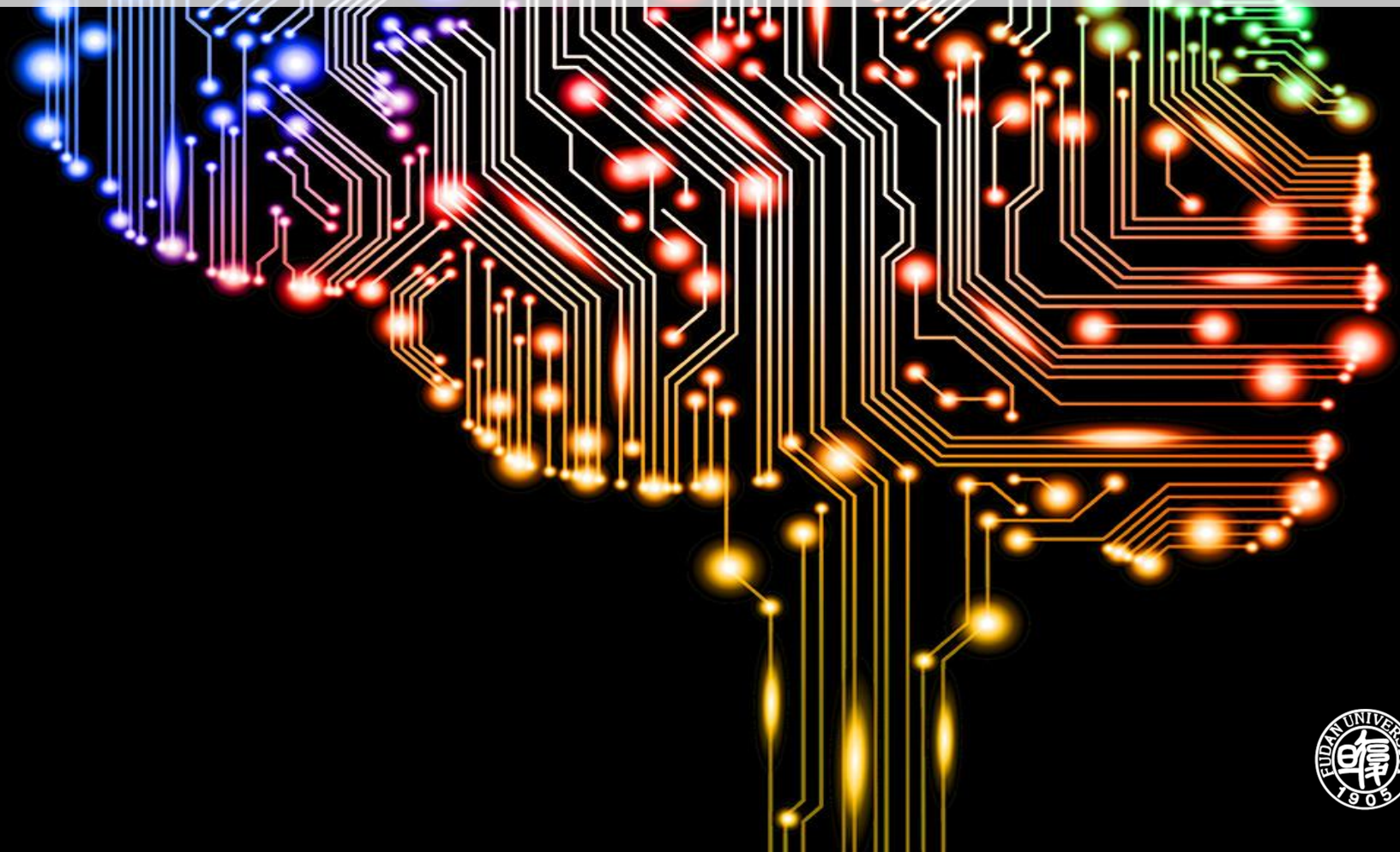
数据驱动的人工智能（3b）神经网络基础

Data Driven Artificial Intelligence

邬学宁 SAP硅谷创新中心

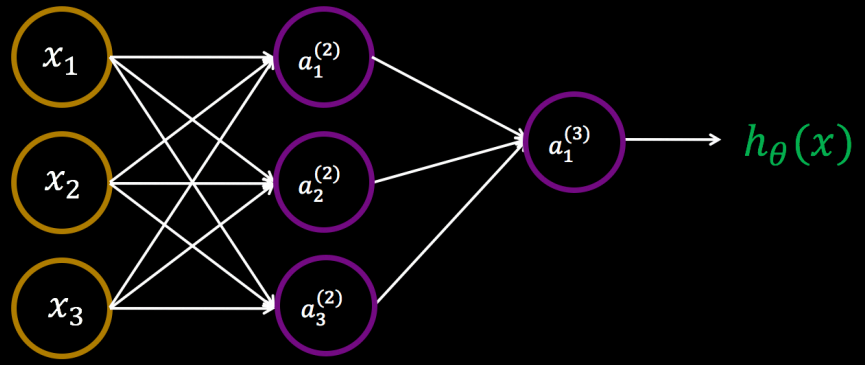
2017 / 03



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$q=x+y$

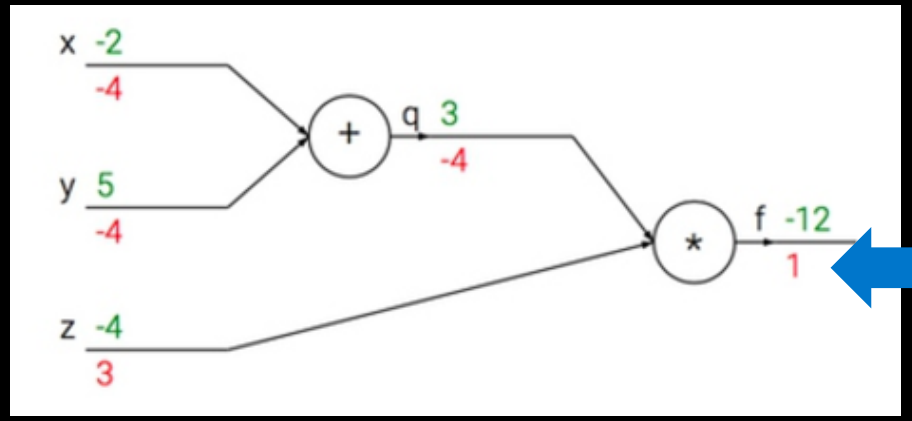
$$q=x+y. \qquad \frac{dq}{dx}=1, \frac{dq}{dy}=1$$

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

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

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

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



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

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

FeiFei LI & Andrej Karpathy & Justin Johnson

Recap：Chain Rule

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

$$\Delta x \to \Delta y \to \Delta z$$

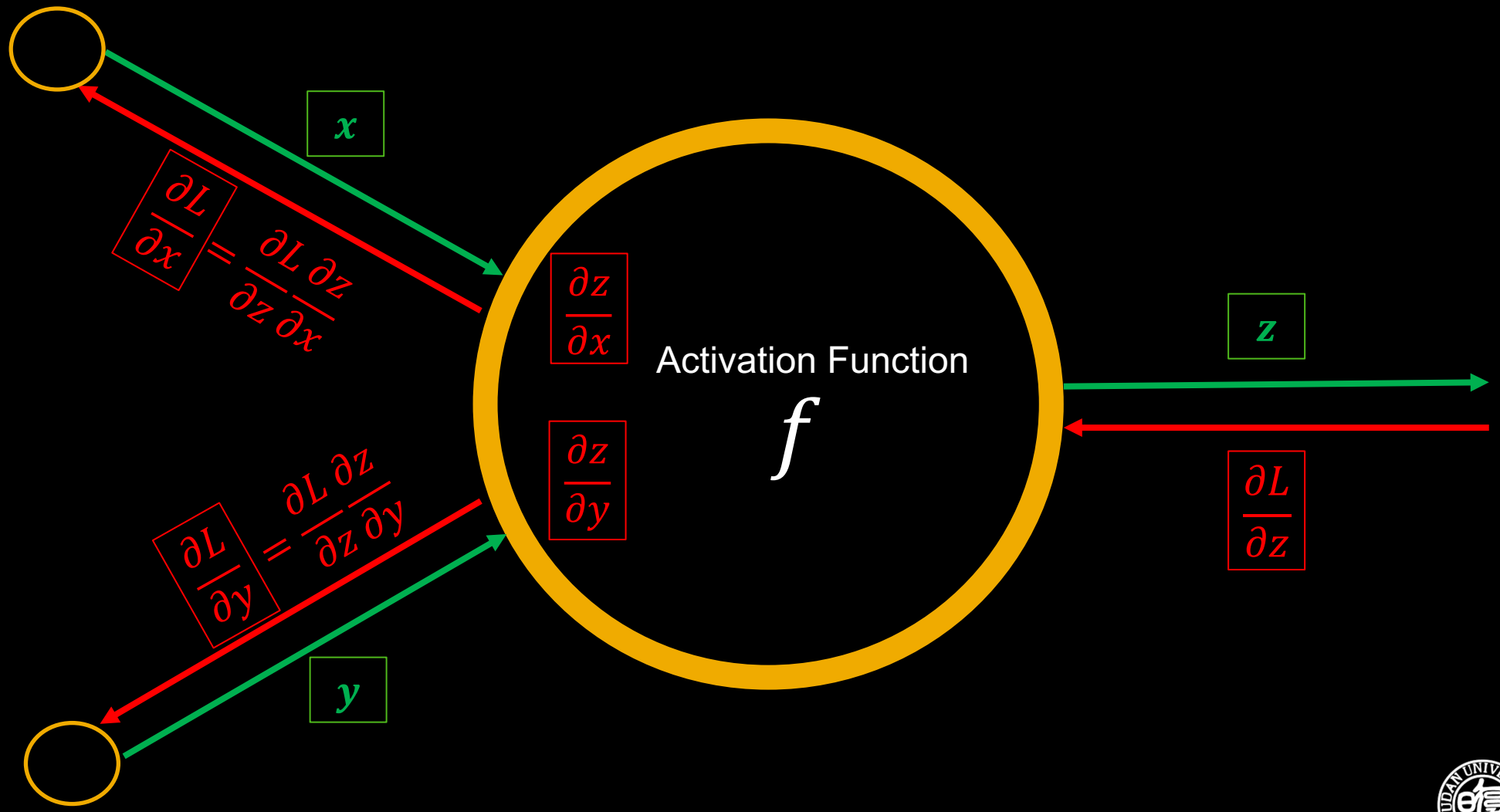
$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

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

$$\Delta s \to \Delta y \to \Delta z$$
$$\Delta s \to \Delta x \to \Delta z$$

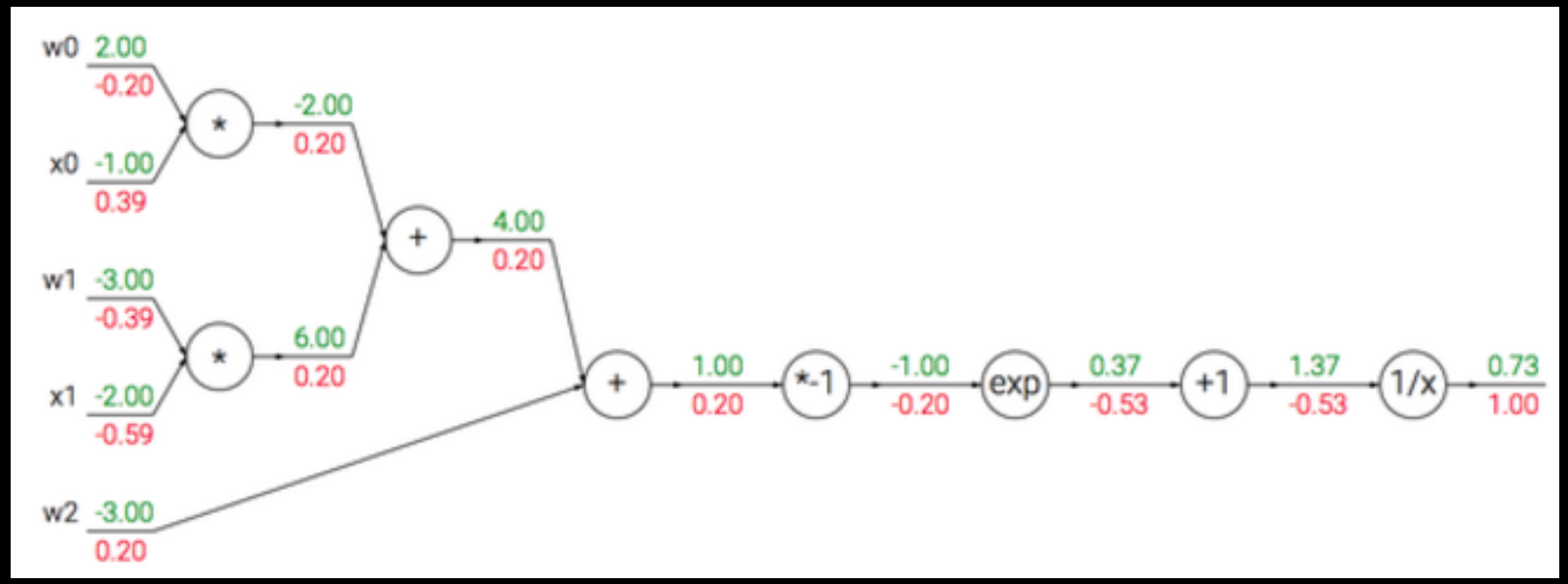
$$\frac{dz}{ds} = \frac{dz}{dx}\frac{dx}{ds} + \frac{dz}{dy}\frac{dy}{ds}$$

Activation Function: Back Propagation (BP)



Back Propagation Example

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



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

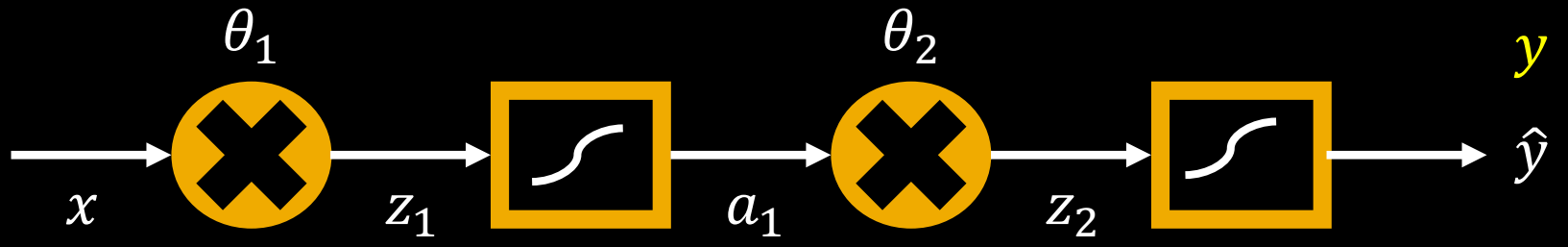
$$f(x) = c + x \qquad \frac{d}{dx} f(x) = 1$$

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

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

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

Back Propagation（永远求偏导）



Cost Function (1 row):

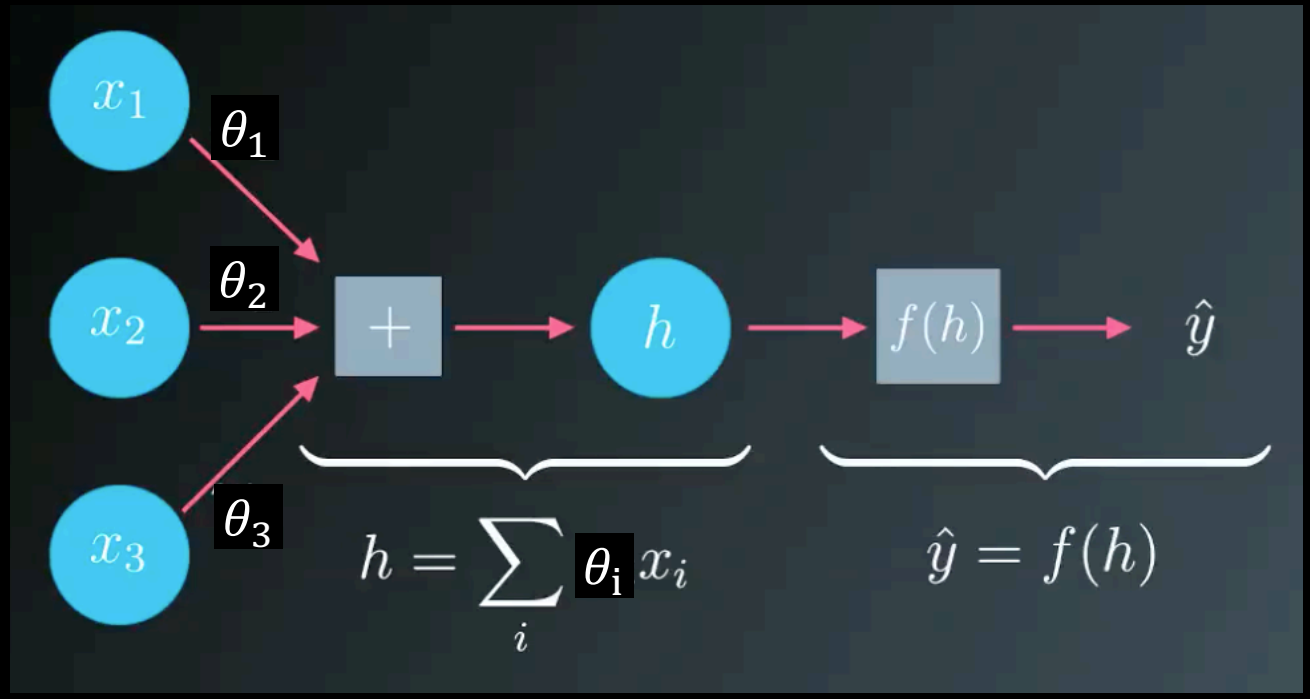
$$\mathcal{J} = \frac{1}{2}(y - \hat{y})^2$$

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

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

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

反向传播与梯度下降



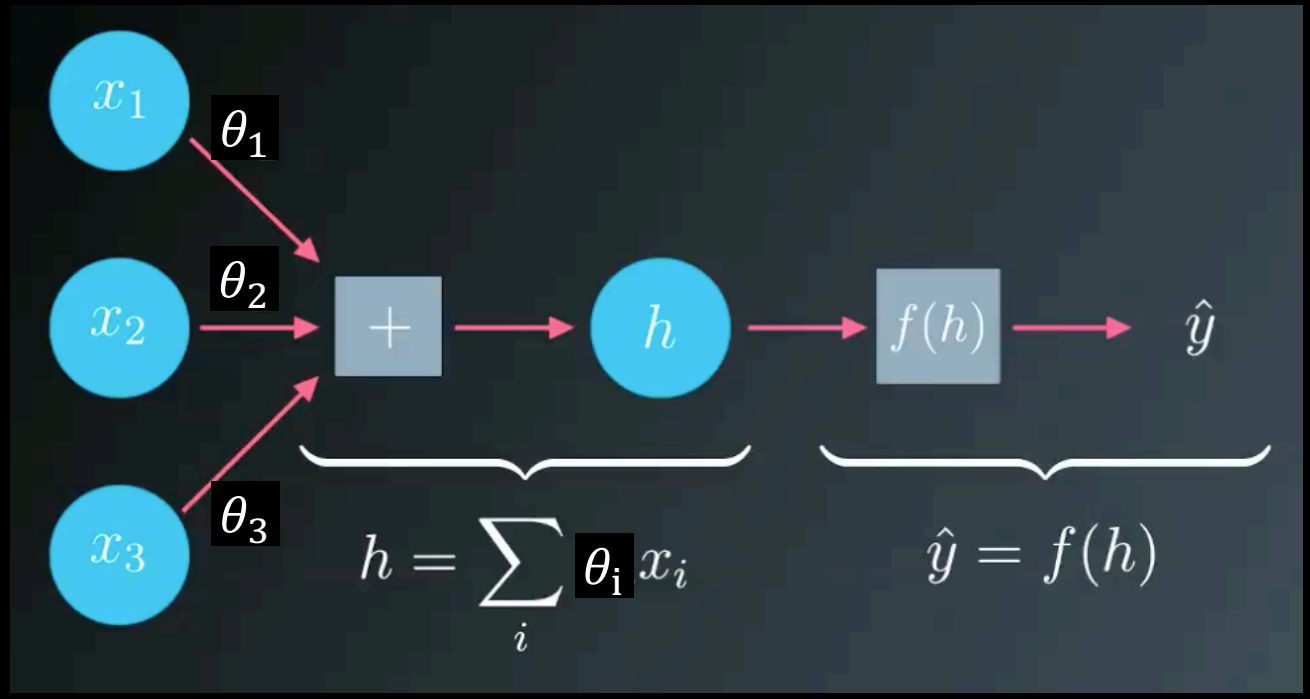
$$\mathcal{J} = \frac{1}{2}\sum_i (y^{(i)} - \hat{y}^{(i)})^2$$

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

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：

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

反向传播与梯度下降



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

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

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

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

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

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

反向传播与梯度下降

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

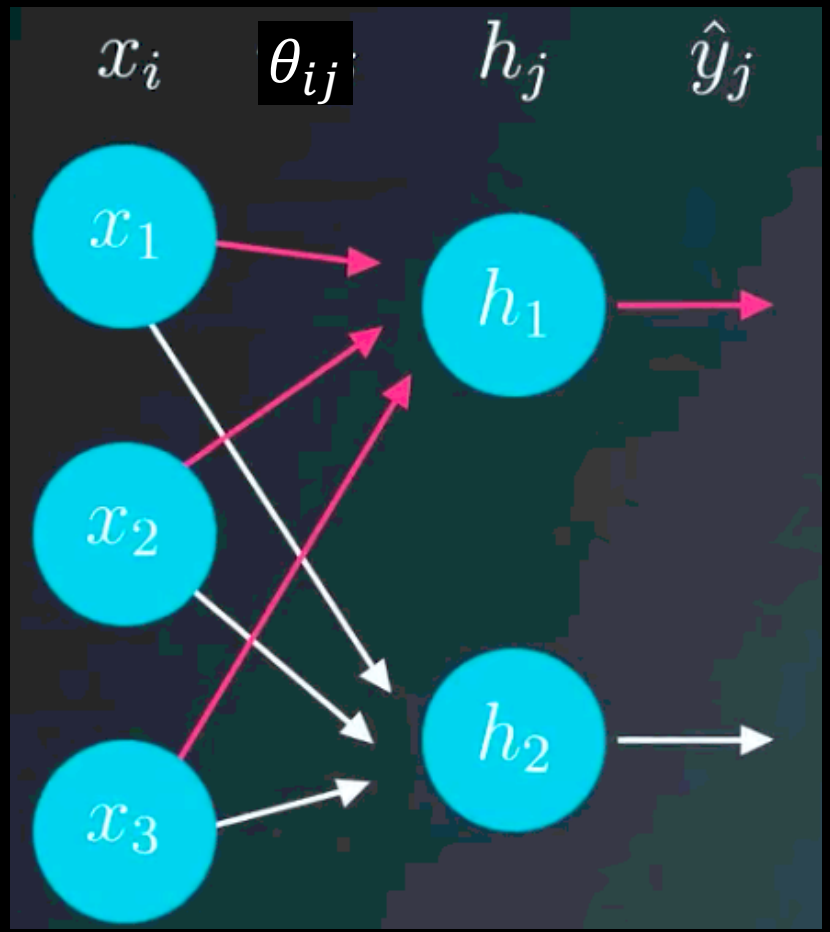
为了方便，我们定义：$\delta = (y - \hat{y}) f'(h)$

* 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）

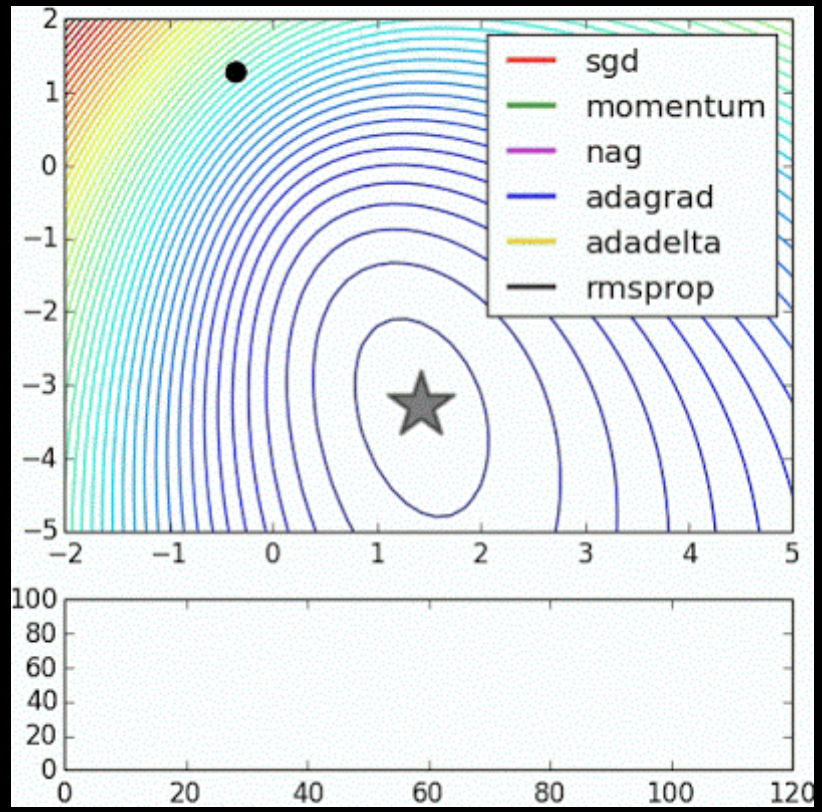
```python
import numpy as np
def sigmoid(x):
    return 1/(1+np.exp(-x))

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



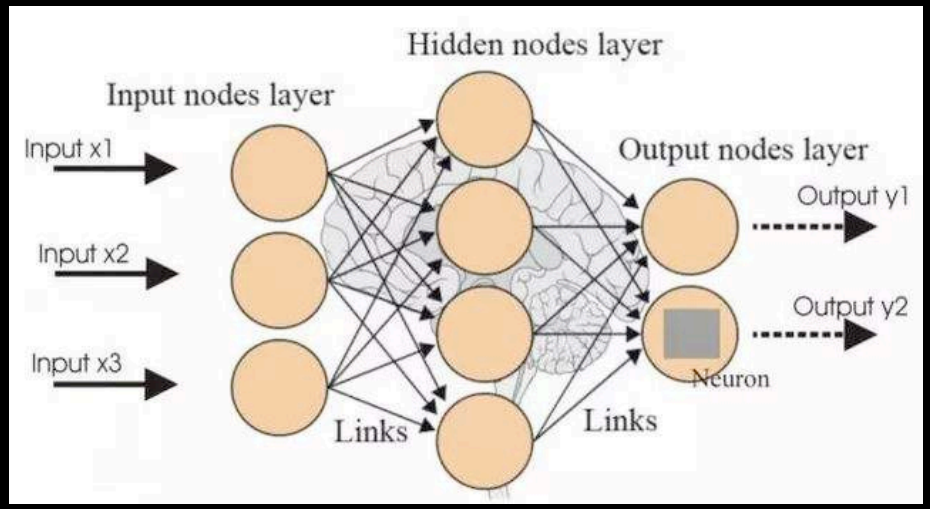
logistic regression on noisy moons

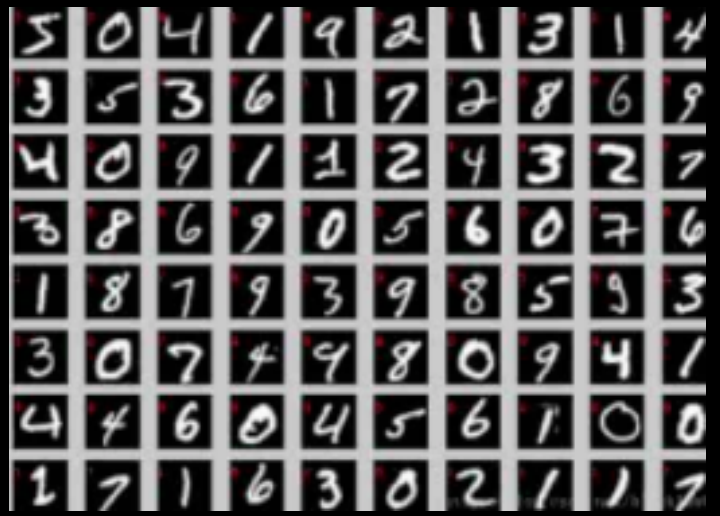
1st Example of Keras

```
from keras.models import Sequential
from keras.layers import Dense
model = Sequential()
model.add(Dense(10, input_dim=20, kernel_initializer = 'random_uniform'))
```

- Sequential: the simplest model, a sequential Keras model is a linear pipeline (a stack) of neural networks layer.
- model.add define a single layer with 10 artificial neurons, and it expects 20 features
- Each neuron can be initialized with *random_uniform* (-0.05, 0.05).
- Weights can also be initialized with *random_normal* and *zero.*
- Dense means each neuron in a layer is connected to all neurons located in the previous layer and to all the neurons in the following layer.

Multilayer Perceptron (MLP) & MNIST





- Perceptron are usually single layer.
- *MNIST*, a database of handwritten digits (Gray Scale) made up of 60,000 examples and a test set of 10,000 examples. Each image contains 28x28 pixels.
- The training examples are annotated by humans with correct answer (Supervised Learning).

Sample Code 1 (Base Line)

```
# Imports
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout
from keras.optimizers import SGD
```

```
# Hyperparameters
NB_EPOCHS = 10
BATCH_SIZE = 128
NB_CLASSES = 10
VALIDATION_SPLIT = 0.2
```

```
# Load Data
# X_train is 60000 rows of 28*28 values, X_test is 10000 rows of 28*28
(X_train, y_train),(X_test, y_test) = mnist.load_data()
```

```
# flatten and convert to float32 for normalization
X_train = X_train.reshape(60000, 784)
X_test = X_test.reshape(10000, 784)
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
```

Sample Code 2 (Base Line)

```
# Normalize
X_train /= 255
X_test /= 255
```

```
# Convert class into one-hot encoding
Y_train = keras.utils.to_categorical(y_train, NB_CLASSES)
Y_test = keras.utils.to_categorical(y_test, NB_CLASSES)
```

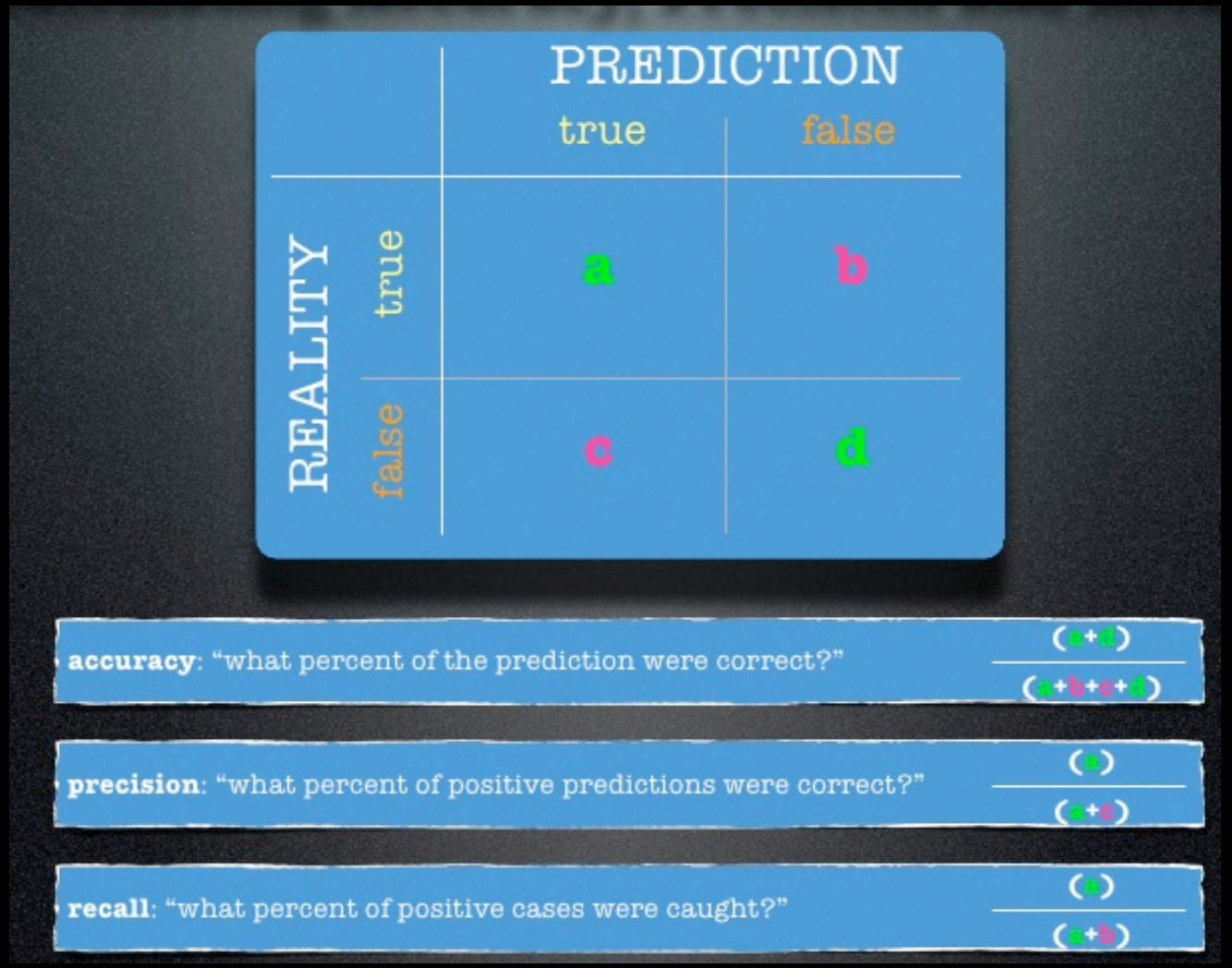
```
# Build the model
model = Sequential()
model.add(Dense(NB_CLASSES, input_shape=(784,)))
model.add(Activation('softmax'))
```

- Softmax is the generalization of logistic regression!
- Softmax squash a k-dimensional vector of arbitrary value into a same dimensional real value vectors in the range of (0,1) and add up to 1.

```
model.compile(loss='categorical_crossentropy',optimizer=SGD(), metrics=['accuracy'])
```

- Cost / Objective Functions
  - MSE
  - Binary Cross-entropy
  - Categorical Cross-entropy
- Evaluation Metrics:
  - Accuracy
  - Precision / Recall

Accuracy / Precision / Recall



Sample Code 2 (Base Line)

```
history = model.fit(X_train, Y_train, batch_size = BATCH_SIZE, epochs = NB_EPOCHS, verbose = 1,
                    validation_data=(X_test,Y_test))
```

```
score = model.evaluate(X_test, Y_test, verbose=0)
print(score)
```

- Epochs: the number of times the model is exposed to the training set, at each iteration, the optimizers tries to adjust the weights to minimize the objective function.
- Batch Size: The number of training instance observed before the optimizer performs a weight update.

Sample Code 2 (Base Line)

```
Epoch 95/100
60000/60000 [==============================] - 0s - loss: 0.2856 - acc: 0.9207 - val_loss: 0.2818 - val_acc: 0.9205
Epoch 96/100
60000/60000 [==============================] - 0s - loss: 0.2854 - acc: 0.9205 - val_loss: 0.2819 - val_acc: 0.9217
Epoch 97/100
60000/60000 [==============================] - 0s - loss: 0.2851 - acc: 0.9210 - val_loss: 0.2819 - val_acc: 0.9207
Epoch 98/100
60000/60000 [==============================] - 0s - loss: 0.2849 - acc: 0.9210 - val_loss: 0.2815 - val_acc: 0.9211
Epoch 99/100
60000/60000 [==============================] - 0s - loss: 0.2847 - acc: 0.9209 - val_loss: 0.2814 - val_acc: 0.9214
Epoch 100/100
60000/60000 [==============================] - 0s - loss: 0.2844 - acc: 0.9213 - val_loss: 0.2812 - val_acc: 0.9208
```

Hidden layer Added (Relu)

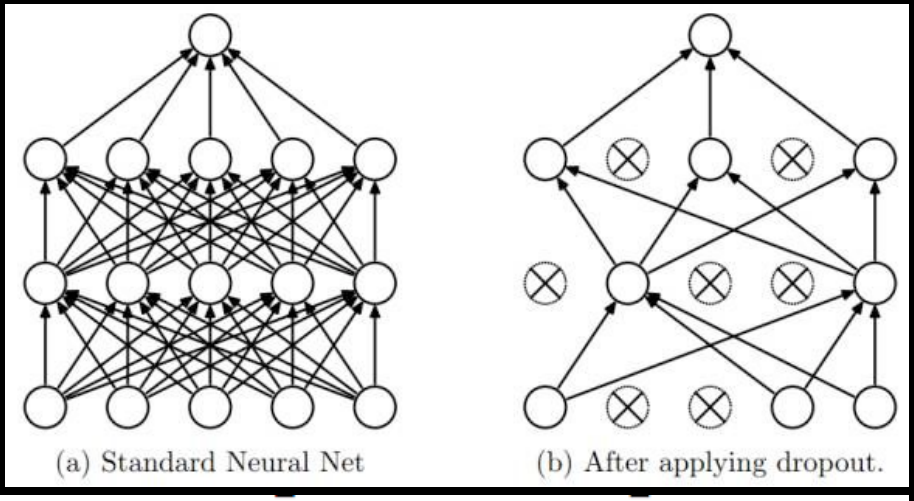
```
# Build the model
model = Sequential()

model.add(Dense(512, activation='relu', input_shape=(784,)))

model.add(Dense(512, activation='relu'))

model.add(Dense(10, activation='softmax'))
```

- Add 2 hidden layers
- Reduce the Epoch to 20

```
Epoch 15/20
60000/60000 [==============================] - 4s - loss: 0.1781 - acc: 0.9501 - val_loss: 0.1754 - val_acc: 0.9489
Epoch 16/20
60000/60000 [==============================] - 4s - loss: 0.1716 - acc: 0.9515 - val_loss: 0.1714 - val_acc: 0.9505
Epoch 17/20
60000/60000 [==============================] - 4s - loss: 0.1657 - acc: 0.9533 - val_loss: 0.1654 - val_acc: 0.9527
Epoch 18/20
60000/60000 [==============================] - 4s - loss: 0.1602 - acc: 0.9548 - val_loss: 0.1599 - val_acc: 0.9524
Epoch 19/20
60000/60000 [==============================] - 4s - loss: 0.1547 - acc: 0.9566 - val_loss: 0.1557 - val_acc: 0.9538
Epoch 20/20
60000/60000 [==============================] - 4s - loss: 0.1497 - acc: 0.9575 - val_loss: 0.1539 - val_acc: 0.9542
```

Hidden layer Added (Dropout)

```
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(10, activation='softmax'))
```



(a) Standard Neural Net (b) After applying dropout.

```
Epoch 15/20
60000/60000 [==============================] - 7s - loss: 0.1700 - acc: 0.9499 - val_loss: 0.1384 - val_acc: 0.9572
Epoch 16/20
60000/60000 [==============================] - 7s - loss: 0.1620 - acc: 0.9521 - val_loss: 0.1343 - val_acc: 0.9580
Epoch 17/20
60000/60000 [==============================] - 7s - loss: 0.1554 - acc: 0.9539 - val_loss: 0.1290 - val_acc: 0.9589
Epoch 18/20
60000/60000 [==============================] - 7s - loss: 0.1493 - acc: 0.9564 - val_loss: 0.1244 - val_acc: 0.9608
Epoch 19/20
60000/60000 [==============================] - 7s - loss: 0.1424 - acc: 0.9580 - val_loss: 0.1207 - val_acc: 0.9616
Epoch 20/20
60000/60000 [==============================] - 7s - loss: 0.1383 - acc: 0.9598 - val_loss: 0.1166 - val_acc: 0.9625
```

Regularization Implementation

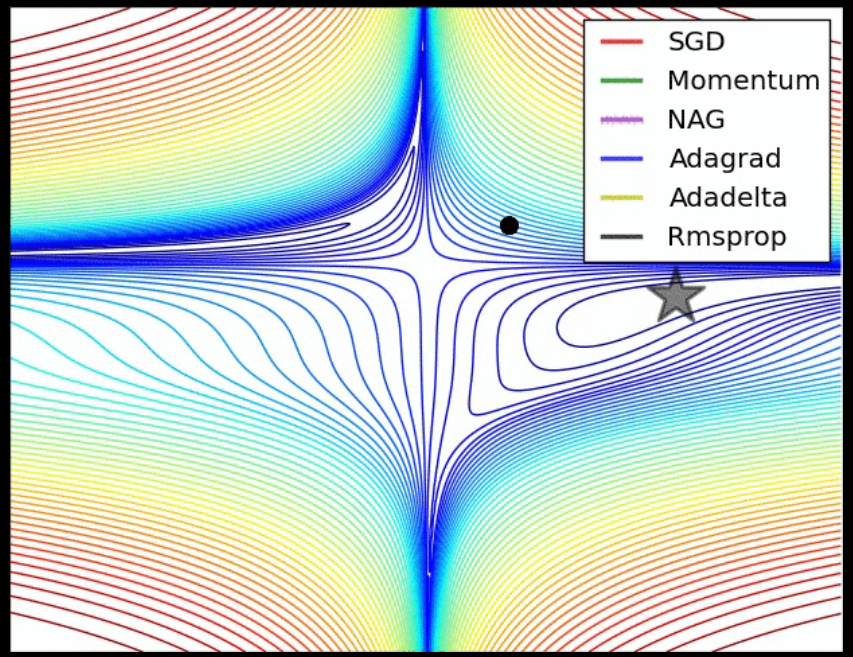
```
from keras import regularizers
model.add(Dense(64, input_dim = 64, kernel_regularizer=regularizers.l2(0.01)))
```

Advance Optimizer (RMSprop & Adam)

```
from keras.optimizers import RMSprop, Adam
```

- RMSprop and Adam includes the concept of *Momentum.*

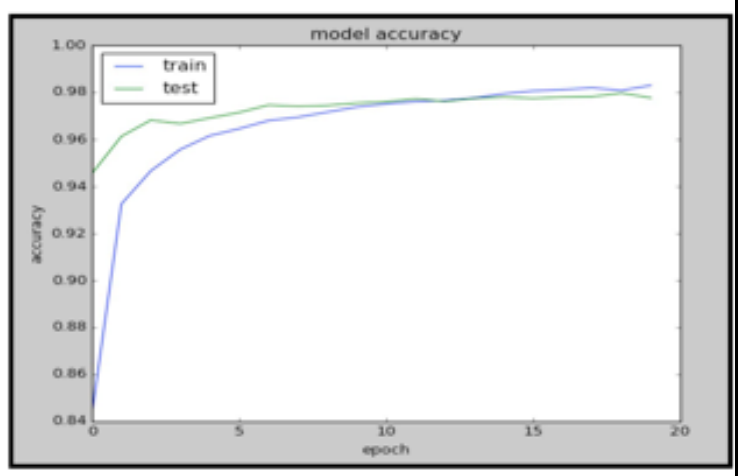
```
model.compile(loss='categorical_crossentropy', optimizer=RMSprop(), metrics=['accuracy'])
```

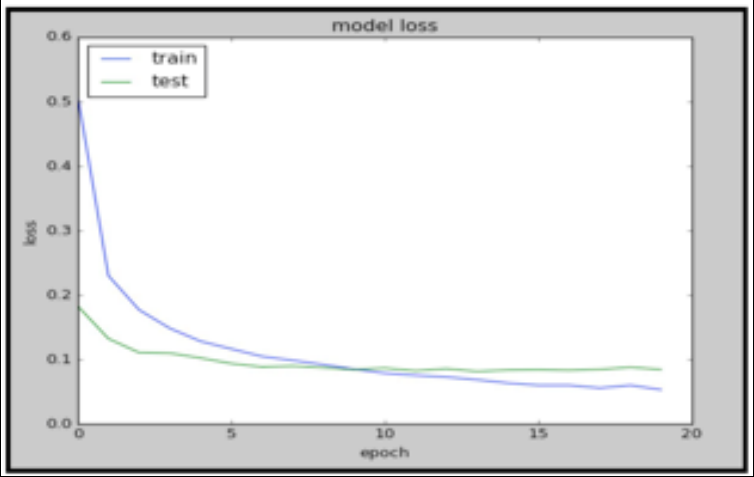


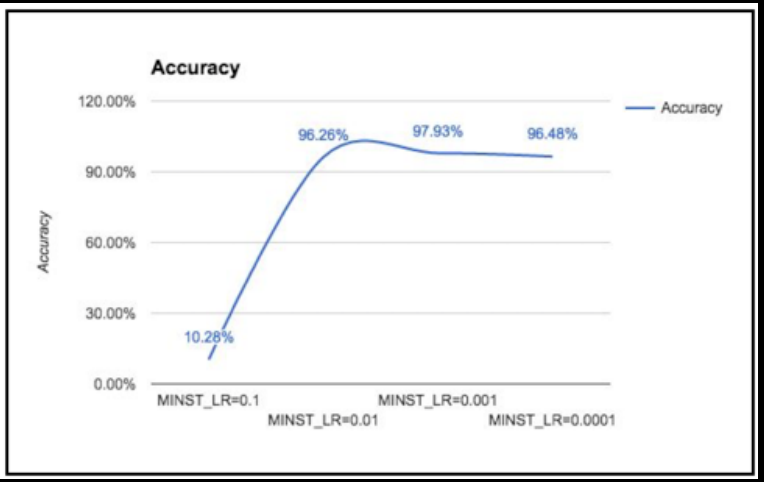

```
Epoch 1/20
60000/60000 [==============================] - 8s - loss: 0.2557 - acc: 0.9217 - val_loss: 0.1140 - val_acc: 0.9652
Epoch 2/20
60000/60000 [==============================] - 8s - loss: 0.1080 - acc: 0.9688 - val_loss: 0.1025 - val_acc: 0.9724
```

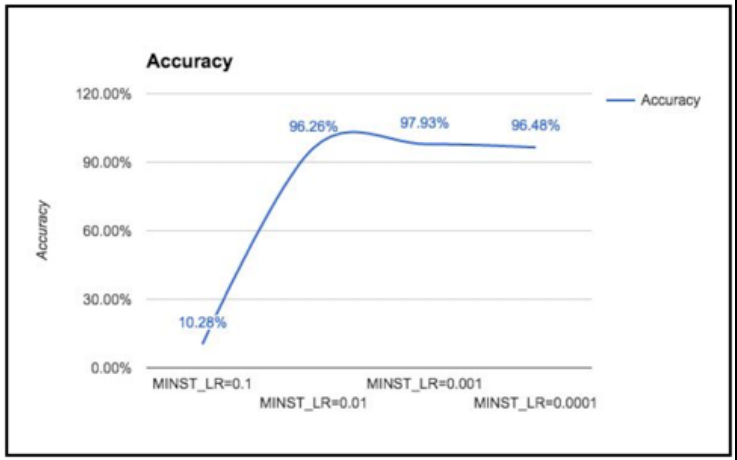
```
Epoch 19/20
60000/60000 [==============================] - 7s - loss: 0.0429 - acc: 0.9917 - val_loss: 0.1144 - val_acc: 0.9840
Epoch 20/20
60000/60000 [==============================] - 8s - loss: 0.0382 - acc: 0.9927 - val_loss: 0.1089 - val_acc: 0.9846
```

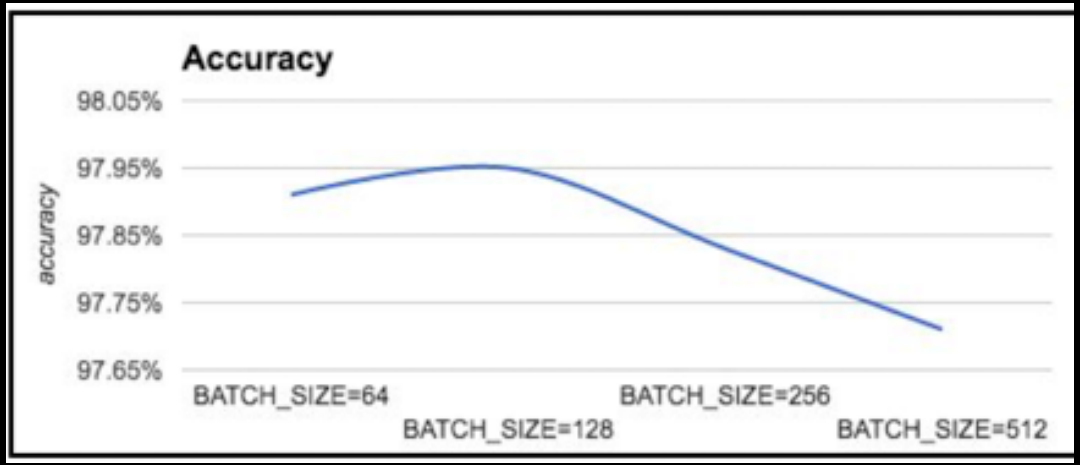
Hyper parameters











Hello World!

```
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')
```

SAP

復旦大學

Logistic/Softmax Regression

```python
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