TENSOR FLOW

An Introduction

DISCLAIMER

https://www.tensorflow.org/versions/r1.15/api_docs/python/tf

什么是Tensorflow

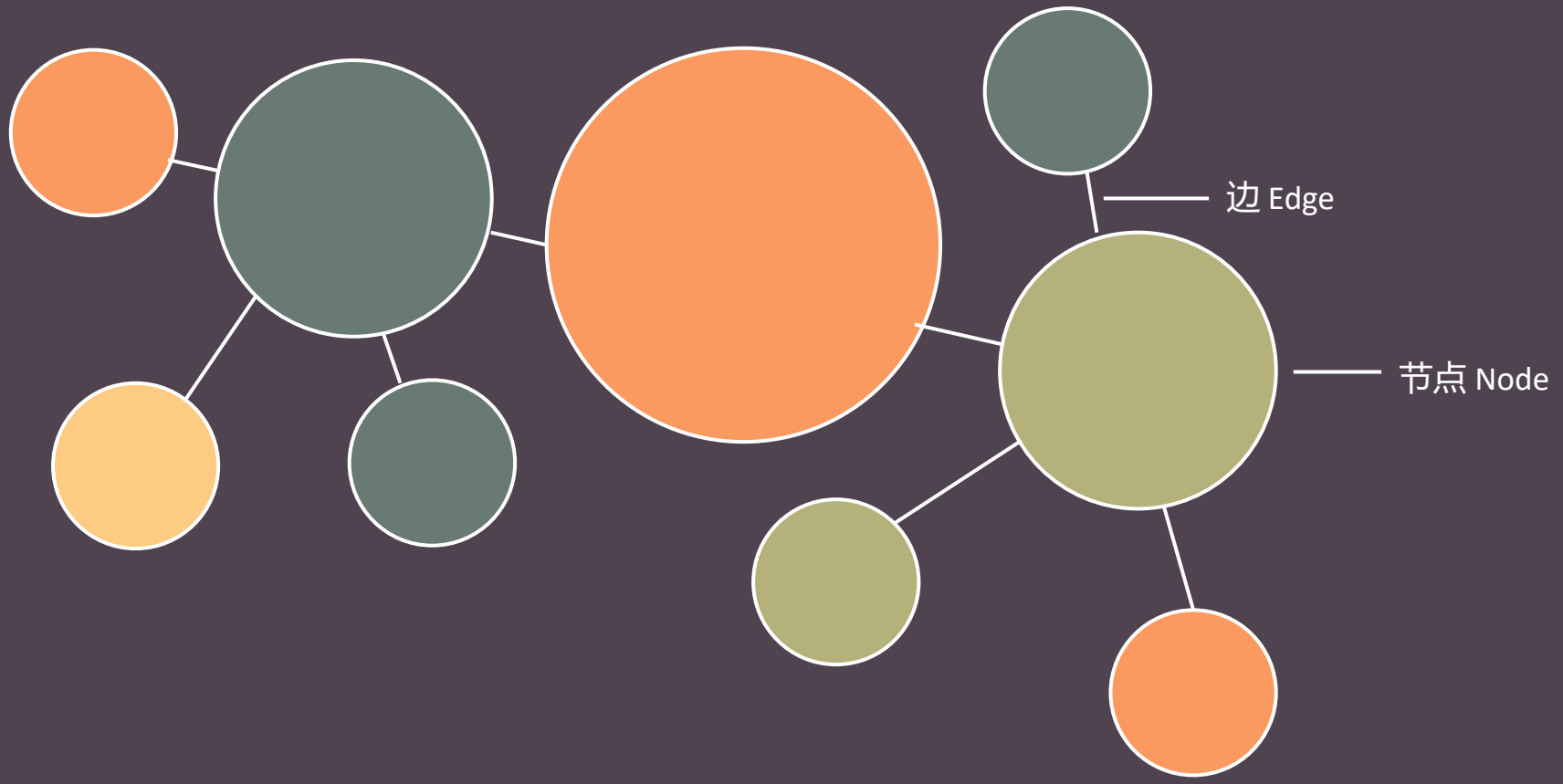
TF原理简介

TENSORFLOW是什么

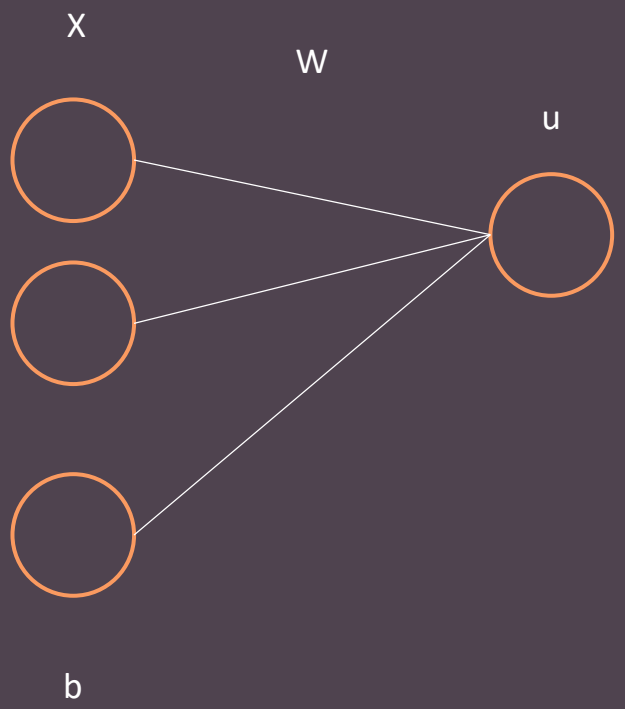
- Python库
- 神经网络建模工具
- 面向对象程序
- 图!
- 图！！
- 图！！！

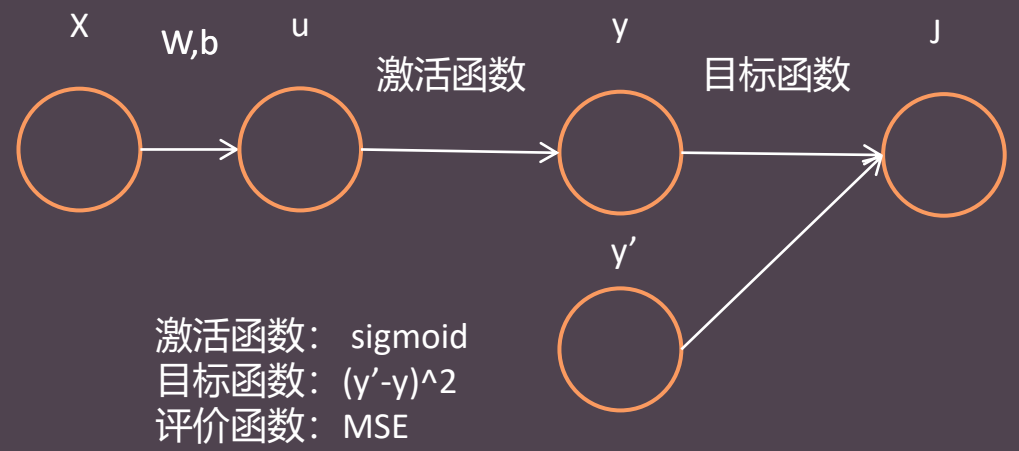
https://www.tensorflow.org/versions/r1.15/api_docs/python/tf

图



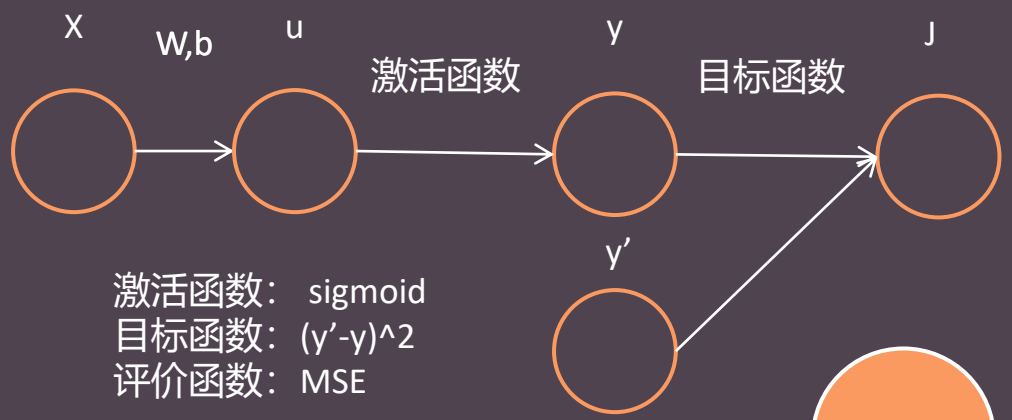
栗子



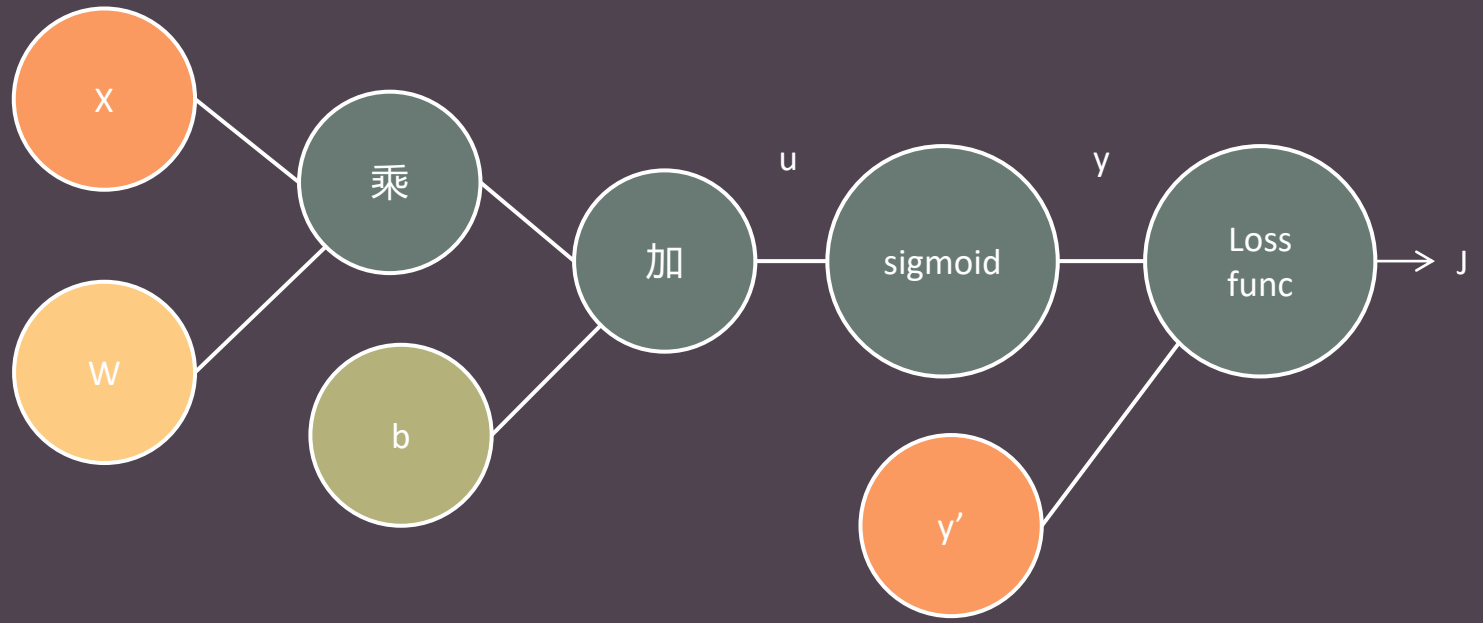


激活函数： sigmoid
目标函数：(y’-y)^2
评价函数：MSE

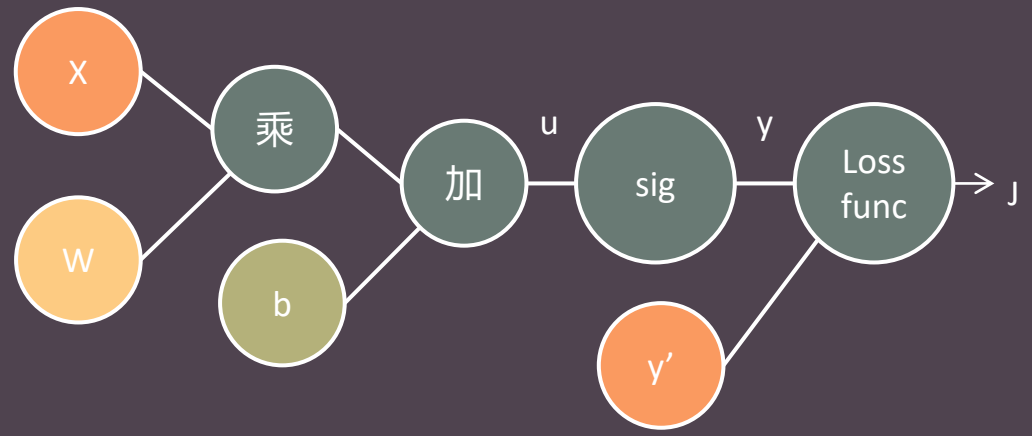
栗子 in TF图



激活函数： sigmoid
目标函数：(y'-y)^2
评价函数：MSE



TF节点的类型



tf.constant()

- 储存常量
- bias

tf.Variable()

- 储存变量
- 随训练而update
- embedding， weight

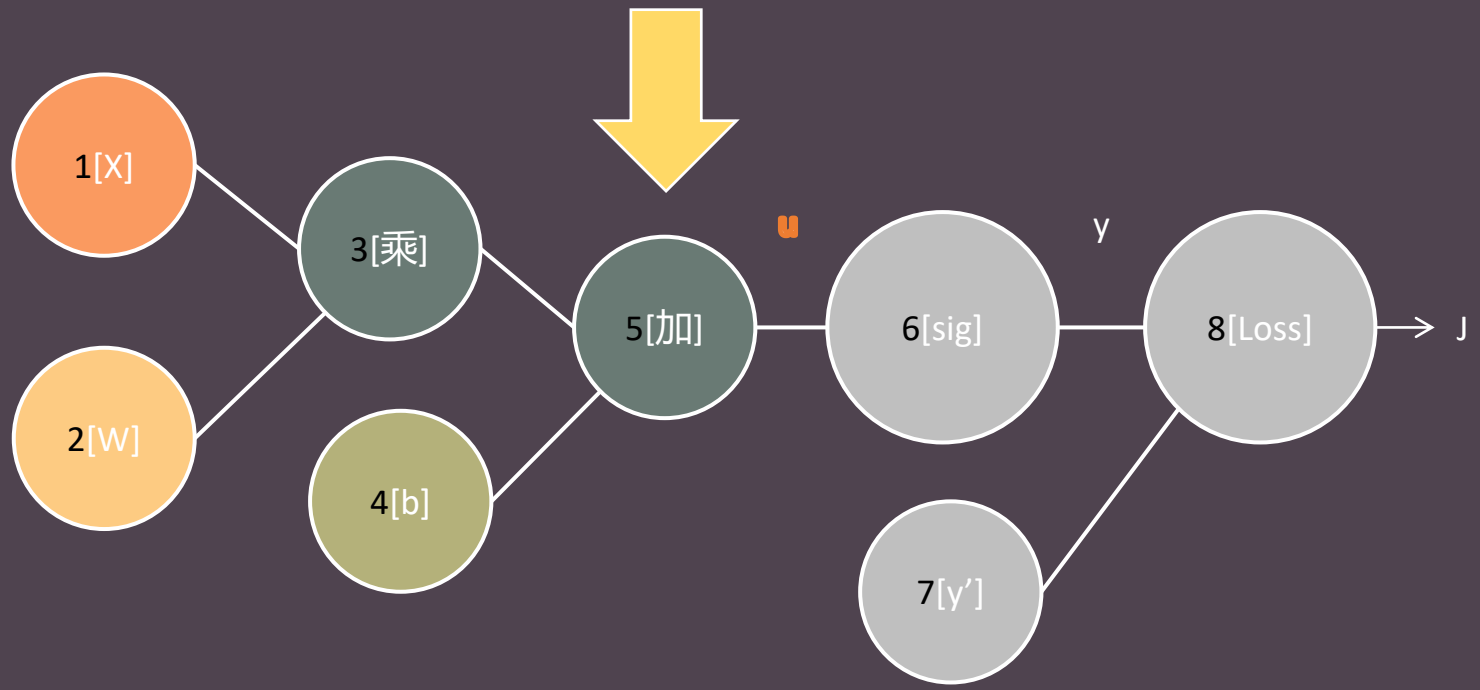
tf.placeholder()

- 储存输入数据
- 每次训练变化
- x, y

tf.Matmul() 和很多

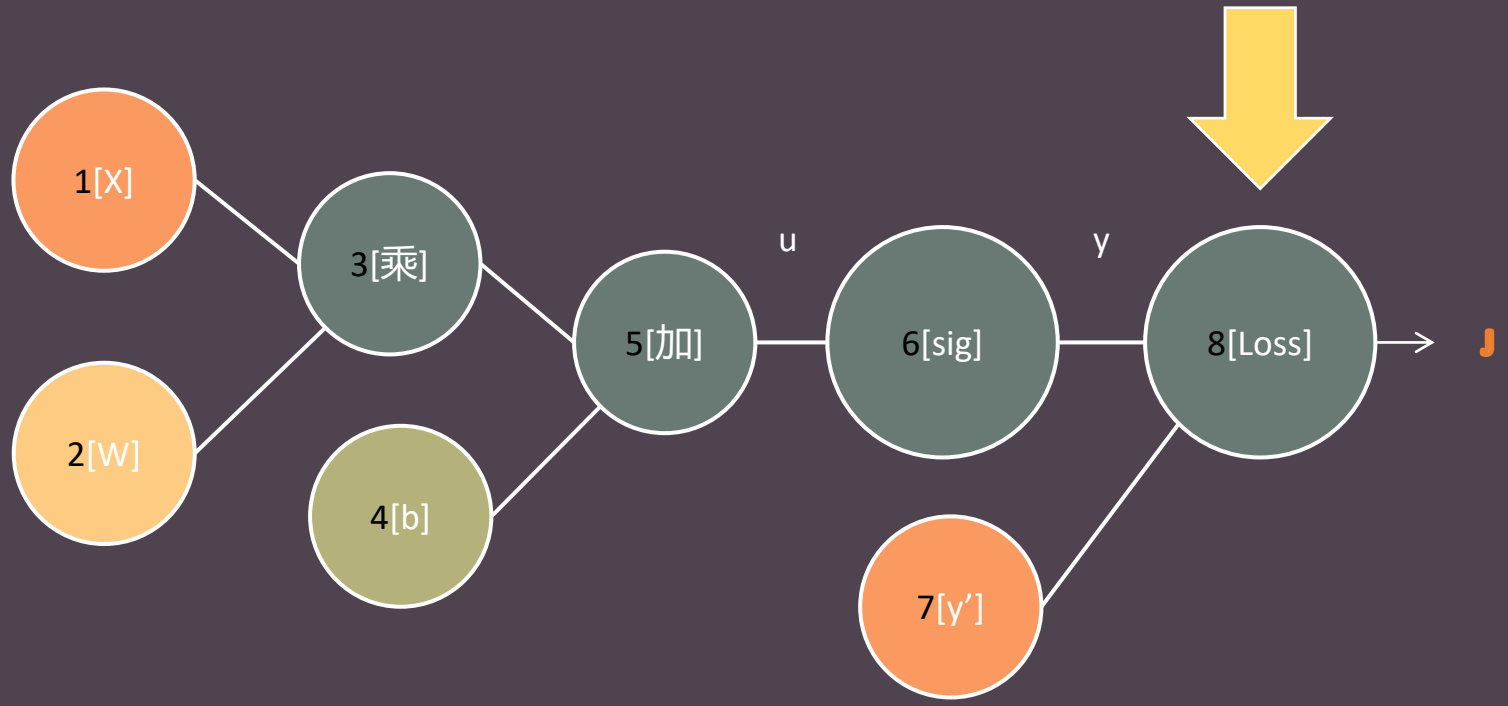
- 其它三项的运算
- 加减乘除
- sigmoid， entropy loss

使用TF图



u = tf.run(5, {1:'X'})

使用TF图



J = tf.run(8, {1: X, 7: y'})

train = tf.train.gradientdescentoptimizer().minimize(J)

tf.run(train)

Tensorflow三步走

定义 – 建模 – 运行

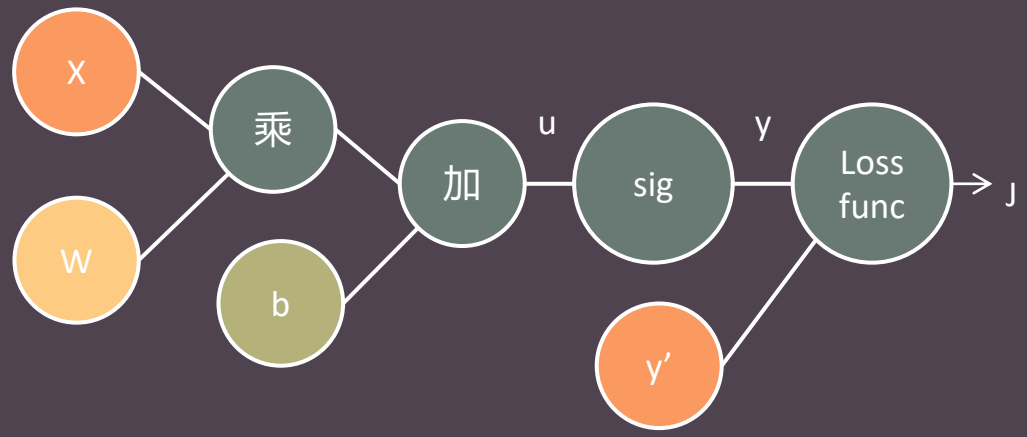
TENSORFLOW三部走

1. 定义： variable， placeholder 以及 constant

2. 建模：使用 variable, placeholder 和 constant 运算 (loss function 和 evaluation score)

3. 运行：放入不同的数据，训练模型，计算performance

1. 定义



1 定义常量 constants

b = tf.constant(1)

b = tf.constant([2, 3] , tf.float32)

2 定义变量 variables (初始weight)

initial_weight = tf.zeros([size_row, size_col])

W = tf.Variable(initial_weight)

initial_weight = tf.truncated_normal ([size_row, size_col], stddev = 1.0 / numpy.sqrt(size_row))

W = tf.Variable(initial_weight)

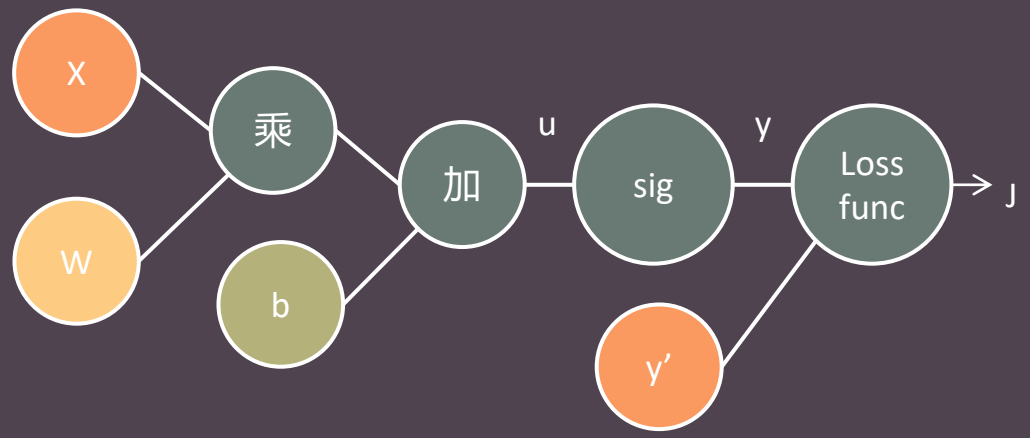
3 定义数据

X = tf.placeholder(tf.float32, shape = 2)

y’ = tf.placeholder(tf.float32, shape = [None, 1])

激活函数： sigmoid
目标函数：(y’-y)^2
评价函数： MSE

2. 建模



激活函数： sigmoid
目标函数：(y'-y)^2
评价函数： MSE

1 定义目标函数

```
u = tf.matmul ( W, X ) + b
y = tf.sigmoid( u )
J = tf.reduce_sum ( tf.square( y' – y ) )
J = tf.reduce_sum ( tf.squared_difference( y', y ) )
J = tf.reduce_sum ( tf.nn.softmax_cross_entropy_with_logits( labels = y', logits = y ) )
```

2 定义optimizer

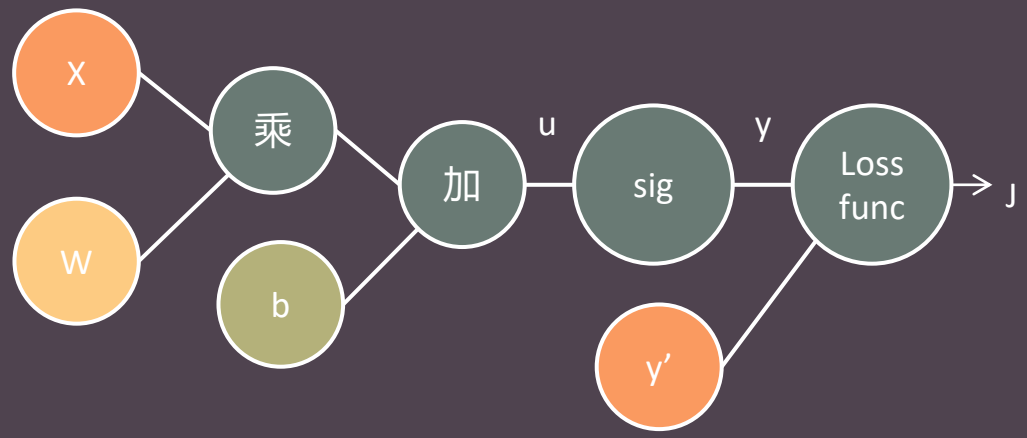
```
optimizer = tf.train.GradientDescentOptimizer ( learning_rate = 0.01 )
optimizer = tf.train.AdagradOptimizer ( learning_rate = 0.01 )
optimizer = tf.train.RMSPropOptimizer ( learning_rate = 0.01 )
optimizer = tf.train.AdamOptimizer ( learning_rate = 0.01 )

train = optimizer.minimize ( J )
```

3 定义评价函数

```
mse = tf.losses.mean_squared_error( y', y )
mse = tf.reduce_mean ( tf.square ( y' - y ) )
matches = tf.equal ( tf.argmax ( y', 1 ), tf.argmax ( y, 1 ) )    # [0, 0, 1, 1]
accuracy = tf.reduce_mean ( tf.cast ( matches, tf.float ) )    # 0.5
```

3. 运行



1 开始一个session 并初始化所有变量

```
with tf.Session() as sess:
    sess.run ( tf.global_variables_initializer() )
```

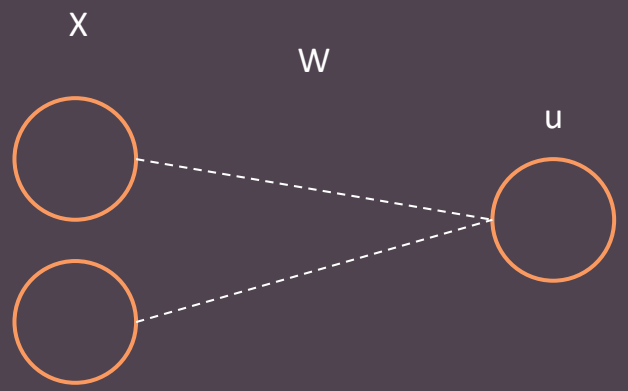
2 训练：run optimize.minimize()

```
…
    for i in iteration_num:
        sess.run ( train, { X: [2,3] , y' : [1] } )
```

3 记录 目标函数/评价函数 数值

```
…
    for …
        …
        loss, error = sess.run ( [ J, mse ], { X: [2,3], y': [1] } )
        …
```

补充：其他TF实用TIPS



1. 防止过度训练 (overfit)

a) 添加dropout

```
X_drop = tf.nn.dropout ( X, rate = 0.1 )
u = tf.Matmul ( W, X_drop )
….
```

b) 添加regularization

```
beta = 0.0025
regularization  = tf.nn.l2_loss ( W ) + …
J = tf.reduce_sum ( tf.square( y’ – y ) ) + beta * regularization
```

补充：其他TF实用TIPS

2 储存 / 读取 模型

a) 储存模型

```
saver = tf.train.Saver()
with tf.Session() as sess:
    for i in sess.run (train):
        …
    saver.save(sess, ‘./model.ckpt’)
```

b) 调用模型

```
with tf.Session() as sess:
    saver.restore ( sess, ‘/.model/ckpt’)
```

补充：其他TF实用TIPS

3 添加embedding

```
init_embedding = tf.random_uniform ( [ vocab_size, embedding_size ] , -1.0, 1.0 )
embedding = tf.Variable ( init_embedding )
X_embed = tf.nn.embedding_lookup ( embedding,  2 )
```

4 改变维度

1. 添加维度：

```
x = tf.Variable ( [ [1,2,2] ] )                 # [1, 3]                  [ [1,2,2] ]
tf.expand_dims ( x , 1 )                        # [1, 3] -> [1,1,3]       [ [ [1,2,2] ] ]
```

2. 对调维度：

```
x = tf.Variable ( [[[2,2,2], [1,2,2]]] )        # [1, 2, 3]               [ [ [2,2,2], [1,2,2] ] ]
tf.transpose( x , [1, 0, 2] )                   # [1, 2, 3] -> [2,1,3]    [ [ [2,2,2] ], [ [1,2,2] ] ]
```

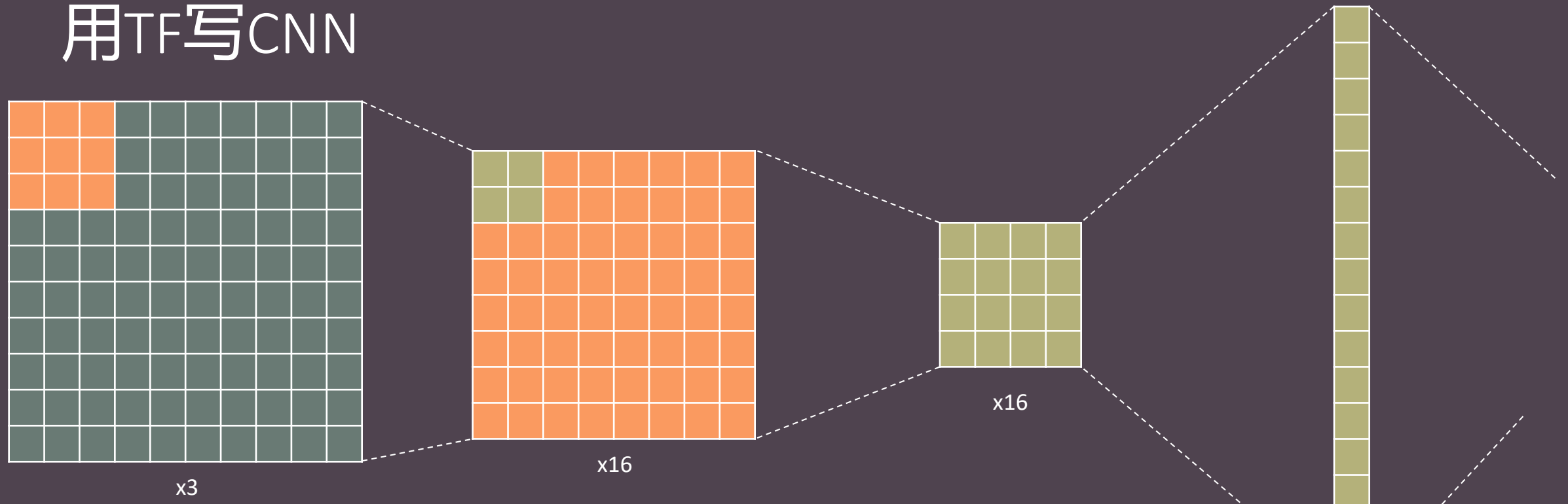
3. 改变维度：

```
x = tf.Variable ( [[[2,2,2], [1,2,2]]] )        # [1, 2, 3]               [ [ [2,2,2], [1,2,2] ] ]
tf.reshape( x , [-1, 2] )                       # [1, 2, 3] -> [3 ,  2]   [ [2,2], [2,1], [2,2] ]
```

用TF写深度学习

CNN and RNN

用TF写CNN



Weight
[width, height, Cin, Cout]
[3x3x3x16]

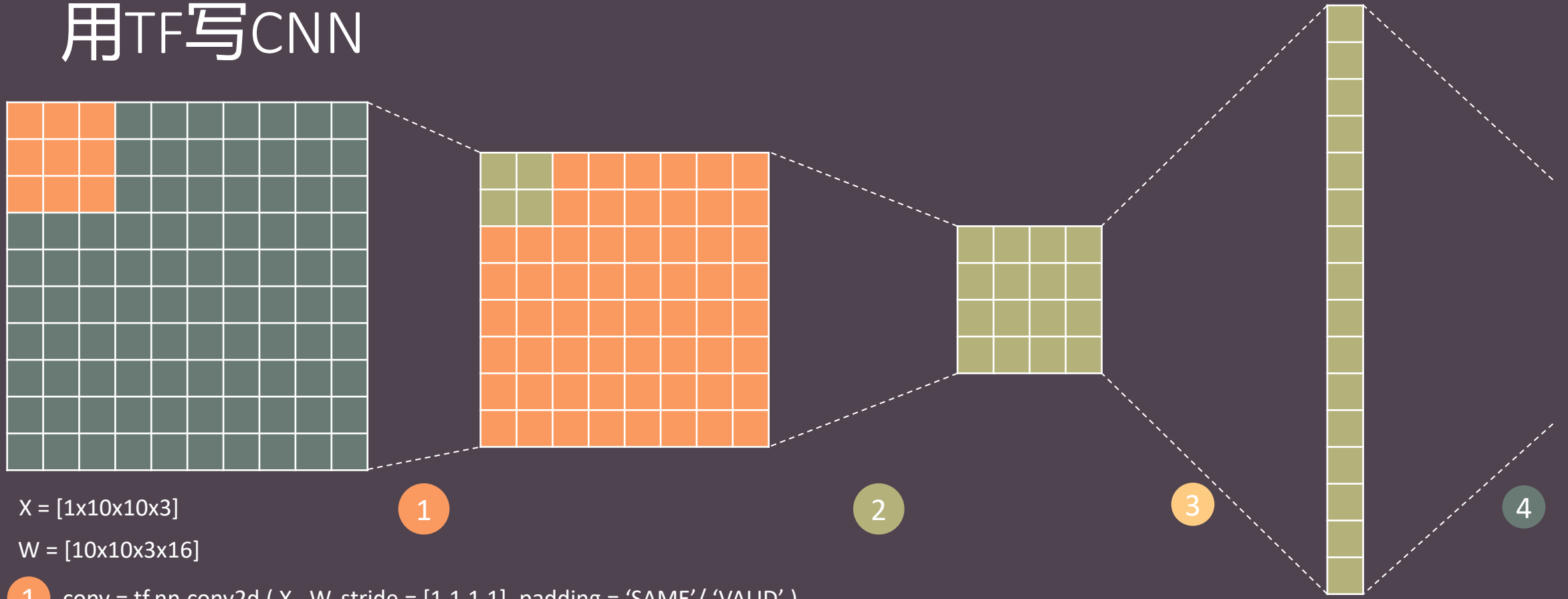
Original Data
[batch_size, width, height, Cin]
[1x10x10x3]

Filtered Feature
[batch_size, width, height, Cout]
[1x8x8x16]
** new width = old_width – filter_size +1*

Pooled Feature
[batch_size,width,height,Cout]
[1x4x4x16]
** new width = old_width/filter_size*

Flattened layer
[1, width*height*Cout]
[1x256]

用TF写CNN



X = [1x10x10x3]

W = [10x10x3x16]

1 conv = tf.nn.conv2d (X, W, stride = [1,1,1,1], padding = ‘SAME’/ ‘VALID’)

2 pool = tf.nn.max_pool (conv, ksize = [1,2,2,1], stride = [1,2,2,1], padding = ‘SAME’ / ‘VALID’)

3 flat = tf.reshape (pool, shape=[-1,4*4*16])

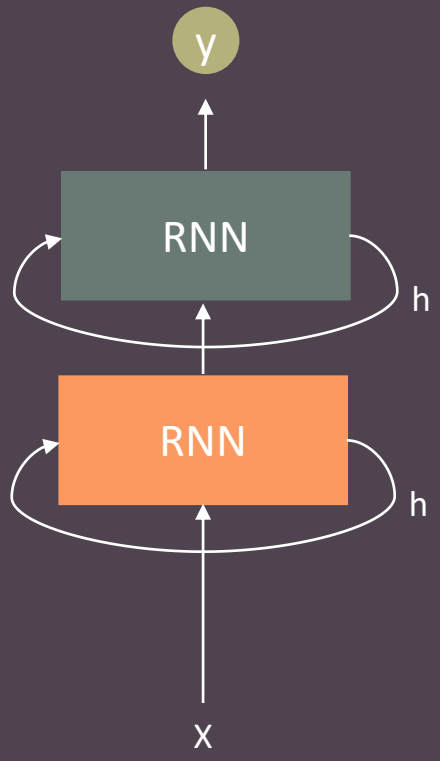
4 logit = tf.sigmoid (tf.matmul (flat, weight) + bias)

***

stride: [1, width, height, 1]

ksize: [1, filter_w, filter_h, 1]

用TF写RNN



1 选择RNN cell的类型：

```
cell = tf.nn.rnn_cell.BasicRNNCell ( output_size )
cell = tf.nn.rnn_cell.BasicLSTMCell ( output_size)
cell = tf.nn.rnn_cell.BasicGRUCell (output_size)
```

2 定义cell纵向的堆叠层数

```
cell_stack = tf.nn.rnn_cell.MultiRNNCell ( [cell]*3 )
```

3 横向扩展RNN

```
init_state = cell.zero_state (batch_size, tf.float32)
outputs, states = tf.nn.dynamic_rnn ( cell_stack, inputs = X, initial_state = init_state)c
```

- outputs: [batch_size, seq_len, dimension]
- states: (c: [batch_size, dimension], h:[batch_size, dimension])

加油 & HAVE FUN!