    ### update x, y
    current_x = current_x - step * derivative_f_x
    current_y = current_y - step * derivative_f_y
    ### current f
    current_f = 3 * current_x * current_x + 4 * current_y + 5
```

```
(loop_count, current_x, current_y, current_f)
(0, 10, 10, 345)
(1, 9.4, 9.2, 306.88000000000005)
(2, 8.836, 8.463999999999999, 273.080688)
(3, 8.30584, 7.786879999999998, 243.1084543168)
(4, 7.8074896, 7.163929599999999, 216.52639996232446)
(5, 7.339040224000001, 6.590815231999999, 192.9477951564699)
(6, 6.898697810560001, 6.063550013439999, 172.03029449803603)
(7, 6.484775941926401, 5.578466012364799, 153.47082110042152)
(8, 6.095689385410817, 5.1321887313756145, 137.00104217573278)
(9, 5.729948022286168, 4.721613632865566, 122.38336754576578)
(10, 5.386151140948998, 4.343884542236321, 109.40741050838388)
```



计算图的导数



规定: dx 表示 J 对 变量 x 的偏导 (因为我们只关注各个变量对最终J的结果会产生怎样的影响)

dv = 3; dv/da = 1; dv/du = 1; du/db = c; du/dc = b

da = dv * dv/da = 3;

db = dv * dv/du * du/db = 3*1*b = 6;

dc = dv * dv/du * du/dc = 3*1*c = 9

a:5->J:33	a:5.1->33.3
b:3->J:33	b:3.1->33.6
c:2->J:33	c:2.1->33.9

计算图的导数: 结果

目标:给出a,b,c 把目标J变小到0.1以下

gradient_test_3.py

```
('J:', 33)
('J:', 8.58)
('J:', 5.46765648)
('J:', 2.7136824448030787)
('J:', 0.9855570692702882)
('J:', 0.2764468379291 3805)
```

```python
step = 0.1

a = 5
b = 3
c = 2

u = b * c
v = a + u
J = 3 * v

while not J < 0.1 :
    print("J:", J)
    # derivatives of variables
    derivative_J_v = v
    derivative_v_a = 1
    derivative_v_u = 1
    derivative_u_b = c
    derivative_u_c = b

    derivative_J_a = derivative_J_v * derivative_v_a
    derivative_J_b = derivative_J_v * derivative_v_u * derivative_u_b
    derivative_J_c = derivative_J_v * derivative_v_u * derivative_u_c

    #update variables
    a = a - step * derivative_J_a
    b = b - step * derivative_J_b
    c = c - step * derivative_J_c

    u = b * c
    v = a + u
    J = 3 * v
```

讨论 $W^Tx + b$

最简单的 $f(x) = w1 * x + b1$

Loss_function : (y_prediction - y)2

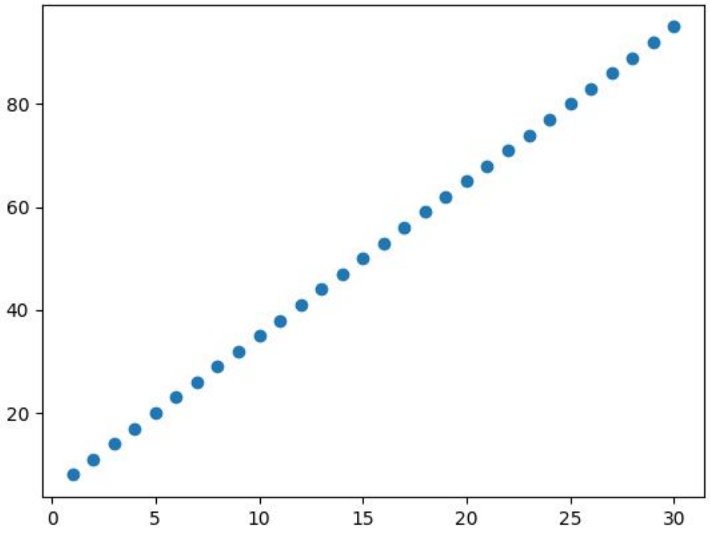
Cost_function : 1/2m * mean(loss_function)

重要: find(w1, b1) to minimize cost_function

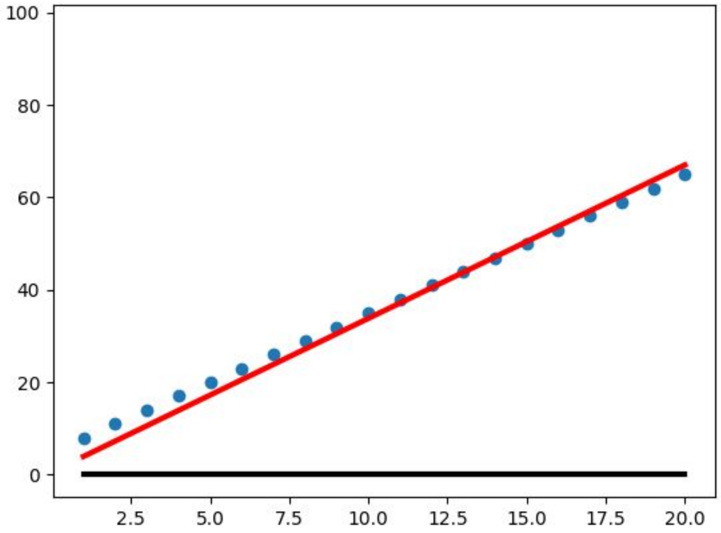
$dw1 = 1/m * sum(y^i\_prediction - y^i) * x^i$

$db1 = 1/m * sum(y^i\_prediction - y^i)$ where $1 <= i <= m$ (num of points)

例子 : f(x) = 3x + 5



80
60
40
20
0
5
10
15
20
25
30



100
80
60
40
20
0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

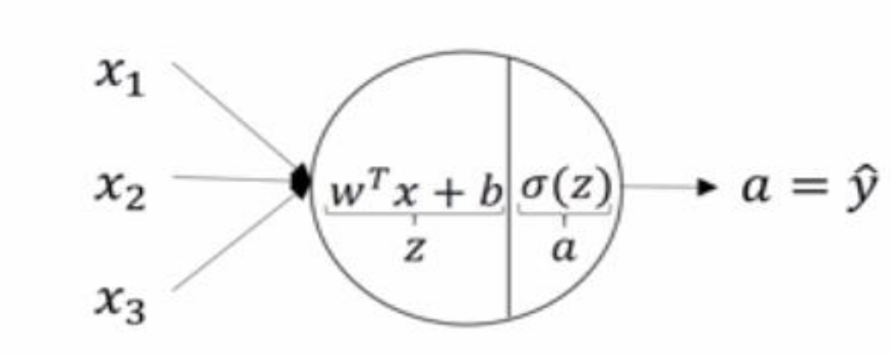
例子 : $f(x) = 3x + 5$ gradient_test_6.py

```python
w1 = 0
b1 = 0
step = 0.01

def cost_function(y_prediction):
    return 1.0/(2 * m) * np.sum(np.square(y_prediction - y_data))

y_prediction = x_data * w1 + b1
ax.plot(x_data, y_prediction, 'black', lw=3)

print("(i, cost_function)")
for i in range(250):

    print(i, cost_function(y_prediction))

    derivative_f_w1 = 1.0/m * np.sum(np.multiply(y_prediction - y_data, x_data))
    derivative_f_b1 = 1.0/m * np.sum(y_prediction - y_data)

    w1 = w1 - step * derivative_f_w1
    b1 = b1 - step * derivative_f_b1
    y_prediction = x_data * w1 + b1
```

```
(i, cost_function)
(0, 815.75)
(1, 161.96366249999997)
(2, 33.825554222343683)
(3, 8.7035657538191487)
(4, 3.7706295837662687)
(5, 2.7943700155573574)
(6, 2.5935802028884014)
(7, 2.5448098378469552)
(8, 2.5258754794532092)
(9, 2.5128315252433113)
(10, 2.5009849384485019)
```

```
(240, 0.86525194401058888)
(241, 0.86126822346531429)
(242, 0.85730284443258287)
(243, 0.85335572246593527)
(244, 0.84942677350771945)
(245, 0.8455159138872903)
(246, 0.8416230603192334)
(247, 0.83774812990159297)
(248, 0.8338910401140972)
(249, 0.83005170881641188)
('w1:', 3.1957173026237342, 'b1:', 2.32949279180156)
```

f(x) = w1 * x + b1外层再加入一个Sigmoid函数h(x).



Sigmoid: $h(x) = \sigma(x) = \frac{1}{1+e^{-x}}$

h’(x) = h(x) * (1 - h(x))

求y对w的导数:

dy/da = 1
da/dz = (1 - h(z)) * h(z)
dz/dw1 = x
dz/db1 = 1

---------->

dy/dw1=dy/da * da/dz * dz/dw1=1*h(z)*(1-h(z))*x
dy/db1 =dy/da * da/dz * dz/db1=1*h(z)*(1-h(z))

激励函数: 一般都是非线性函数

Common Activation Functions

- Threshold: $h(x) = \begin{cases} 1 & x \geq 0 \\ -1 & x < 0 \end{cases}$
- Sigmoid: $h(x) = \sigma(x) = \frac{1}{1+e^{-x}}$
- Gaussian: $h(x) = e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$
- Hyperbolic tangent: $h(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
- Identity: $h(x) = x$

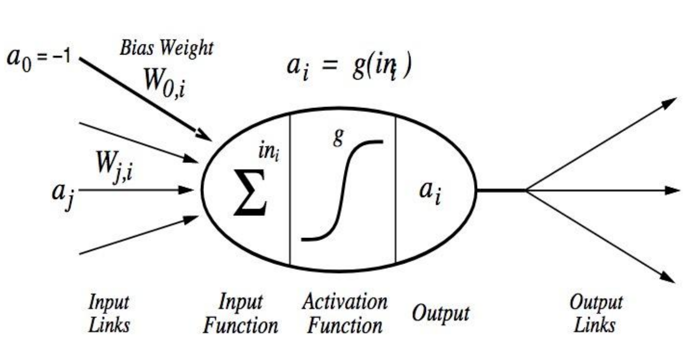
为什么需要非线性函数:

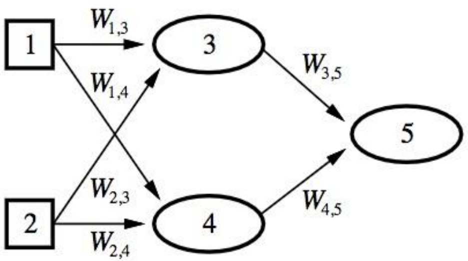
如果是线性激励函数的话,整个网络还是线性方程.可以表示成:

(参数)x_1 + (参数)x_2 + … + (参数)x_n

ANN: Artificial neural network

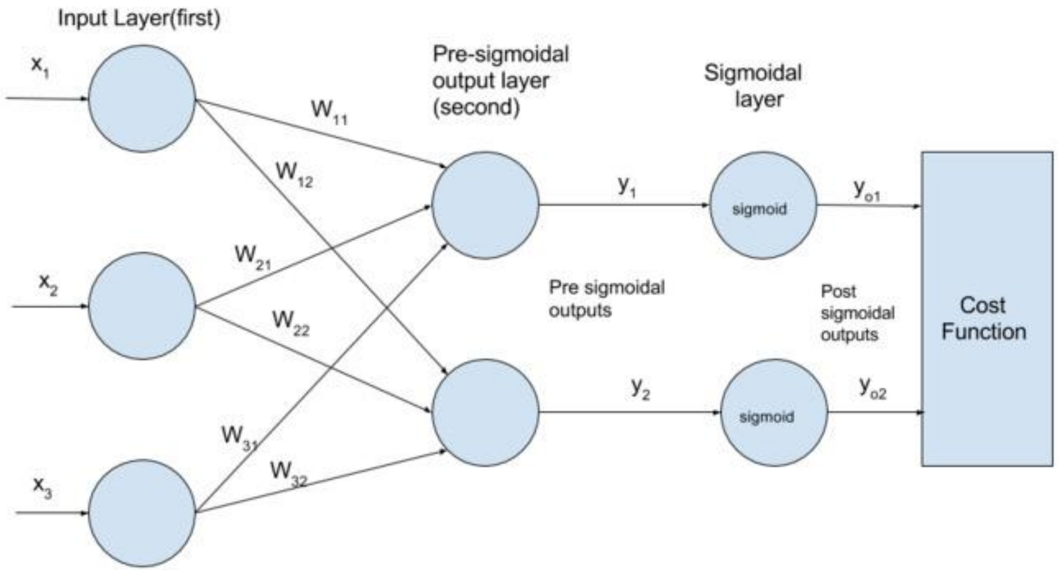
ANN Unit



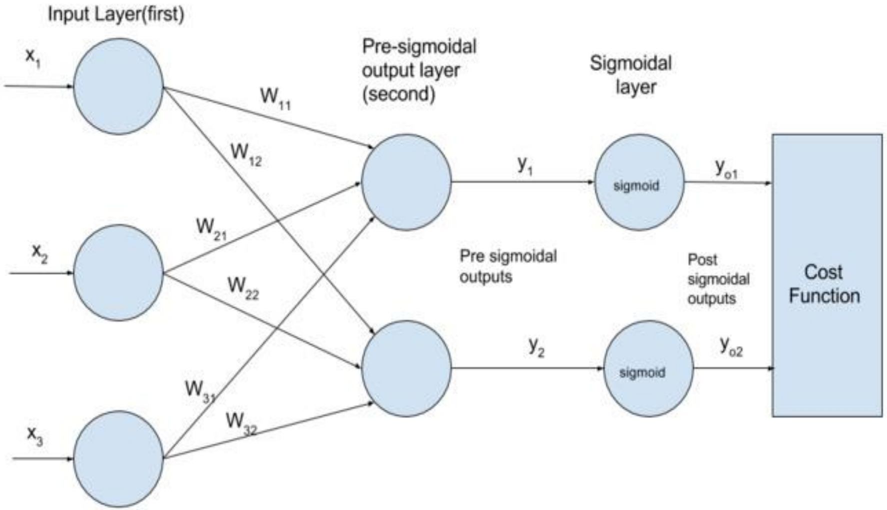


$$
\begin{aligned}
a_5 &= g(W_{3,5}a_3 + W_{4,5}a_4) \\
&= g(W_{3,5}g(W_{1,3}a_1 + W_{2,3}a_2) + W_{4,5}g(W_{1,4}a_1 + W_{2,4}a_2))
\end{aligned}
$$

例子: cost_function : Least squares(最小平方法)



例子: cost_function : Least squares(**最小平方法**) num_of_samples=1



forward:

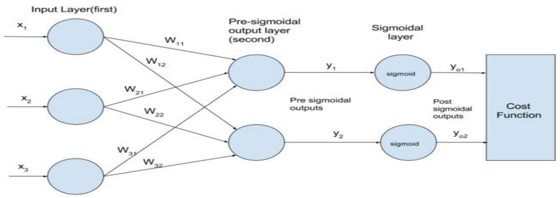
X = [[x1, x2, x3]] shape=[1,3]

W = [
[W11, W12]
[W21, W22]
[W31, W32]
]
shape = [3, 2]

y = np.dot(X, W) = [[y1, y2]]. **shape = [1, 2]**

yo = sigmoid(y) = [[yo1, yo2]] **shape = [1, 2]**

例子: cost_function : Least squares(**最小平方法**)



目标:

想要知道W对
cost_func的影响

[
[dc/dw11   dc/dw12]
[dc/dw21   dc/dw22]
[dc/dw31   dc/dw32]
]

求导: h(x) = sigmoid Y1,Y2是样本数据

dc/dyo1 = (yo1 - Y1)/m
dc/dyo2 = (yo2 - Y2)/m

dc/dyo = [[dc/dyo1, dc/dyo2]]

dyo1/dy1= h(y1)(1-h(y1))
dyo2/dy2= h(y2)(1-h(y2))

dyo/dy = [[dyo1/dy1, dyo2/dy2]]

Y1 = x1*W11+x2*W21+x3*W31
Y2 = x1*W12+x2*W22+x3*W32

dy1/dw11=x1;dy1/dw21=x2;dy1/dw31=x3
dy2/dw12=x1;dy2/dw22=x2;dy2/dw32=x3

例子: cost_function : Least squares(**最小平方法**)

$$
\begin{bmatrix}
\left[\frac{dc}{dy_{01}}\cdot\frac{dy_{01}}{dy_1}\cdot\frac{dy_1}{dw_{11}} \quad \frac{dc}{dy_{02}}\cdot\frac{dy_{02}}{dy_2}\cdot\frac{dy_2}{dw_{12}}\right] \\
\left[\frac{dc}{dy_{01}}\cdot\frac{dy_{01}}{dy_1}\cdot\frac{dy_1}{dw_{21}} \quad \frac{dc}{y_{02}}\cdot\frac{dy_{02}}{dy_2}\cdot\frac{dy_2}{dw_{22}}\right] \\
\left[\frac{dc}{dy_{01}}\cdot\frac{dy_{01}}{dy_1}\cdot\frac{dy_1}{dw_{31}} \quad \frac{dc}{y_{02}}\cdot\frac{dy_{02}}{dy_2}\cdot\frac{dy_2}{dw_{32}}\right]
\end{bmatrix}
\qquad shape :[3,2]
$$

$$\Rightarrow$$

$$
\begin{bmatrix} [x_1] \\ [x_2] \\ [x_3] \end{bmatrix}
*
\left[\left[\frac{dc}{dy_{01}}\cdot\frac{dy_{01}}{dy_1} \quad \frac{dc}{y_{02}}\cdot\frac{dy_{02}}{dy_2}\right]\right]
\Rightarrow np.dot(X.T, np.multiply(\tfrac{dc}{dy_0}, \tfrac{dy_0}{dy}))
$$

$$\Downarrow$$

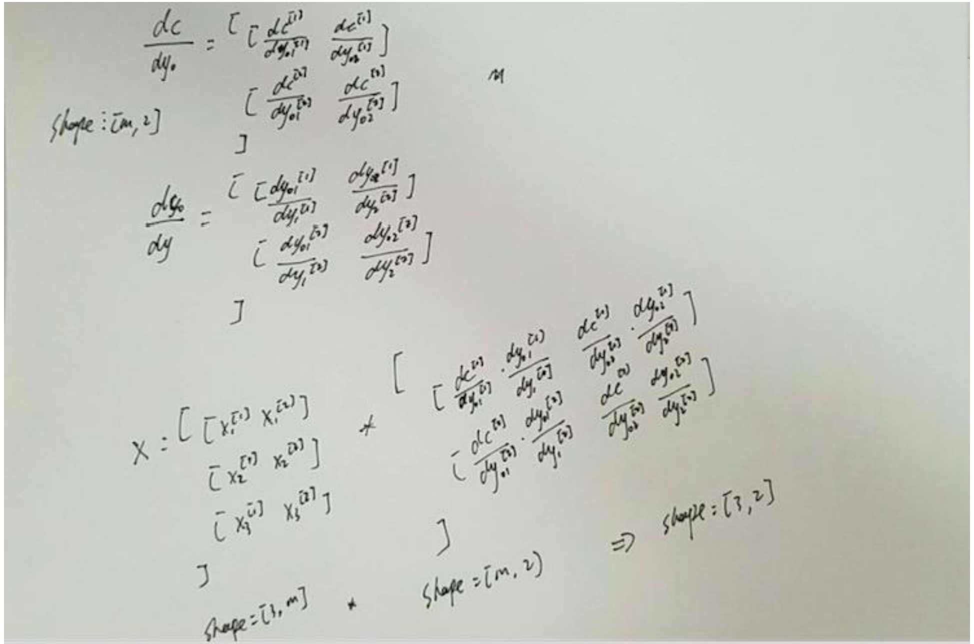
$$shape\,[3,1] * shape\,[1,2]$$

例子: cost_function : Least squares(**最小平方法**)

```
('start:', 0.25179515150219911)
('end:', 0.099951514321263799)
('cnt:', 1250)
```

```python
m = 1
step = 0.01

def sigmoid(x):
    return 1/(1+np.exp(-x))

def derivative_sigmoid(x):
    return np.multiply(1 - sigmoid(x), sigmoid(x))

def cost_function(yo, Y):
    return 1./(2*m) * np.sum(np.square(np.subtract(yo, Y)))

#shape 1*3
X = np.ones((m, 3))
Y = np.random.rand(m, 2)

#shape 3*2
W = np.ones((3, 2))

#shape 1*2
y = np.dot(X, W)

#shape 1*2
yo = sigmoid(y)

cost = cost_function(yo, Y)

print("start:", cost)

cnt = 0;

while not cost < 0.1 :
    derivative_c_y = np.subtract(yo, Y) / m

    derivative_yo_y = derivative_sigmoid(y)

    dw = np.dot(X.T, np.multiply(derivative_c_y, derivative_yo_y))

    W = W - step * dw
    y = np.dot(X, W)
    yo = sigmoid(y)
    cost = cost_function(yo, Y)
    cnt += 1

print("end:", cost)
print("cnt:", cnt)
```

例子: num_of_samples = m (m > 1) m = 2



结论:

与m=1的导数公式是一样不用做改变

Tensorflow 实现神经网络

```python
import tensorflow as tf
import numpy as np

x_data = np.float32(np.random.rand(2, 100))
y_data = np.dot([0.100, 0.200], x_data) + 0.300

b = tf.Variable(tf.zeros([1]))
W = tf.Variable(tf.random_uniform([1, 2], -1.0, 1.0))
y = tf.matmul(W, x_data) + b


loss = tf.reduce_mean(tf.square(y - y_data))
optimizer = tf.train.GradientDescentOptimizer(0.5)
train = optimizer.minimize(loss)


init = tf.initialize_all_variables()


sess = tf.Session()
sess.run(init)

for step in xrange(0, 201):
    sess.run(train)
    if step % 20 == 0:
        print step, sess.run(W), sess.run(b)
```

Thank you.