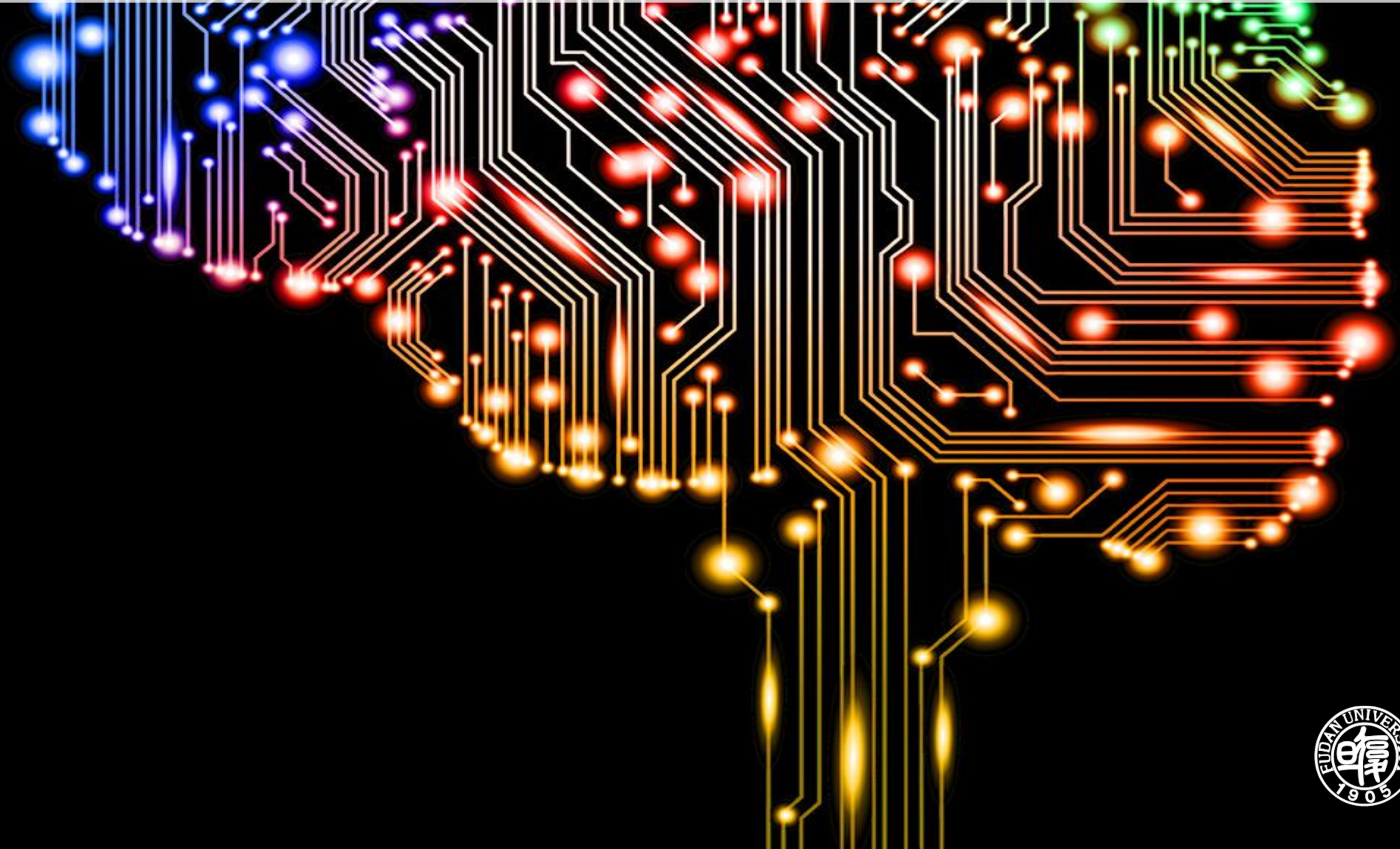
数据驱动的人工智能（3b）神经网络基础

Data Driven Artificial Intelligence

邬学宁 SAP硅谷创新中心

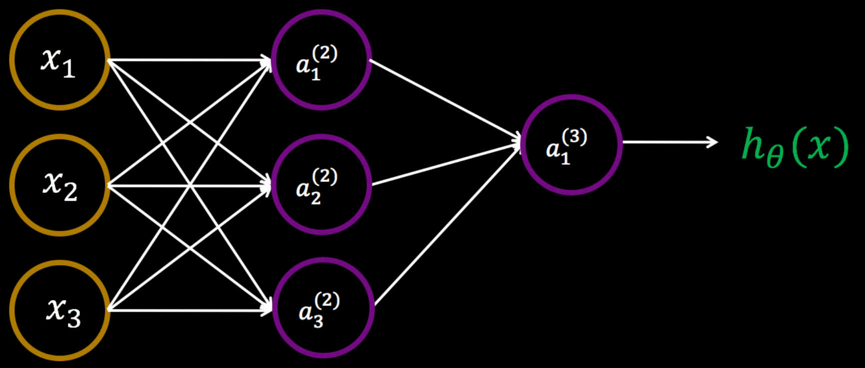
2017 / 03



日程

Feeding Forward MLP
Back Propagation
TensorFlow Introduction

ANN: Feeding Forward Vectorization



$$a_1^{(2)} = \sigma(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3)$$

$$a_2^{(2)} = \sigma(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3)$$

$$a_3^{(2)} = \sigma(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3)$$

$$h_\theta(x) = a_1^{(3)} = \sigma(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)})$$

$$x = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad z^{(2)} = \begin{bmatrix} z_0^{(2)} \\ z_0^{(2)} \\ z_0^{(2)} \\ z_0^{(2)} \end{bmatrix}$$

$$z^{(2)} = \Theta^{(1)}x$$

$$a^{(2)} = \sigma(z^{(2)})$$

$$add\ a_0^{(2)} = 1$$

$$z^{(3)} = \Theta^{(2)}a^{(2)}$$

$$h_\theta(x) = a_1^{(3)} = \sigma(z^{(3)})$$

Recap：Chain Rule

- $f(x,y,z) = (x+y)z$ 可以写成$f = qz$,其中$q = x + y$

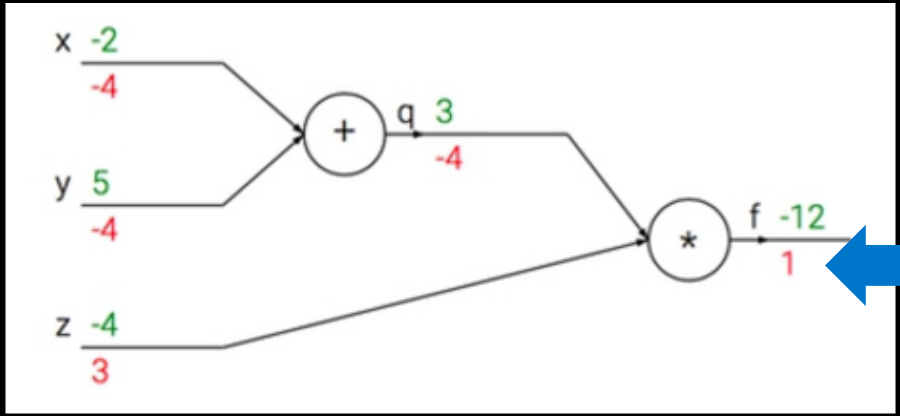
$$q = x + y. \qquad \frac{dq}{dx} = 1, \frac{dq}{dy} = 1$$

$$f = qz. \qquad \frac{df}{dq} = z, \frac{df}{dz} = q$$

$$\frac{df}{dx} = \frac{df}{dq}\frac{dq}{dx} = z = -4$$

$$\frac{df}{dy} = \frac{df}{dq}\frac{dq}{dy} = z = -4$$

$$\frac{df}{dz} = q = x + y = 3$$



FeiFei LI & Andrej Karpathy & Justin Johnson

$$\frac{df}{dq} = z = -4$$

$$\frac{df}{df} = 1$$

SAP

复旦大学 FUDAN UNIVERSITY 1905

Recap：Chain Rule

Case 1: $y = g(x); z = h(y)$

$$\Delta x \rightarrow \Delta y \rightarrow \Delta z$$

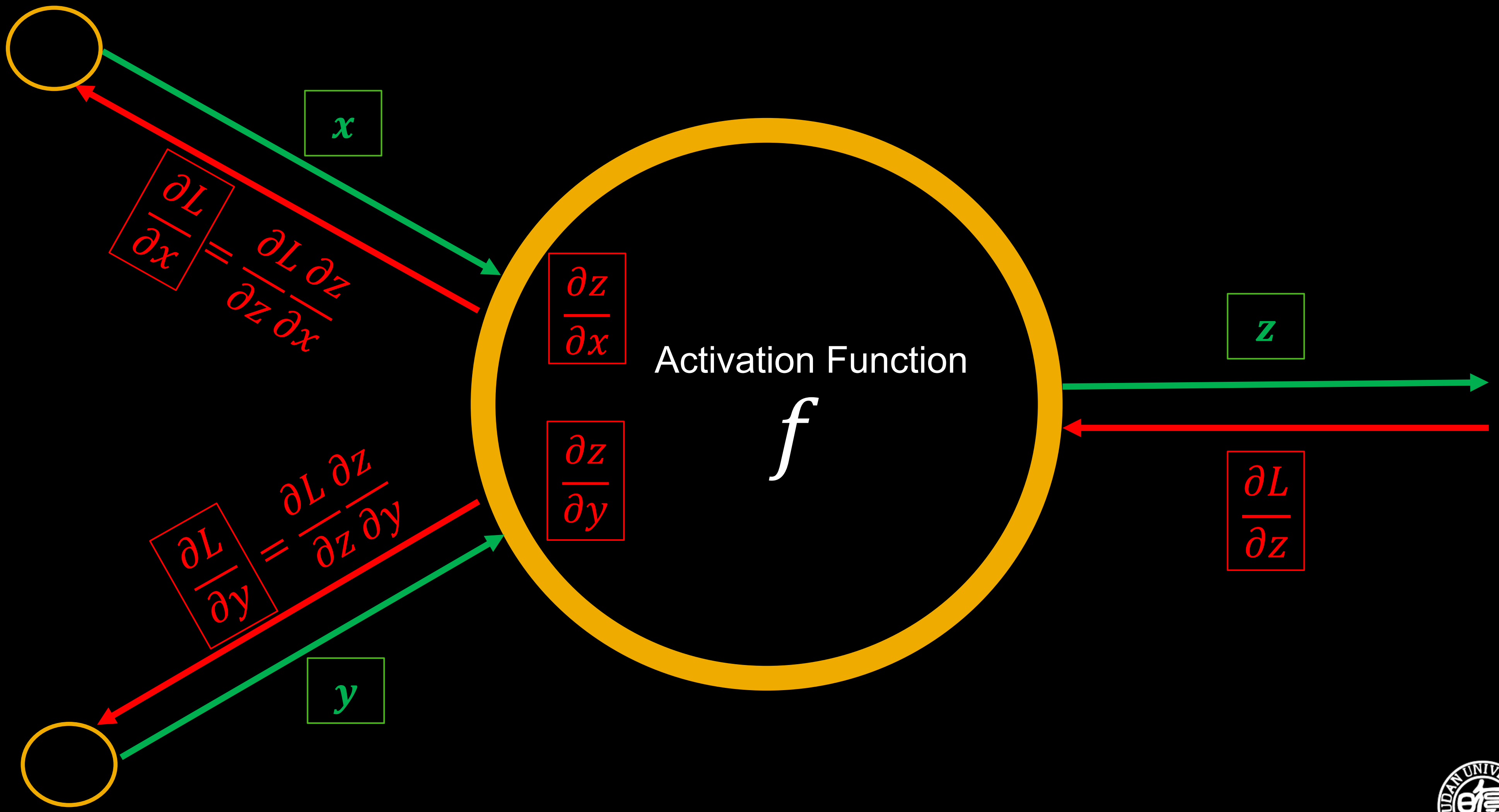
$$\boxed{\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}}$$

Case 2: $x = g(s);\ \ y = h(s);\ \ z = k(x, y)$

$$\begin{tikzcd} & \Delta x \arrow[dr] & \\ \Delta s \arrow[ur] \arrow[r] & \Delta y \arrow[r] & \Delta z \end{tikzcd}$$

$$\boxed{\frac{dz}{ds} = \frac{dz}{dx}\frac{dx}{ds} + \frac{dz}{dy}\frac{dy}{ds}}$$

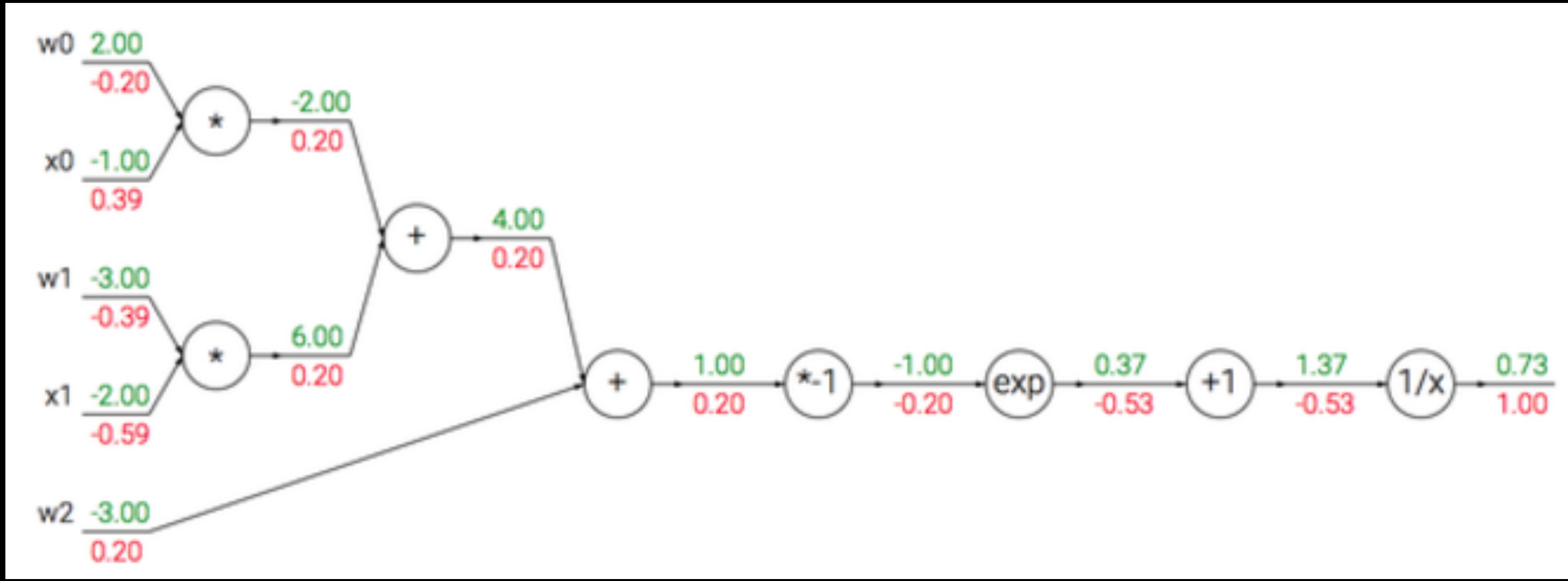
Activation Function: Back Propagation (BP)



FeiFei LI & Andrej Karpathy & Justin Johnson

Back Propagation Example

$$f(w,x) = \frac{1}{1 + e^{-(w_0x_0+w_1x_1+w_2)}}$$



$$f(x) = \frac{1}{x} \qquad \frac{d}{dx}f(x) = -\frac{1}{x^2}$$

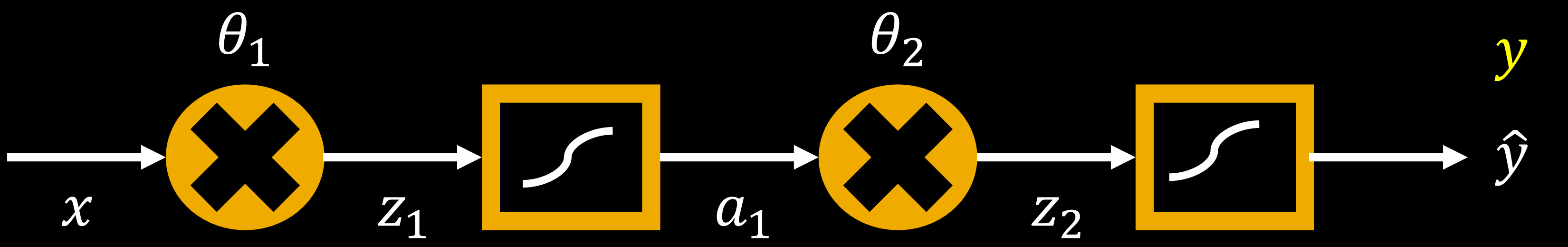
$$f(x) = c + x \qquad \frac{d}{dx}f(x) = 1$$

$$f(x) = e^x \qquad \frac{d}{dx}f(x) = e^x$$

$$f(x) = ax \qquad \frac{d}{dx}f(x) = a$$

$$-0.53e^x = -0.53e^{-1} = -0.20 \qquad \left(-\frac{1}{x^2}\right)(1.00) = -\frac{1}{1.37^2} = -0.53$$

Back Propagation（永远求偏导）



Cost Function (1 row):

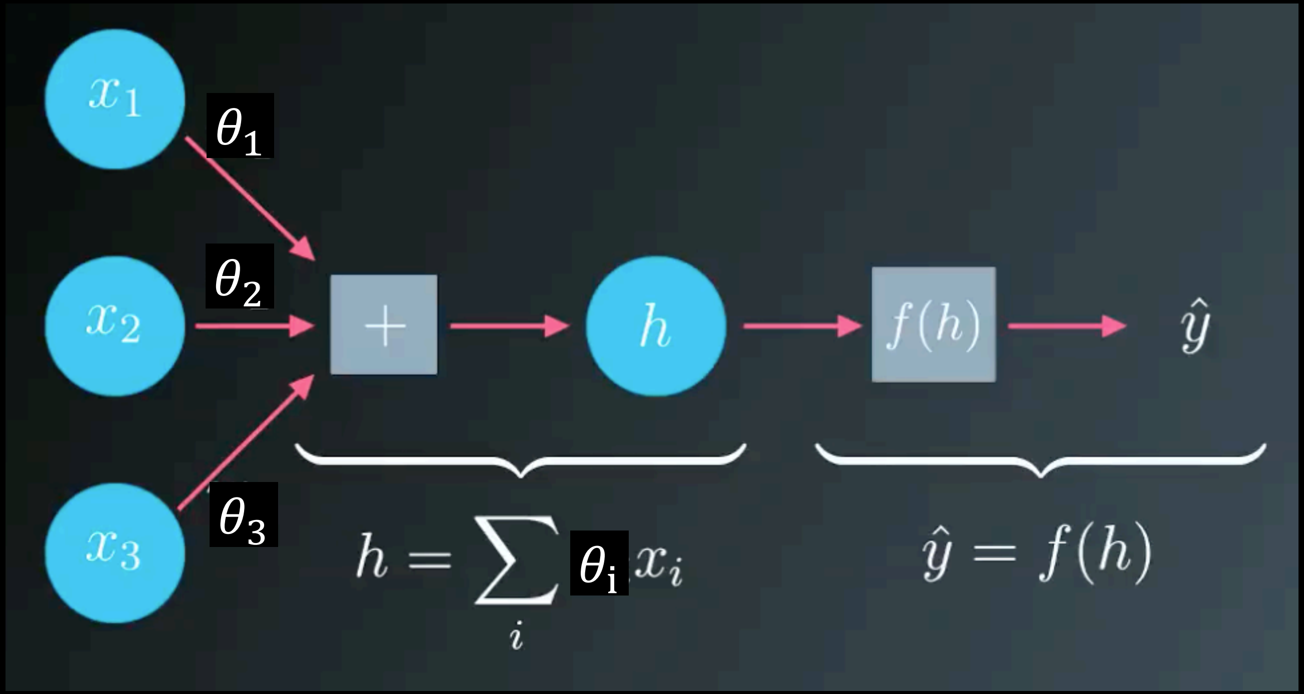
$$\boxed{\mathcal{J} = \frac{1}{2}(y-\hat{y})^2}$$

$$\frac{\partial \mathcal{J}}{\partial \theta_2} = \frac{\partial \mathcal{J}}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial \theta_2} = \frac{\partial \mathcal{J}}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial z_2}\frac{\partial z_2}{\partial \theta_2} = 2\frac{1}{2}(y-\hat{y})\frac{\partial}{\partial \hat{y}}(y-\hat{y})\frac{\partial \hat{y}}{\partial z_2}\frac{\partial z_2}{\partial \theta_2}$$

$$= -(y-\hat{y})\hat{y}(1-\hat{y})a_1$$

$$\frac{\partial \mathcal{J}}{\partial \theta_1} = \frac{\partial \mathcal{J}}{\partial z_2}\frac{\partial z_2}{\partial \theta_1} = \frac{\partial \mathcal{J}}{\partial z_2}\frac{\partial z_2}{\partial a_1}\frac{\partial a_1}{\partial z_1}\frac{\partial z_1}{\partial \theta_1} = -(y-\hat{y})\hat{y}(1-\hat{y})\,\theta_2 a_1(1-a_1)x$$

反向传播与梯度下降



$$\mathcal{J} = \frac{1}{2}\sum_i (y^{(i)} - \hat{y}^{(i)})^2$$

$$= \frac{1}{2}\sum_i (y^{(i)} - f(\sum_j \theta_j x_j^{(i)}))^2$$

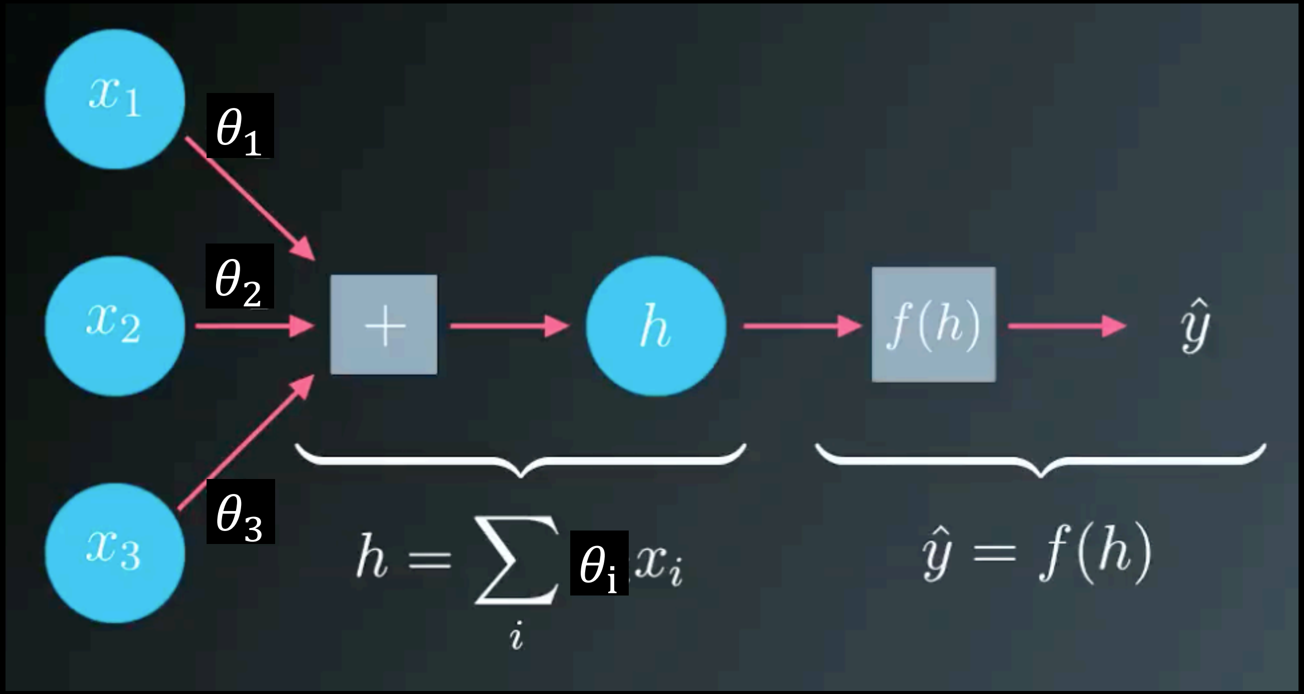
为了简化问题，我们假设只有一个数据样本，和一个输出节点：

$$\mathcal{J} = \frac{1}{2}(y - \hat{y})^2 = \frac{1}{2}(y - f(\sum_j \theta_j x_j)^2$$

SAP

复旦大學 FUDAN UNIVERSITY 1905

反向传播与梯度下降



$$\mathcal{J} = \frac{1}{2}(y - f\left(\sum_j \theta_j x_j\right))^2$$

$$\theta_j = \theta_j + \Delta\theta_j$$

$$\theta_j \propto -\frac{\partial \mathcal{J}}{\partial \theta_j}$$

$$\theta_j = -\alpha \frac{\partial \mathcal{J}}{\partial \theta_j}$$

$$\frac{\partial \mathcal{J}}{\partial \theta_j} = \frac{\partial}{\partial \theta_j}\frac{1}{2}(y - \hat{y})^2 = (y - \hat{y})\frac{\partial}{\partial \theta_j}(y - \hat{y}) = -(y - \hat{y})\frac{\partial}{\partial \theta_j} f(\sum_j \theta_j x_j)$$

$$= -(y - \hat{y})\, f'(h) x_j \qquad where\ h = \textstyle\sum_j \theta_j x_j$$

反向传播与梯度下降

$$\frac{\partial \mathcal{J}}{\partial \theta_i} = -(y - \hat{y}) f'(h) x_i \qquad \Delta\theta_i = \alpha (y - \hat{y}) f'(h) x_i$$

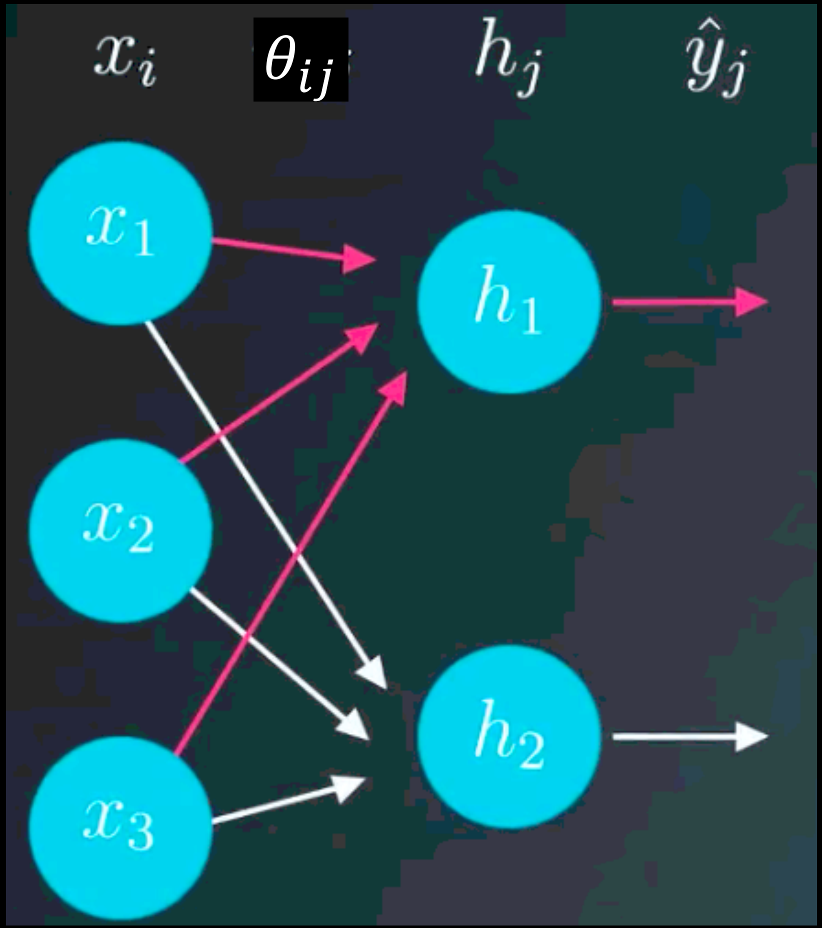
为了方便，我们定义：$\delta = (y - \hat{y}) f'(h)$

* 对于Sigmoid 激活函数：$f'(h) = f(h)(1 - f(h))$

所以，我们可将权重更新公式重写为：

$$\theta_i = \theta_i + \alpha \delta x_i$$

反向传播与梯度下降（多输出节点）



$$\delta = (y - \hat{y})f'(h)$$

$$\delta_j = (y_j - \hat{y}_j)f'(h_j)$$

$$\Delta\theta_i = \alpha\delta x_i$$

$$\Delta\theta_{ij} = \alpha\delta_j x_i$$

反向传播与梯度下降（实现）

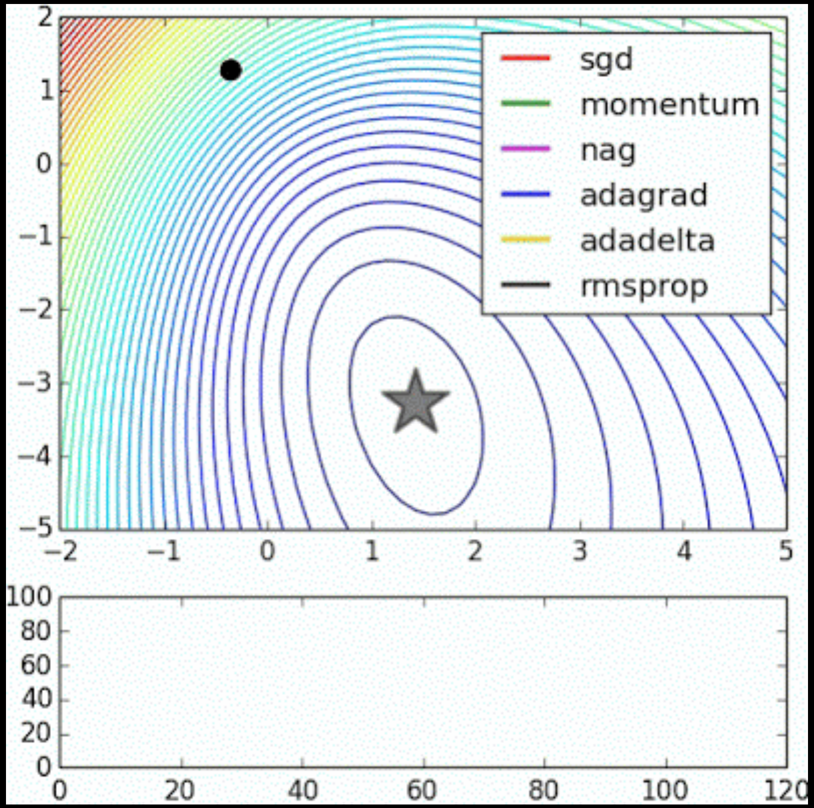
```python
import numpy as np
def sigmoid(x):
    return 1/(1+np.exp(-x))

def sigmoid_prime(x):
    return sigmoid(x) * (1 - sigmoid(x))

# Input/output/weights data
x = np.array([0.1, 0.3])
y = 0.2
weights = np.array([-0.8, 0.5])

learnrate = 0.5
```

```python
# the linear combination performed by the node
h = np.dot(x, weights)

# The neural network output (y-hat)
nn_output = sigmoid(h)

# output error (y - y-hat)
error = y - nn_output

# output gradient (f'(h))
output_grad = sigmoid_prime(h)

# error term (lowercase delta)
delta = error * output_grad

# Gradient descent step
del_w = learnrate * delta * x
print(del_w)
```

$$\delta_j = (y_j - \hat{y}_j) f'(h_j)$$

$$\Delta\theta_{ij} = \alpha \delta_j x_i$$

参数更新



logistic regression on noisy moons

Hello World!

```python
import tensorflow as tf

hello_constant = tf.constant('Hello World!')

with tf.Session() as sess:
    output = sess.run(hello_constant)
    print(output)
```

Linear Regression

```python
# imports
import numpy as np
import tensorflow as tf
import sklearn
import matplotlib.pyplot as plt

# hyper parameters

learning_rate = 0.01
training_epochs = 1000
display_steps = 100

# training data
train_X = np.array([3.3,4.4,5.5,6.71,6.93,4.168,9.779,6.182,7.59,2.167,7.042,10.791,5.313,7.997,5.654,9.27,3.1])
train_Y = np.array([1.7,2.76,2.09,3.19,1.694,1.573,3.366,2.596,2.53,1.221,2.827,3.465,1.65,2.904,2.42,2.94,1.3])

n_samples = train_X.shape[0]

X = tf.placeholder("float")
Y = tf.placeholder("float")

W = tf.Variable(np.random.randn(), name = 'weight')
b = tf.Variable(np.random.randn(), name = 'bias')

pred = tf.add(tf.mul(X,W),b)

cost = tf.reduce_sum(tf.pow(pred-Y,2))/2/n_samples

optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)

sess = tf.Session()

init = tf.global_variables_initializer()

sess.run(init)

for epoch in range(training_epochs):
    for (x,y) in zip(train_X,train_Y):
        sess.run(optimizer, feed_dict={X:x,Y:y})
    if epoch % 100 == 0:
        c = sess.run(cost, feed_dict={X:x,Y:y})
        print('epoch=',epoch,'cost=',c,'weight=',sess.run(W),'bias=',sess.run(b))
print('Optimization Finished')
```

Logistic/Softmax Regression

```
import tensorflow as tf

from tensorflow.examples.tutorials.mnist import input_data

mnist = input_data.read_data_sets("/tmp/data/", one_hot=True)

learning_rate = 0.01
training_epochs = 25
batch_size = 100
display_step = 1

x = tf.placeholder(tf.float32, [None, 784])
y = tf.placeholder(tf.float32, [None, 10])

W = tf.Variable(tf.random_normal([784,10]))
b = tf.Variable(tf.random_normal([10]))

pred = tf.nn.softmax(tf.matmul(x, W) + b)

cost = tf.reduce_mean(-tf.reduce_mean(y*tf.log(pred),reduction_indices=1))

optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)

init = tf.global_variables_initializer()

sess = tf.Session()

sess.run(init)

for epoch in range(training_epochs):
    num_batch = int(mnist.train.num_examples/batch_size)
    for i in range(num_batch):
        batch_xs,batch_ys = mnist.train.next_batch(batch_size)
        c = sess.run([optimizer, cost], feed_dict ={x: batch_xs, y:batch_ys})
```

Thank you!

Contact information:

邬学宁 (i025497)
Chief Data Scientist, SAP Silicon Valley Innovation Center
Address: No. 1001, Chenghui Road, Shanghai, 201023
Phone number: +8621-6108 5287
Email: x.wu@sap.com