TensorFlow和Keras介绍

雍宾宾

yongbb@lzu.edu.cn

深度学习库

TensorFlow是谷歌研发的人工智能学习系统。Tensor（张量）意味着N维数组，Flow（流）意味着基于数据流图的计算，TensorFlow为张量从流图的一端流动到另一端计算过程。

Keras是一个高层神经网络API，Keras由纯Python编写，以Tensorflow、Theano以及CNTK为后端。

旧版TensorFlow和Keras 安装

```
conda install keras==2.2.4
conda install tensorflow==1.13.1

 >>> import tensorflow
 >>> import keras
 #无报错即可
 >>> keras.__version__
'2.2.4’
 >>> tf.__version__
'1.13.1’
```

https://colab.research.google.com/

新版TensorFlow和Keras 安装

```
conda create -name tf2 python=3.7
conda activate tf2
pip install tensorflow==2.3 #export LD_LIBRARY_PATH="/usr/local/cuda-10.2/lib64"

>>> import tensorflow as tf
>>> tf.__version__
 '2.3.0'
>>> tf.test.gpu_device_name()
'/device:GPU:0'


https://colab.research.google.com/
```

TensorFlow API – 数据类型

```
import tensorflow as tf
```

1.标量:
```
a = tf.constant(1.2)
type(a), tf.is_tensor(a)
```

2.向量:
```
b = tf.constant([1,2.,3.3])
b.numpy(), b.shape
```

3.矩阵:
```
c = tf.constant([[1,2],[3,4]])
c.shape
```

4.字符串:
```
d = tf.constant('Hello, DL.')
tf.strings.lower(d)
```

5.布尔型:
```
e = tf.constant([True, False])
```

精度:
```
tf.constant(np.pi, dtype=tf.float32)
```

tf.int16, tf.int32, tf.int64, tf.float16, tf.float32, tf.float64

```
if a.dtype != tf.float32:
    a = tf.cast(a,tf.float32)
```

TensorFlow API – 待优化张量

List转张量：　　a = tf.Variable([[1,2],[3,4]])

np.array转张量：　tf.convert_to_tensor(np.array([[1,2.],[3,4]])) tf.float64

全0向量矩阵：　　tf.zeros([2,2,2]), tf.zeros_like(a)

全1向量矩阵：　　tf.ones([2,2,2]), tf.ones_like(a)

全99向量矩阵：　　tf.fill([2,2], 99)

TensorFlow API – 参数张量

正态随机分布：

```
tf.random.normal([2,2])
tf.random.normal([2,2], mean=1,stddev=2)
tf.random.truncated_normal([784, 256], stddev=0.1)
#[ mean - 2 * stddev, mean + 2 * stddev ]
```

均匀分布：

```
tf.random.uniform([2,2])
tf.random.uniform([2,2],maxval=10)
tf.random.uniform([2,2],maxval=100,dtype=tf.int32)
```

序列：

```
tf.range(10)
tf.range(10,delta=2)
tf.range(1,10,delta=2)
```

TensorFlow API – 参数张量

切片：

```
x = tf.random.normal([4,32,32,3])
x[0]
x[0,:,:,1].shape
```

改变维度：

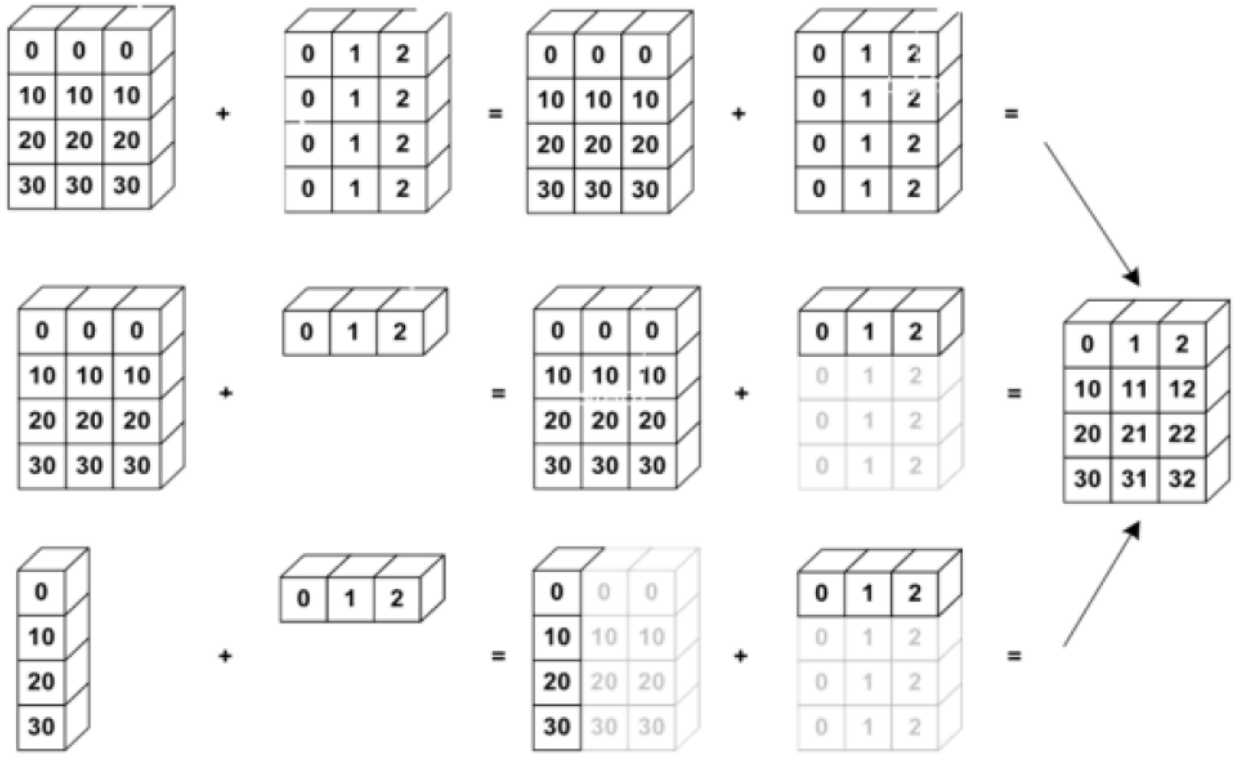
```
x=tf.range(12)
x=tf.reshape(x,[3,4])
```

交换维度：

```
x = tf.random.normal([2,32,32,3])
tf.transpose(x,perm=[0,3,1,2])
```

广播机制：

```
x = tf.random.normal([2,4])
w = tf.random.normal([4,3])
b = tf.random.normal([3])
y = x@w+b
```



TensorFlow API – 基本运算

加减乘除： tf.add, tf.subtract, tf.multiply, tf.divide

\+ - * / //(整除) %(余数)

乘方：

```
x = tf.range(4)
tf.pow(x,2),tf.square(x)
x**2
tf.sqrt(tf.cast(x,dtype=tf.float32))
```

矩阵乘法：

```
a = tf.random.normal([23,32])
b = tf.random.normal([32,2])
a@b, tf.matmul(a, b)

a = tf.random.normal([4,3,23,32])
b = tf.random.normal([4,3,32,2])
a@b
```

对数e：

```
tf.math.log(tf.cast([0,1,np.e],tf.float32))
```

<tf.Tensor: shape=(3,), dtype=float32, numpy=array([-inf, 0. , 0.99999994], dtype=float32)>

TensorFlow API – 拼接

合并样本：

```
a = tf.random.normal([4,35,8]) # 4个班级
b = tf.random.normal([6,35,8]) # 6个班级
tf.concat([a,b],axis=0) # 合并成绩册

a = tf.random.normal([10,35,3]) # 3门课程
b = tf.random.normal([10,35,5]) # 5门课程
tf.concat([a,b],axis=2) # 在科目维度拼接
```

分割样本：

```
x=tf.random.normal([6000,28,28,1])
x_train,x_test=tf.split(x, num_or_size_splits=[4000,2000], axis=0)
```

TensorFlow API – 数据统计

向量范数：

```
x = tf.ones([2,2])
tf.norm(x)
tf.norm(x,ord=2)
tf.norm(x,ord=3)
```

均值和：

```
x = tf.random.normal([4,10])
tf.reduce_max(x,axis=1)
tf.reduce_min(x,axis=1)
tf.reduce_mean(x,axis=1)
tf.reduce_max(x),tf.reduce_min(x),tf.reduce_mean(x), tf.reduce_sum(x)#全局统计

out  = tf.random.uniform([4,10])    #随机模拟网络输出
y    = tf.constant([2,3,2,0])       # 随机构造样本真实标签
y    = tf.one_hot(y, depth=10)      # one-hot 编码
Loss = tf.keras.losses.mse(y, out)  # 计算每个样本的MSE
loss = tf.reduce_mean(loss)         # 平均MSE
print(loss)
```

TensorFlow API – 数据统计

概率输出：

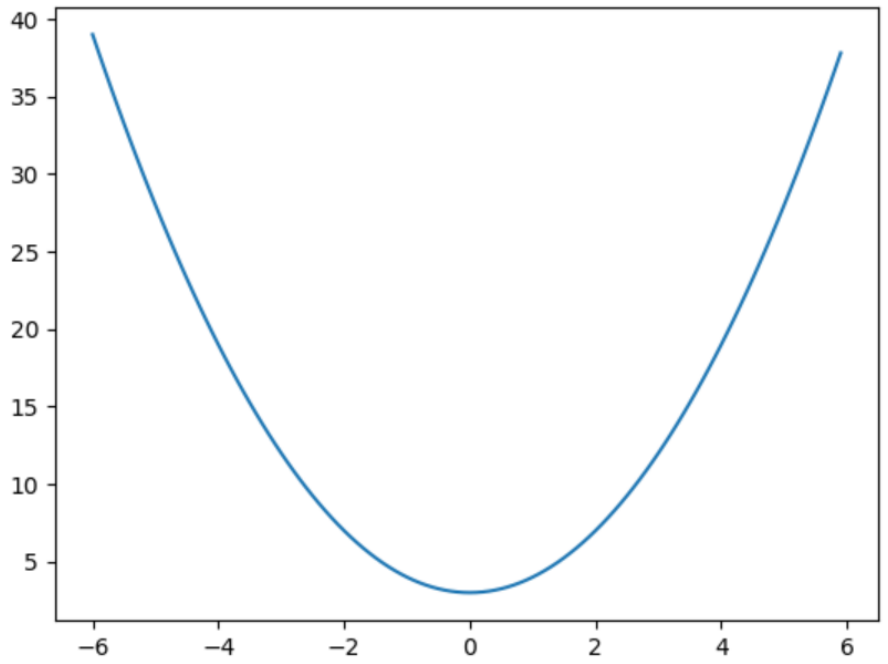
```
out = tf.random.normal([2,10])
out = tf.nn.softmax(out, axis=1)
pred = tf.argmax(out, axis=1)
y = tf.random.uniform([2],dtype=tf.int64,maxval=10)
out = tf.equal(pred,y)
out = tf.cast(out, dtype=tf.float32)
correct = tf.reduce_sum(out)
```

TensorFlow梯度

```
for step in range(200):# loop 200 times
    with tf.GradientTape(persistent=True) as tape: # 默认只算一次梯度
        tape.watch([w1,b1,w2,x]) # add to gradient list
        out = tf.sigmoid((x@w1+b1))@w2 # feedforward
        loss = tf.reduce_sum(tf.square(out - y))# 计算每个样本的MSE
        grads_x = tape.gradient(loss, x)
        grads_x2 = tape.gradient(grads_x, x)

    grads = tape.gradient(loss, [w1,b1,w2])
    lr = 0.01
    w1 = w1 - lr * grads[0]
    b1 = b1 - lr * grads[1]
    w2 = w2 - lr * grads[2]
```

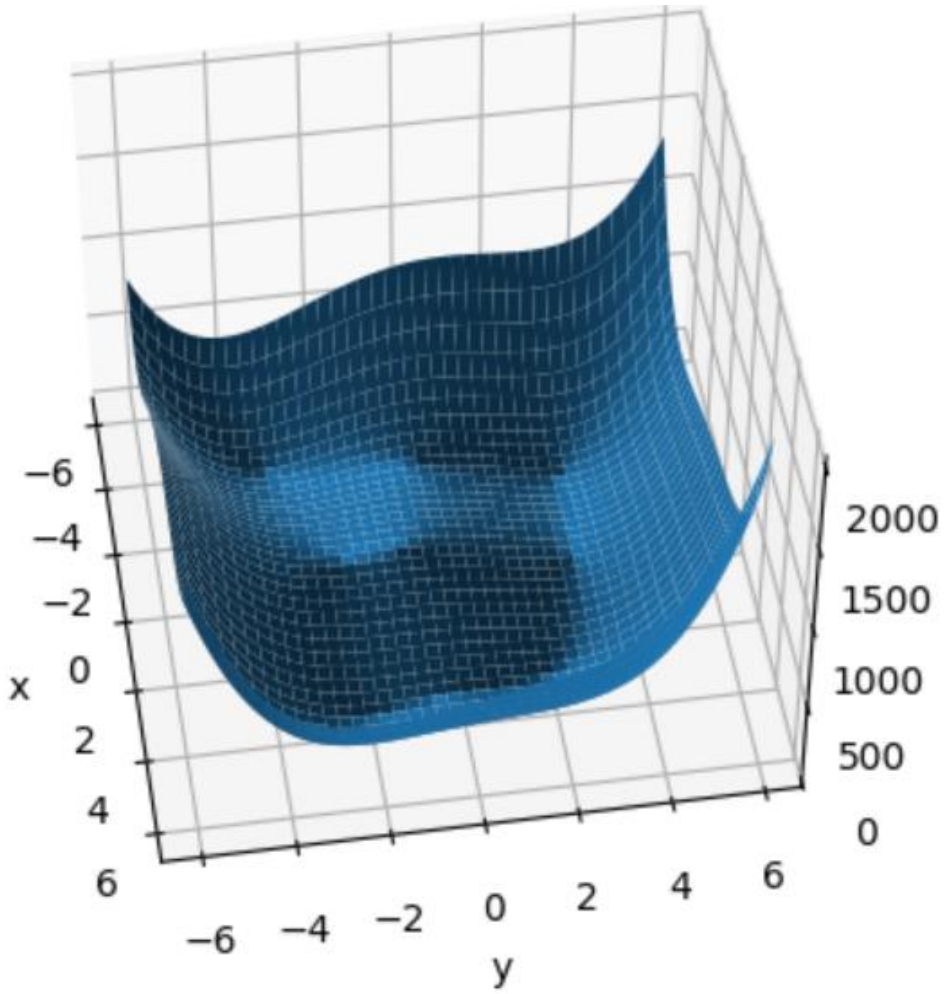
TensorFlow梯度下降实践




```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
x = np.arange(-6, 6, 0.1)
y = x**2 + 3
plt.plot(x,y)
plt.show()
x = tf.constant([6., 0.]) # init
for step in range(200):# loop 200 times
    with tf.GradientTape() as tape: # gradient
        tape.watch([x]) # add to gradient list
        y = x**2+3 # feedforward
    grads = tape.gradient(y, [x])[0]
    x -= 0.01*grads  # lr=0.01
    if step % 20 == 19: # print min
        print ('step {}: x = {}, f(x) = {}'.format(step,
x.numpy(), y.numpy()))
```

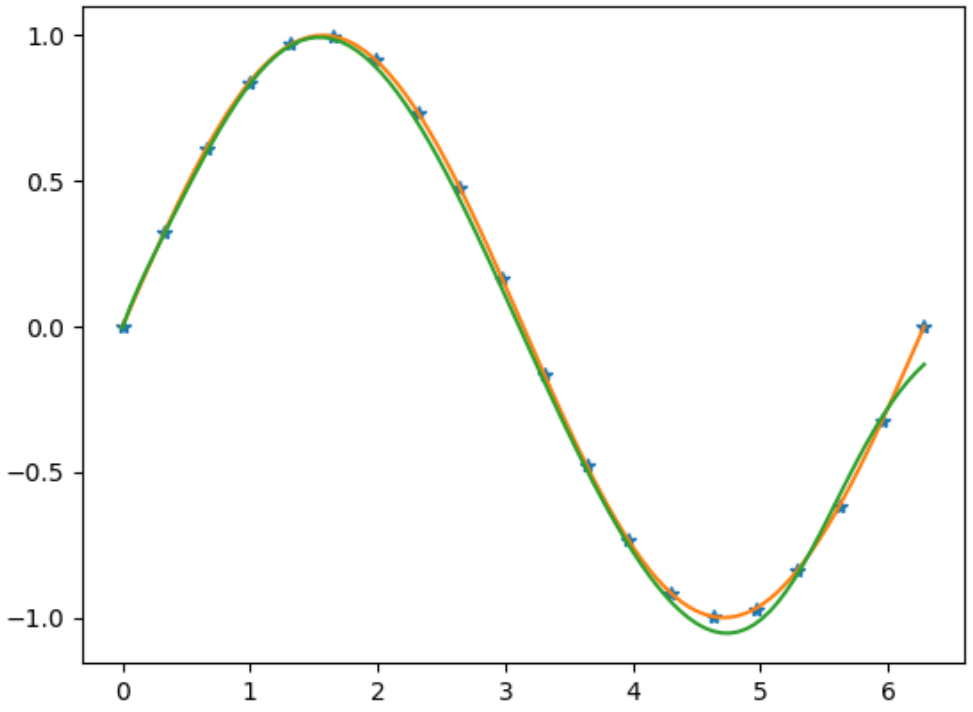
TensorFlow梯度下降实践

$$f(x,y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2$$




```python
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
def himmelblau(x): # himmelblau 函数实现
    return (x[0] ** 2 + x[1] - 11) ** 2 + (x[0] + x[1] ** 2 - 7) ** 2
x = np.arange(-6, 6, 0.1)
y = np.arange(-6, 6, 0.1)
print('x,y range:', x.shape, y.shape)
X, Y = np.meshgrid(x, y) # 生成x-y 平面采样网格点，方便可视化
print('X,Y maps:', X.shape, Y.shape)
Z = himmelblau([X, Y]) # 计算网格点上的函数值并利
fig = plt.figure('himmelblau')
ax = fig.gca(projection='3d')
ax.plot_surface(X, Y, Z)
ax.view_init(60, -30)
ax.set_xlabel('x')
ax.set_ylabel('y')
plt.show()
x = tf.constant([4., 0.]) # 初始化参数
for step in range(200):# 循环优化200 次
    with tf.GradientTape() as tape: #梯度跟踪
        tape.watch([x]) # 加入梯度跟踪列表
        y = himmelblau(x) # 前向传播
    grads = tape.gradient(y, [x])[0] # 反向传播
    x -= 0.01*grads  # 更新参数,0.01 为学习率
    if step % 20 == 19: # 打印优化的极小值
        print ('step {}: x = {}, f(x) = {}'.format(step, x.numpy(), y.numpy()))
```

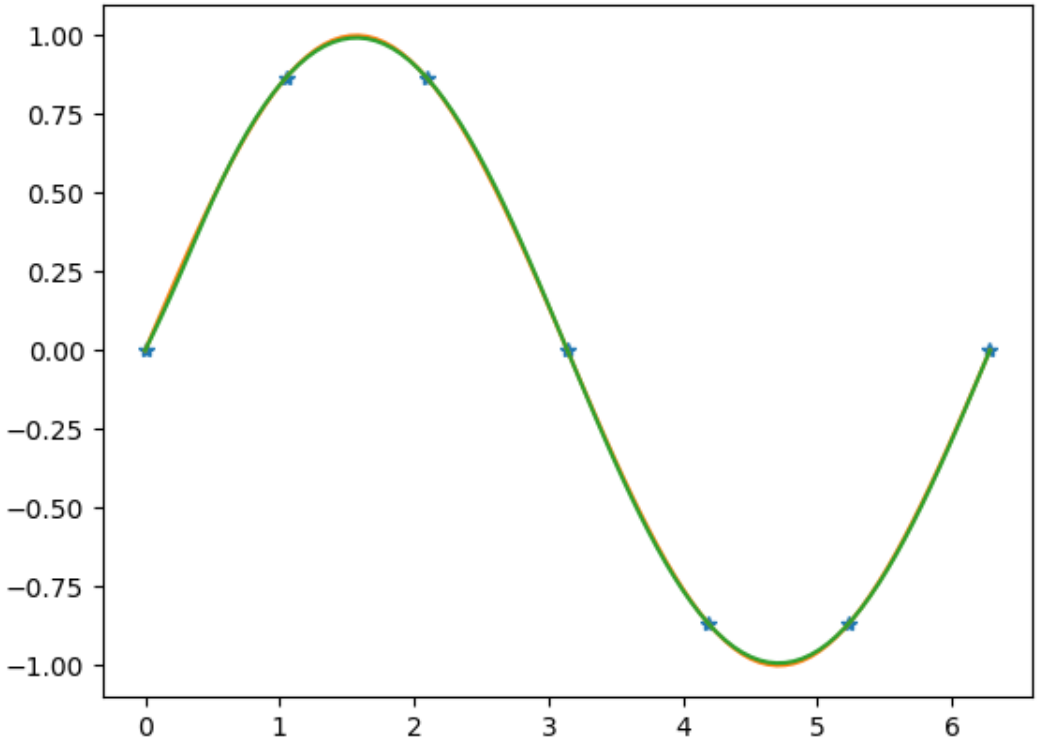
TensorFlow Sin函数拟合



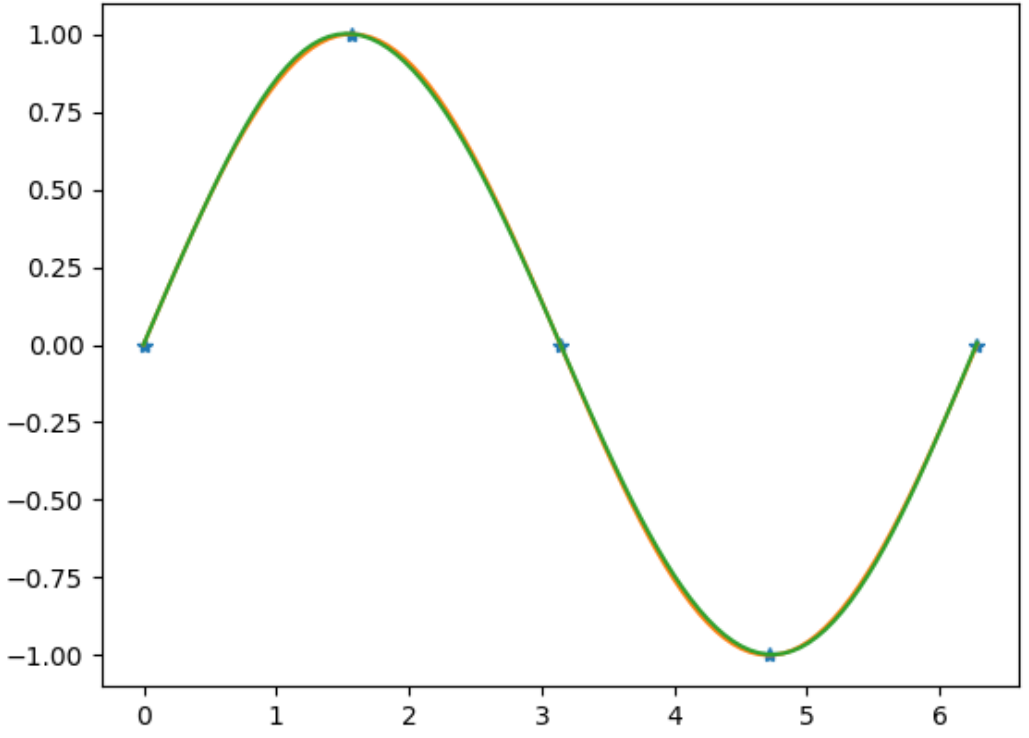

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
x = np.linspace(0, 2*np.pi, 20).reshape((-1,1))
y = tf.Variable(np.sin(x))
x = tf.Variable(x)
w1 = tf.Variable(np.random.random((1,100))*2-1)
b1 = tf.Variable(np.random.random((100,))*2-1)
w2 = tf.Variable(np.random.random((100,1))*2-1)
for step in range(100000):# loop 200 times
    with tf.GradientTape(persistent=True) as tape: # 默认只算一次梯度
        tape.watch([w1,b1,w2]) # add to gradient list
        out = tf.sigmoid((x@w1+b1))@w2 # feedforward
        loss = tf.reduce_sum(tf.square(out - y))  # 计算每个样本的MSE
    grads = tape.gradient(loss, [w1,b1,w2])
    w1 = w1 - 0.01 * grads[0]
    b1 = b1 - 0.01 * grads[1]
    w2 = w2 - 0.01 * grads[2]
    if step % 20 == 19: # print min
        print ('step {}: Loss = {}'.format(step, loss.numpy()))

x_test = np.linspace(0,np.pi*2,100).reshape((-1,1))
x_test = tf.constant(x_test)
y_test = tf.constant(np.sin(x_test))
y_pred = (tf.sigmoid((x_test@w1+b1))@w2).numpy()
plt.plot(x.numpy(),y.numpy(),'*')
plt.plot(x_test,y_test,x_test,y_pred)
plt.show()
```

TensorFlow Sin函数拟合



7个样本



5个样本

Sequential（）

Keras有两种类型的模型，序贯模型（Sequential）和函数式模型（Model），函数式模型应用更为广泛，序贯模型是函数式模型的一种特殊情况。

tensorflow.keras

add ()

```
model.add(Conv2D(6, (5,5), padding='same', activation='relu', input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(16, (5,5), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(120, activation='relu'))
model.add(Dense(84, activation='relu'))
model.add(Dropout(0.5))
#model.add(Dense(1024, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))
```

```
Dense()          #全连接层
Convolution2D()  #卷积层
MaxPooling2D()   #池化层
LSTM()           #长短记忆层
Activation()     #激活函数
Flatten()        #压扁
```

compile（）

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

Keras可以通过model.compile()进行配置。

optimizer　　# 优化器

loss　　# 损失函数

metrics　　# 指标列表

tensorflow.keras

fit（）

Keras可以通过model.fit()进行模型的训练。

```
model.fit(x_train, y_train, batch_size = 128, epochs = 15,
        verbose=1, validation_data=(x_val, y_val))
```

summary（）

通过model.summary() 可输出模型结构

tensorflow.keras

直接对张量求和

tf.keras.layers.add

```
import tensorflow as tf
input1 = tf.keras.layers.Input(shape=(16,))
x1 = tf.keras.layers.Dense(8, activation='relu')(input1)
input2 = tf.keras.layers.Input(shape=(32,))
x2 = tf.keras.layers.Dense(8, activation='relu')(input2)
added = tf.keras.layers.add([x1, x2])
out = tf.keras.layers.Dense(4)(added)
model = tf.keras.models.Model(inputs=[input1, input2], outputs=out)
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   Connected to
=================================================================
input_1 (InputLayer)         [(None, 16)]              0
_________________________________________________________________
input_2 (InputLayer)         [(None, 32)]              0
_________________________________________________________________
dense (Dense)                (None, 8)                 136       input_1[0][0]
_________________________________________________________________
dense_1 (Dense)              (None, 8)                 264       input_2[0][0]
_________________________________________________________________
add (Add)                    (None, 8)                 0         dense[0][0]
                                                                 dense_1[0][0]
_________________________________________________________________
dense_2 (Dense)              (None, 4)                 36        add[0][0]
=================================================================
Total params: 436
Trainable params: 436
```

tensorflow.keras

组合特征：
串联一个列表的输入张量

tf.keras.layers.concatenate

```
import tensorflow as tf
input1 = tf.keras.layers.Input(shape=(16,))
x1 = tf.keras.layers.Dense(8, activation='relu')(input1)
input2 = tf.keras.layers.Input(shape=(32,))
x2 = tf.keras.layers.Dense(8, activation='relu')(input2)
conatenated = tf.keras.layers.concatenate([x1, x2])
out = tf.keras.layers.Dense(4)(conatenated)
model = tf.keras.models.Model(inputs=[input1, input2], outputs=out)
model.summary()
```

```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 16)]         0
__________________________________________________________________________________________________
input_2 (InputLayer)            [(None, 32)]         0
__________________________________________________________________________________________________
dense (Dense)                   (None, 8)            136         input_1[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 8)            264         input_2[0][0]
__________________________________________________________________________________________________
concatenate (Concatenate)       (None, 16)           0           dense[0][0]
                                                                 dense_1[0][0]
__________________________________________________________________________________________________
dense_2 (Dense)                 (None, 4)            68          concatenate[0][0]
==================================================================================================
Total params: 468
Trainable params: 468
Non-trainable params: 0
```

tensorflow.keras

Sequential() — Model, Sequential

add() — Dense, Convolution, MaxPooling, LSTM, Activation, Flatten

compile()

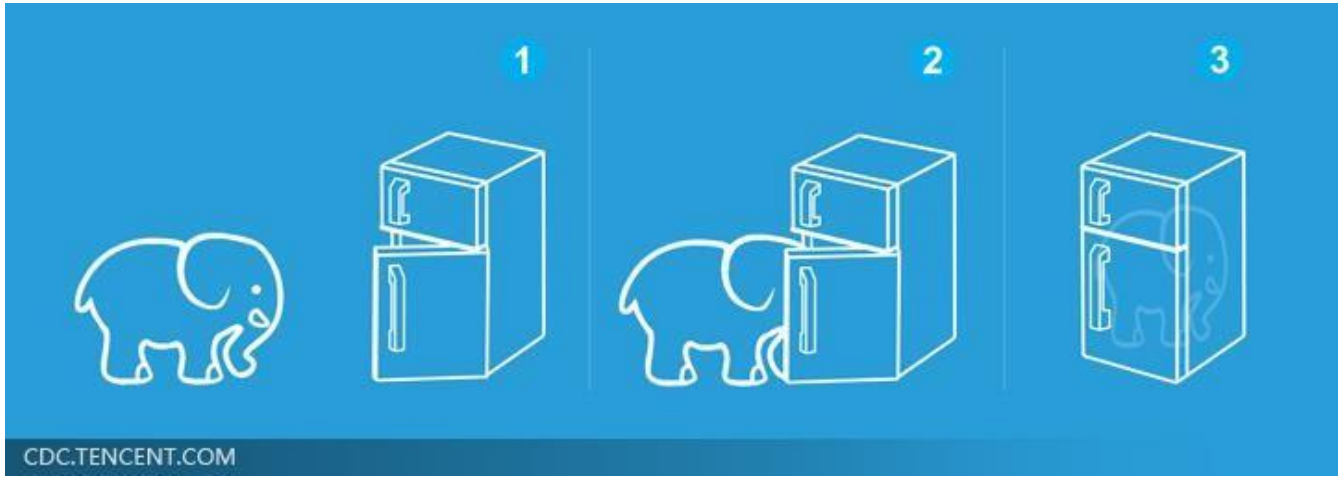
```
model.compile(loss=keras.metrics.categorical_crossentropy,
              optimizer=keras.optimizers.SGD(),
              metrics=['accuracy'])
```

fit()

```
model.fit(x_train, y_train, batch_size = 128, epochs = 15,
          verbose=1, validation_data=(x_val, y_val))
```

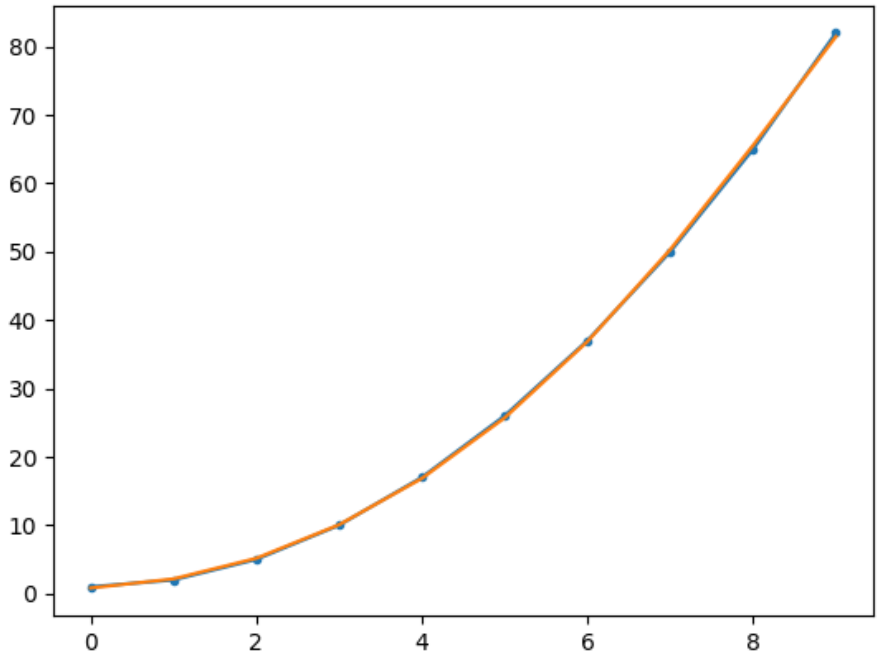
predict() — evaluate

构建神经网络



Fit: y=x**2+1

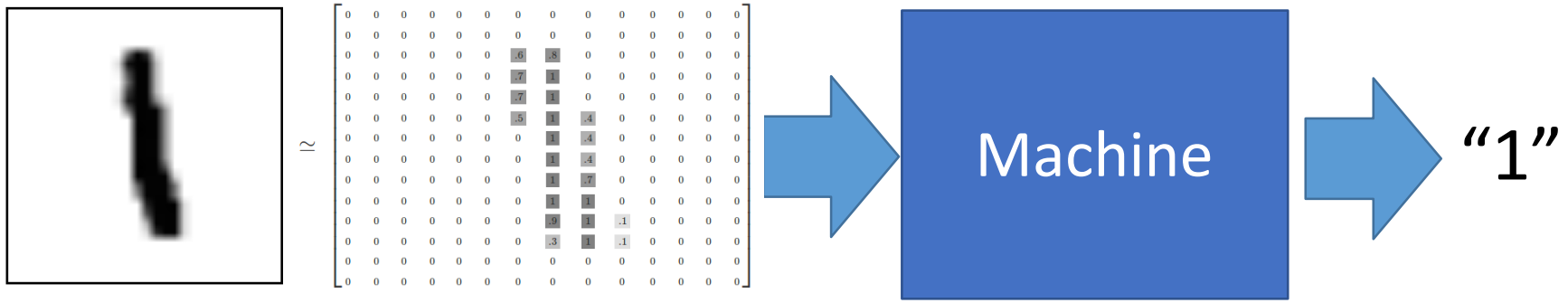
```
import numpy as np
x=np.array(range(10)).reshape((10,1))
y=x*x+1
import keras
from keras.models import Sequential
from keras.layers import Dense
model=Sequential()
model.add(Dense(100,activation='sigmoid',input_shape=(1,)))
model.add(Dense(1))    # 不再指定input_shape
model.compile(loss='mse',optimizer='adam')
model.fit(x,y,epochs=1000)
yp = model.predict(x)
import matplotlib.pyplot as plt
plt.plot(x,y,'.-',x,yp)
plt.show()
```



分类：XOR问题

```
import tensorflow
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import numpy as np
import matplotlib.pyplot as plt
x=np.array([[0,0],[0,1],[1,0],[1,1]])
y=np.array([0,1,1,0])
model=Sequential()
model.add(Dense(2,activation='sigmoid',input_shape=(2,)))
model.add(Dense(1))
model.compile(loss='mse',optimizer=tensorflow.keras.optimizers.SGD(lr=0.1))
history = model.fit(x,y, epochs=10000)
print(model.predict(x))
```

Mnist手写数字

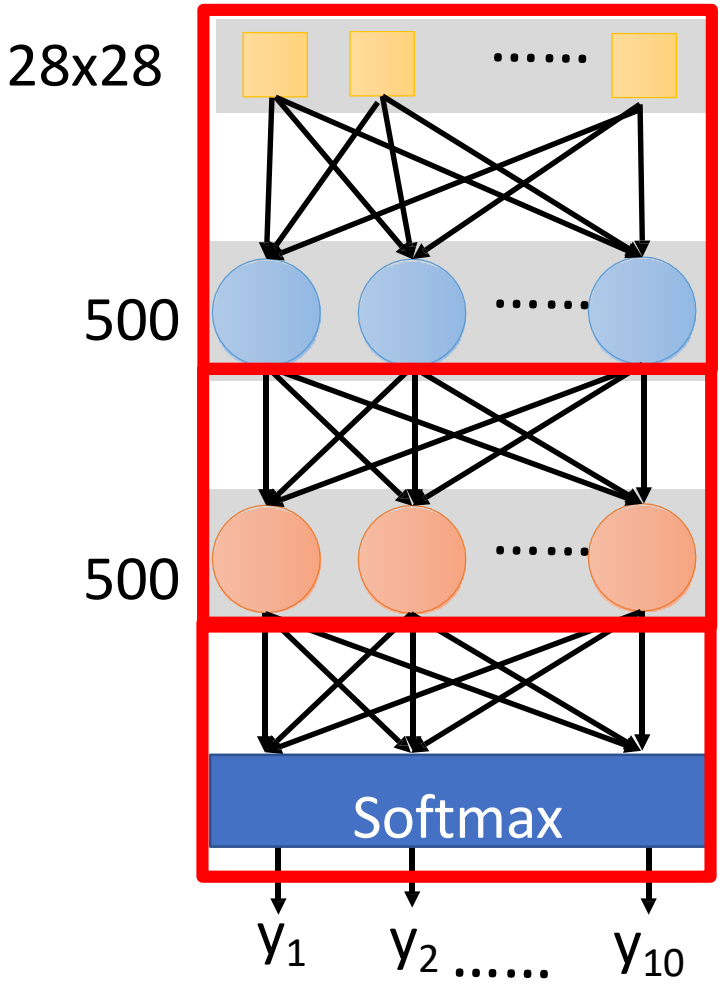


Keras

Step 1: define a set of function

Step 2: goodness of function

Step 3: pick the best function




```
model = Sequential()
```

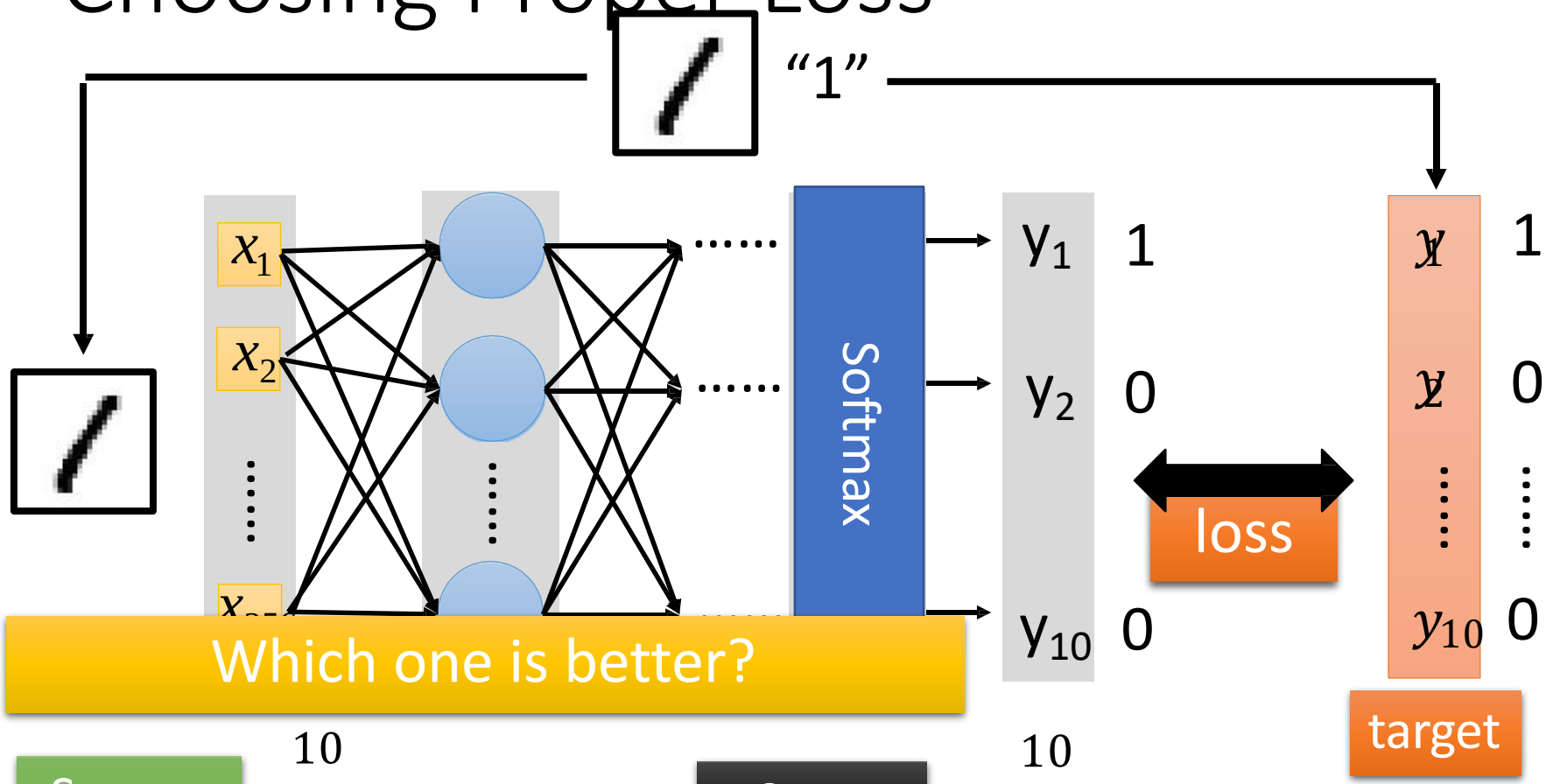
```
model.add( Dense( input_dim=28*28,
                  output_dim=500 ))
model.add( Activation('sigmoid') )
```

```
model.add( Dense( output_dim=500 ) )
model.add( Activation('sigmoid') )
```

model.add(dropout(0.8))

```
model.add( Dense(output_dim=10 ) )
model.add( Activation('softmax') )
```

Choosing Proper Loss



Square Error

$$\sum_{i=1}^{10}(y_i - \hat{y})^2$$

=0

Cross Entropy

$$-\sum_{i=1}^{10}\hat{y}\ln y_i$$

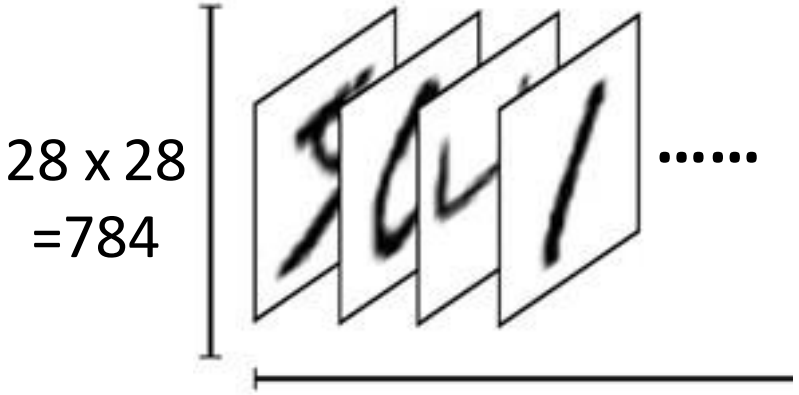
=0

Keras

Step 3.2: Find the optimal network parameters

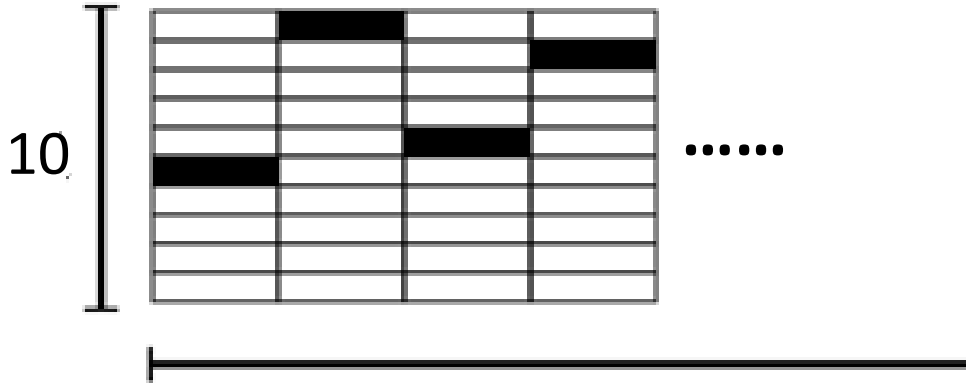
```
model.fit(x_train, y_train, batch_size=100, nb_epoch=20)
```

numpy array



Number of training examples

numpy array



Number of training examples

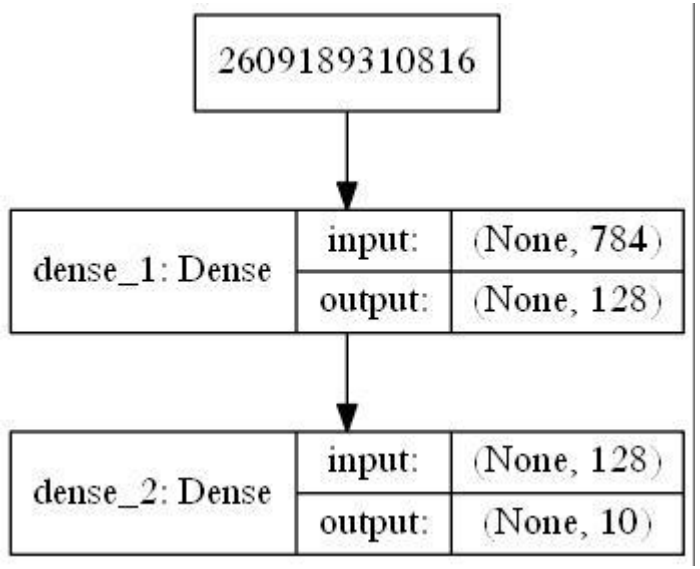
MLP手写数字识别

```
import tensorflow
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import *
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train/255.0
x_test = x_test/255.0
x_train = x_train.reshape((-1,784))
x_test = x_test.reshape((-1,784))
y_train = tensorflow.keras.utils.to_categorical(y_train, 10)
y_test = tensorflow.keras.utils.to_categorical(y_test, 10)

model = Sequential()
model.add(Dense(128,activation='relu',input_shape=(784,)))
model.add(Dense(256,activation='relu'))
model.add(Dense(10,activation='softmax'))

model.compile(loss=tensorflow.keras.losses.categorical_crossentropy,
optimizer=tensorflow.keras.optimizers.SGD(),metrics=['accuracy'])
model.fit(x_train,y_train,batch_size=512, verbose=1, epochs=100)
yp = model.predict(x_test)
print(model.evaluate(x_test,y_test))
```

```
from keras.utils import plot_model
plot_model(model, './model.jpg',
show_shapes=True)
```



Thank you! Questions?