

神经网络

神经网络介绍

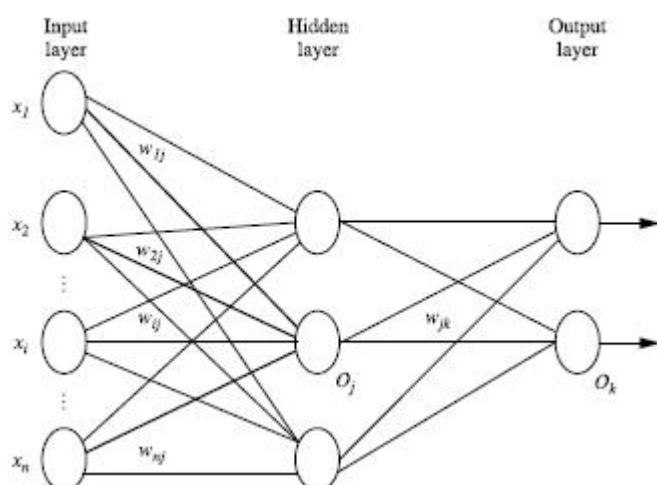
人工神经网络（Artificial Neural Networks）也简称为神经网络（NN）。是模拟人类大脑神经网络的结构和行为。

20 世纪 80 年代以来，人工神经网络（Artificial Neural Network）研究所取得的突破性进展。神经网络辨识是采用神经网络进行逼近或建模，神经网络辨识为解决复杂的非线性、不确定、未知系统的控制问题开辟了新途径。

神经网络主要应用领域有：模式识别与图象处理（语音、指纹、故障检测和图象压缩等）、控制与优化、系统辨识、预测与管理（市场预测、风险分析）、通信等。

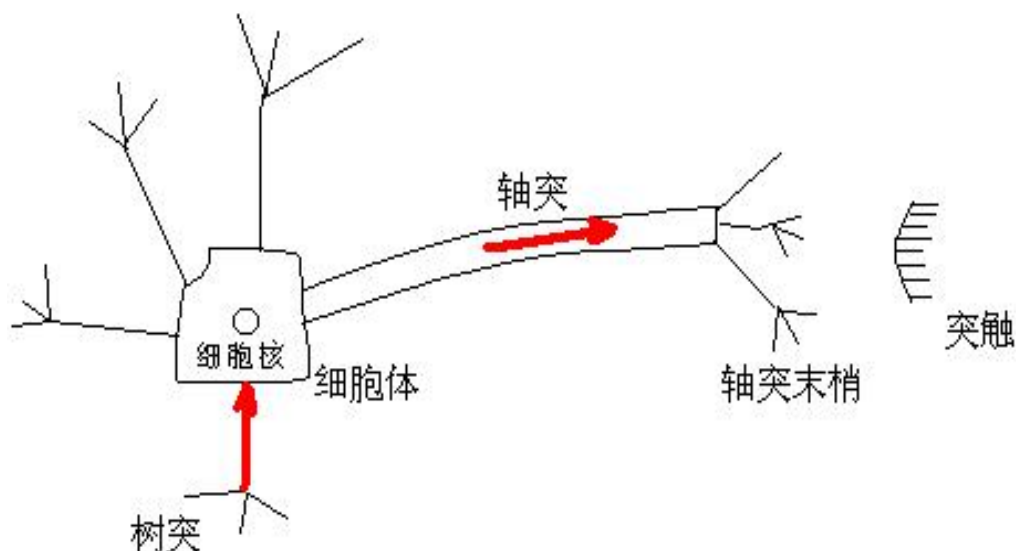
神经网络原理

经典的神经网络有以下三个层次组成：输入层(input layer), 隐藏层 (hidden layers), 输出层 (output layers)。



每个圆圈就是一个神经元。每层与每层之间是没有连接的，但是层与层之间都有连接。每个连接都是带有权重值的。隐藏层和输出层的神经元由输入的数据计算输出，但输入层神经元只有输入，一般指一个训练数据样本的数据。

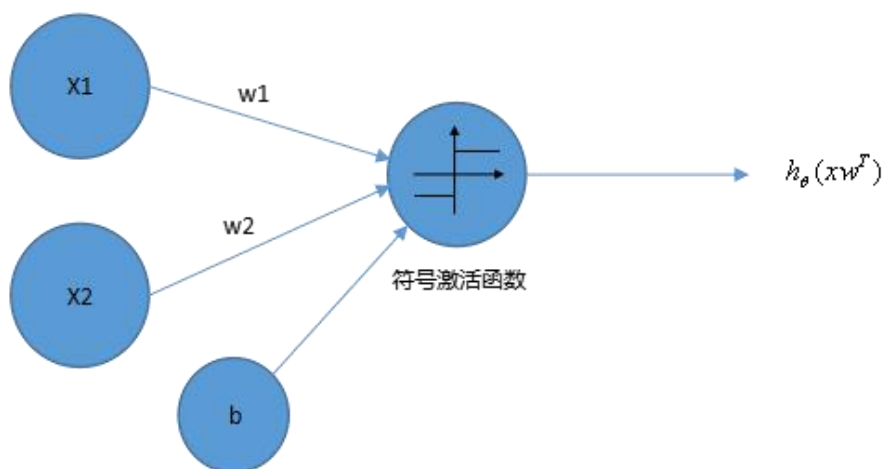
神经系统的基本构造是神经元(神经细胞)，它是处理人体内各部分之间相互信息传递的基本单元。每个神经元都由一个细胞体，一个连接其他神经元的轴突和一些向外伸出的其它较短分支—树突组成。轴突功能是将本神经元的输出信号(兴奋)传递给别的神经元，其末端的许多神经末梢使得兴奋可以同时传送给多个神经元。树突的功能是接受来自其它神经元的兴奋。神经元细胞体将接收到的所有信号进行简单地处理后，由轴突输出。神经元的轴突与另外神经元神经末梢相连的部分称为突触。



神经元的解剖图

感知机

感知机是一类人造神经元，模拟这样的大脑神经网络处理数据的过程。感知机模型如下图所示：

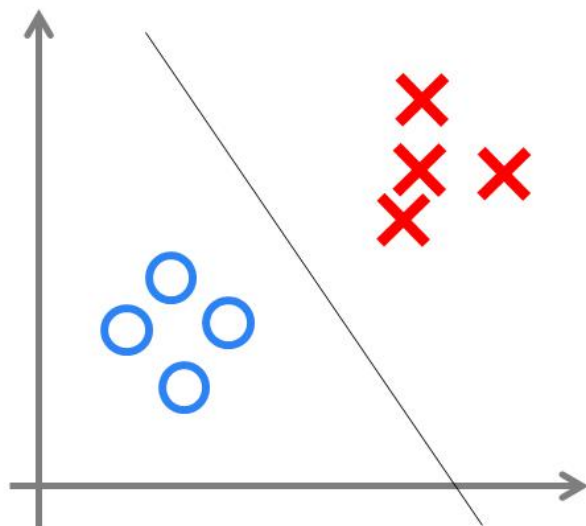


其中 x_1 , x_2 为输入， b 为偏置，激活函数被称为符号函数（sign function），感知机是一种基本的分类模型，类似于逻辑回归。不同的是感知机的逻辑函数用的是 sign，而逻辑回归用的是 Sigmoid 函数，感知机也具有连接权重和偏置。

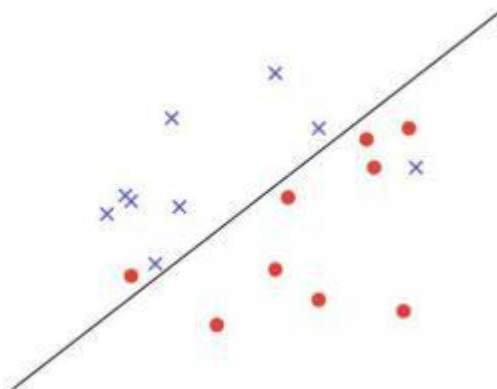
$$f(x) = \text{sign}(w \cdot x + b)$$

$$\text{sign}(x) = \begin{cases} +1, & x \geq 0 \\ -1, & x < 0 \end{cases}$$

感知机可以用来处理线性可分类问题，线性不可分简单来说，就是不可用一条直线把图上两类点划分开。如第二张图所示，无论怎么画直线都无法将两类点区分开。



线性不可分



对于线性不可分问题一般用多层神经网络，打开 <http://playground.tensorflow.org/>。使用 playground 体会感知机的分类。

激活函数

Sigmoid 函数

sigmoid 函数 由于其单增及反函数单增等性，sigmoid 函数常被用做神经网络的激活函数，将变量映射到 0, 1 之间。所以主要用来做二分类神经网络。

由于其平滑、易于求导的特性，处理特征相差不是很大或者复杂的数据效果比较好。

sigmoid 函数的公式：

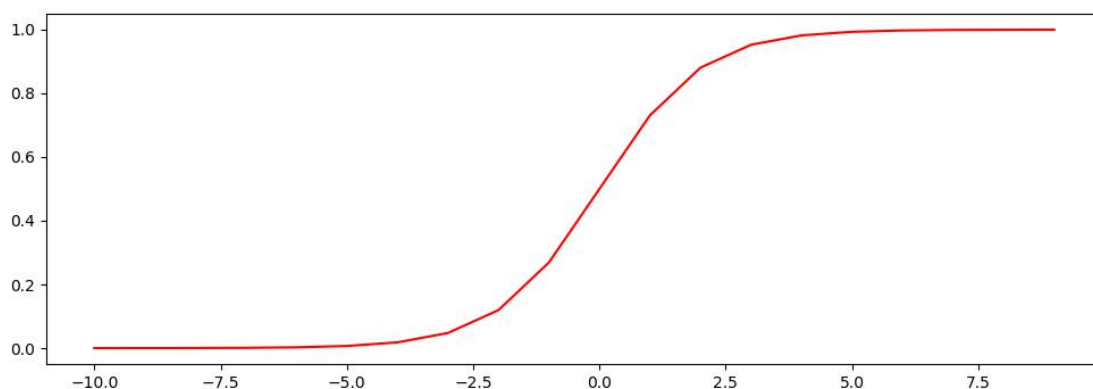
$$S(x) = \frac{1}{(1 + e^{-x})}$$

【示例】使用 matplotlib 绘制 Sigmoid 函数

```
def sigmoid(x):
    return 1.0/(1+np.exp(-x))

nums = np.arange(-10, 10, step=1) #生成一个 numpy 数组
fig, ax = plt.subplots(figsize=(12,4)) #绘制子图
ax.plot(nums, sigmoid(nums), 'r')    #绘制 sigmoid 的函数图像
plt.show()
```

执行结果如图所示：



双曲正切函数（tanh）

双曲正切函数（tanh）是双曲正弦函数（sinh）与双曲余弦函数（cosh）的比值，语法格式如下：

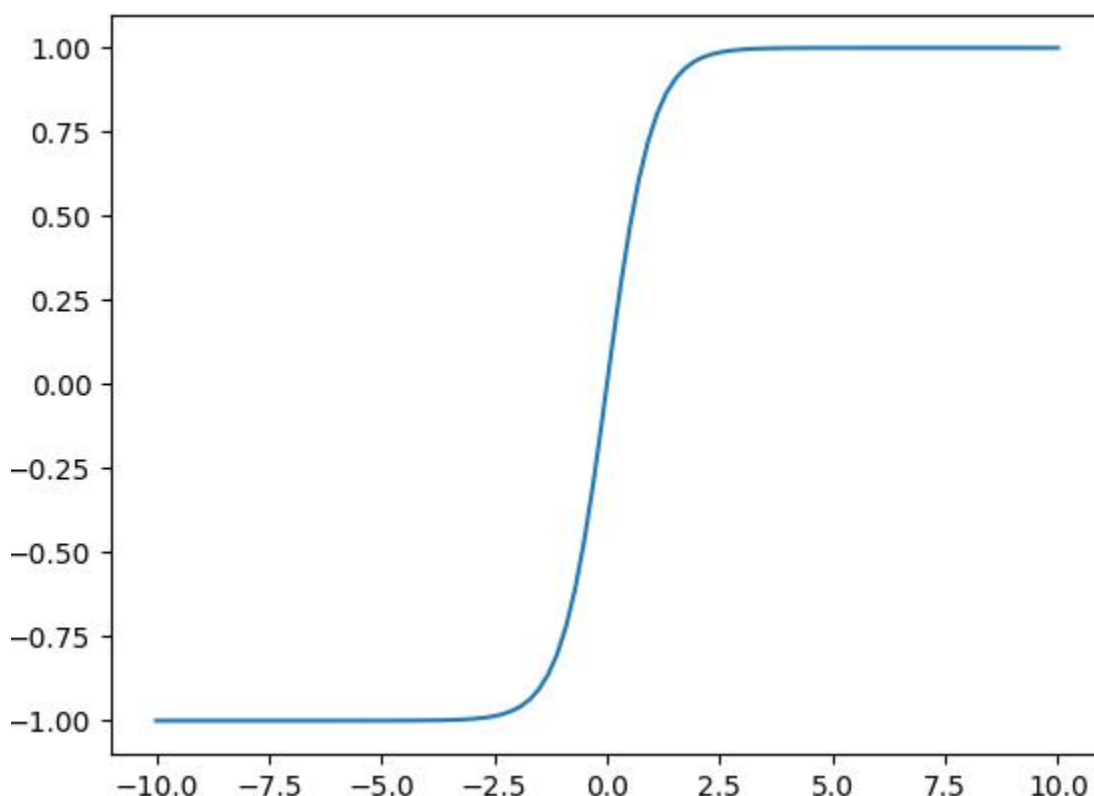
$$\tanh x = \frac{\sinh x}{\cosh x} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

双曲正切函数（tanh）与 tf.sigmoid 非常接近，且与后者具有类似的优缺点。tf.sigmoid 和 tf.tanh 的主要区别在于后者的值域为[-1.0, 1.0]。

【示例】使用 matplotlib 绘制 tanh 函数

```
import matplotlib.pyplot as plt
import numpy as np
#生成 x 数据
x = np.linspace(-10,10,100)
y = np.tanh(x)
plt.plot(x,y)
plt.show()
```

执行结果如图所示：



TensorFlow 框架介绍

TensorFlow 简介

TensorFlow 是一个开源的、基于 Python 的机器学习框架，它由 Google 开发，并在图形分类、音频处理、推荐系统和自然语言处理等场景下有着丰富的应用，是目前最热门的机器学习框架。除了 Python，TensorFlow 也提供了 C/C++、Java、Go、R 等其它编程语言的接口。

- (1) TensorFlow 是谷歌基于 DistBelief 进行研发的第二代人工智能学习系统。
- (2) 2015 年 11 月在 GitHub 上开源。
- (3) 2016 年 4 月分布式版本。
- (4) 2017 年发布了 1.0 版本，趋于稳定。
- (5) Google 希望让这个优秀的工具得到更多的应用，从整体上提高深度学习的效率。

TensorFlow 相关链接：

- (1) TensorFlow 官网：www.tensorflow.org
- (2) GitHub 网址：github.com/tensorflow/tensorflow
- (3) 模型仓库网址：github.com/tensorflow/models

TensorFlow 的安装：

TensorFlow 提供 CPU 版本和 GPU 加速的版本。CPU 是核芯的数量更少，但是每一个核芯的速度更快，性能更强，更适合于处理连续性任务。GPU 是核芯数量更多，但是每一个核芯的处理速度较慢，更适合于并行任务。

```
pip install tensorflow==1.12.0
```

【示例】使用 TensorFlow 实现加法运算

```
import tensorflow as tf
def tensorflowDemo():
    #回顾原生的 python 相加
    a=10
    b=20
    c=a+b
    print('普通加法运算: ',c)
    print('使用 TensorFlow 进行加法运算')
    a_tf=tf.constant(10)
    b_tf=tf.constant(20)
    c_tf=a_tf+b_tf
    print('TensorFlow 加法运算的结果: ',c_tf)
    #开启会话
    with tf.Session() as sess:
        c_tf_value= sess.run(c_tf)
        print('c_tf_value 的值: ',c_tf_value)

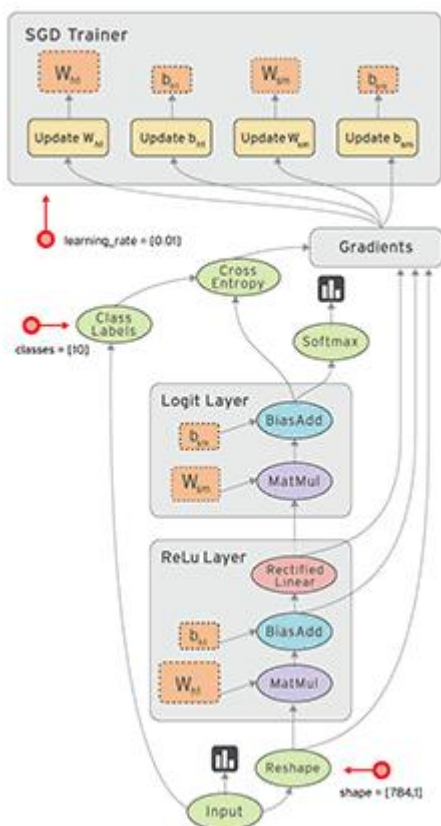
if __name__ == '__main__':
    tensorflowDemo()
```

从上面的示例可以看到使用 TensorFlow 实现加法运算时候，直接输出 `c_tf` 的结果，并没有获得加法运算的结果。如果要获取运算的结果要开启会话。TensorFlow 程序通常被组织成一个构建图阶段和一个执行图阶段。构建阶段指数据与操作的执行步骤被描述成一个图。在执行阶段，使用会话执行构建好的图中的操作。

TensorFlow 的结构有：

- (1) 图：这是 TensorFlow 将计算表示为指令之间的依赖关系的一种表示法。
- (2) 会话：TensorFlow 跨一个或多个本地或远程设备运行数据流图的机制。
- (3) 张量：TensorFlow 中的基本数据对象。
- (4) 节点：提供图当中执行的操作。

TensorFlow 的数据流图



TensorFlow 是一个采用数据流图（data flow graphs），用于数值计算的开源框架。

节点（Operation）在图中表示数学操作，线（edges）则表示在节点间相互联系的多维数据数组，即张量（tensor）

图与 TensorFlow

图包含了一组 `tf.Operation` 代表的计算单元对象和 `tf.Tensor` 代表的计算单元之间流动的数据。

（1）默认图

通常 TensorFlow 会默认创建一张图，通过调用 `tf.get_default_graph()` 访问，要将操作添加到默认图中，直接创建 OP 即可。op、sess 都含有 `graph` 属性。

【示例】查看默认图的图属性

```
#查看默认图
#1.调用方法查看
default_g=tf.get_default_graph()
print('默认图: ',default_g)
```


#2.查看属性

```
print('a_t 的图属性: ',a_t.graph)
```

```
print('b_t 的图属性: ',b_t.graph)
```

```
print('c_t 的图属性: ',c_t.graph)
```

(2) 自定义图

可以通过 `tf.Graph()` 自定义创建图。如果要在这张图中创建 OP, 使用 `tf.Graph.as_default()` 上下文管理器。

【示例】自定义图

```
import tensorflow as tf
import os

os.environ['TF_CPP_MIN_LOG_LEVEL']='2' #不想让警告的信息输出可以添加

def graphDemo():
    #创建图
    new_g = tf.Graph()
    #在自己的图中定义数据和操作
    with new_g.as_default():
        a_new = tf.constant(20)
        b_new = tf.constant(30)
        c_new = a_new + b_new
        print('c_new:',c_new)

    #开启会话 需要传入图参数
    with tf.Session(graph=new_g) as sess:
        c_new_value= sess.run(c_new)
        print('c_new_value 的值: ',c_new_value)

if __name__ == '__main__':
    graphDemo()
```

TensorBoard 可视化

TensorFlow 可用于训练大规模深度神经网络所需的计算,使用该工具涉及的计算往往复杂而深奥。为了方便 TensorFlow 程序的理解、调试和优化,TensorFlow 提供了 TensorBoard 可视化工具。

(1) 数据序列化成 events 文件

【示例】将数据序列化 events 文件

```
import tensorflow as tf

def events_demo():
    a = tf.constant(20)
    b = tf.constant(30)
    c = a+b
    print('c:',c)

    #开启会话
    with tf.Session() as sess:
        c_value = sess.run(c)
        print('c_value:',c_value)
        #sess 的图属性
        print('sess 的图图属性: ',sess.graph)
        #将图写入本地生成的 events 文件
        writer = tf.summary.FileWriter('e:/events/test',graph=tf.get_default_graph())
        writer.close()
```

(2) 启动 TensorBoard

```
tensorboard --logdir='e:/events/test'
```

在浏览器中打开 TensorBoard 的图页面,输入 <http://localhost:6006>,会看到如下图形类似的图。



会话

一个会话里包括了 TensorFlow 运行时的控制和状态。通过调用 `tf.Session()` 就可以生成一个会话 (Session) 对象。会自动调用 Session 类的初始化 `__init__()` 方法, 语法格式如下:

```
def __init__(target='', graph=None, config=None)
```

其中参数 `target`: 如果设置为空 (默认设置), 会话将仅使用本地计算机中的设备, 可以指定 `grpc://网址`, 以便指定 TensorFlow 服务器的地址, 这使得会话可以访问该服务器控制的计算机上的所有设备。参数 `graph`: 默认情况下, 新的 `tf.Session` 将绑定到当前的默认图。

参数 `config`: 此参数允许您指定一个 `tf.ConfigProto` 以便控制会话的行为。例如, `ConfigProto` 协议用于打印设备使用信息。

【示例】`config` 参数的使用

```
import tensorflow as tf

def sess_demo():
    a = tf.constant(10)
    b = tf.constant(20)
    c = a+b
    print('tensorflow 的加法运算: ',c)
    #开启会话
    with tf.Session(config=tf.ConfigProto(
                                allow_soft_placement=True,
                                log_device_placement=True)) as sess:
        c_value = sess.run(c)
        print('c_value:',c_value)
if __name__ == '__main__':
    sess_demo()
```

执行如下图:



```
add: (Add): /job:localhost/replica:0/task:0/device:CPU:0
Const: (Const): /job:localhost/replica:0/task:0/device:CPU:0
Const_1: (Const): /job:localhost/replica:0/task:0/device:CPU:0
c_value: 30
```

会话的 `run()`

```
def run(self, fetches, feed_dict=None, options=None, run_metadata=None):
```

其中参数 `fetches`: 单一的操作或者列表、元组 (其它不属于 `TensorFlow` 类型的不行)。

【示例】会话中获取结果

```
import tensorflow as tf

def sess_demo():
    a = tf.constant(10)
    b = tf.constant(20)
    c = a+b
    print('tensorflow 的加法运算: ',c)
    #开启会话
```

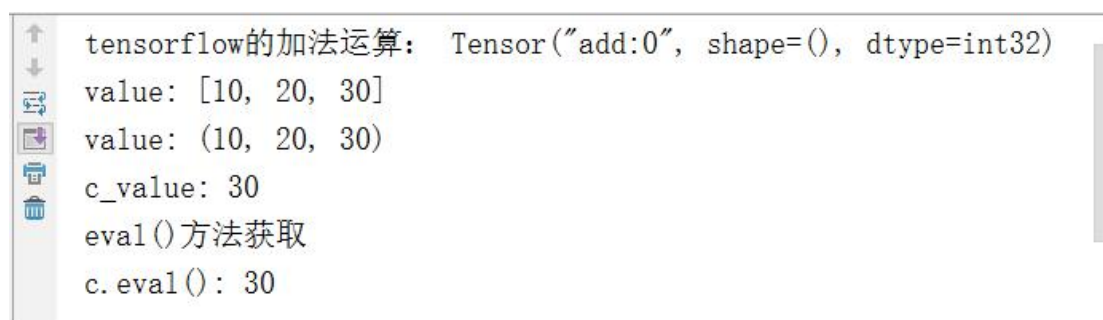
```

with tf.Session() as sess:
    value = sess.run([a,b,c])
    print('value:',value)
    value2 = sess.run((a, b, c))
    print('value:', value2)
    c_value = sess.run(c)
    print('c_value:',c_value)
    print('eval()方法获取')
    print('c.eval():',c.eval())

if __name__ == '__main__':
    sess_demo()

```

执行结果如下图:



```

tensorflow的加法运算: Tensor("add:0", shape=(), dtype=int32)
value: [10, 20, 30]
value: (10, 20, 30)
c_value: 30
eval()方法获取
c.eval(): 30

```

参数 `feed_dict`: 允许调用者覆盖图中张量的值, 运行时赋值。使用占位符的方式, 占位符是一个可以在之后赋给它数据的变量。它是用来接收外部输入的。占位符可以是一维或者多维, 用来存储 n 维数组。`feed_dict` 必须与 `tf.placeholder` 搭配使用, 则会检测值的形状是否与占位符兼容。

【示例】`feed_dict` 的使用

```

import tensorflow as tf
a = tf.placeholder(tf.float32)
b = tf.placeholder('float32')
c = a+b
cc = tf.add(a,b)
x = tf.placeholder(tf.float32,None)
y = x*20 +100
with tf.Session() as sess:
    c_value = sess.run(c,feed_dict={a:10,b:20})
    cc_value = sess.run(cc,feed_dict={a:10,b:20})
    print('c_value:',c_value)

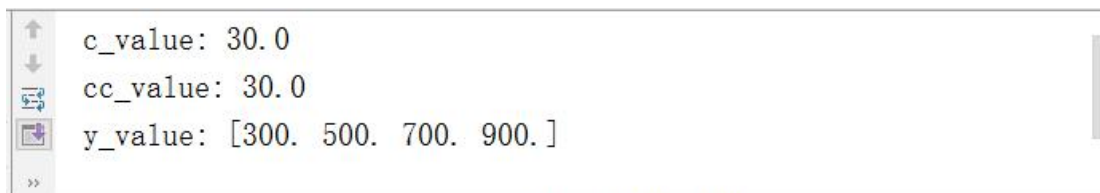
```

```
print('cc_value:',cc_value)

y_value=sess.run(y,feed_dict={x:[10,20,30,40]})

print('y_value:',y_value)
```

执行结果如下图:



```
c_value: 30.0
cc_value: 30.0
y_value: [300. 500. 700. 900.]
```

张量

张量是一个数学对象，它是对标量、向量和矩阵的泛化。张量可以表示为一个多维数组。零秩张量就是标量。向量或者数组是秩为 1 的张量，而矩阵是秩为 2 的张量。简而言之，张量可以被认为是一个 n 维数组。

TensorFlow 用张量这种数据结构来表示所有的数据。可以把一个张量想象成一个 n 维的数组或列表。张量可以在图中的节点之间流通。其实张量代表的就是一种多维数组。在 TensorFlow 系统中，张量的维数来被描述为阶。

阶	数学实例	Python	例子
0	标量	(只有大小)	$s = 483$
1	向量	(大小和方向)	$v = [1.1, 2.2, 3.3]$
2	矩阵	(数据表)	$m = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]$
3	3 阶张量	(数据立体)	$t = [[[2], [4], [6]], [[8], [10], [12]], [[14], [16], [18]]]$
n	n 阶		

张量的类型

数据类型	Python 类型	描述
DT_FLOAT	tf.float32	32 位浮点数.
DT_DOUBLE	tf.float64	64 位浮点数.
DT_INT64	tf.int64	64 位有符号整型.
DT_INT32	tf.int32	32 位有符号整型.

数据类型	Python 类型	描述
DT_INT16	tf.int16	16 位有符号整型.
DT_INT8	tf.int8	8 位有符号整型.
DT_UINT8	tf.uint8	8 位无符号整型.
DT_STRING	tf.string	可变长度的字节数组. 每一个张量元素都是一个字节数组.
DT_BOOL	tf.bool	布尔型.
DT_COMPLEX 64	tf.complex64	由两个 32 位浮点数组成的复数: 实数和虚数.
DT_QINT32	tf.qint32	用于量化 0ps 的 32 位有符号整型.
DT_QINT8	tf.qint8	用于量化 0ps 的 8 位有符号整型.
DT_QUINT8	tf.quint8	用于量化 0ps 的 8 位无符号整型.

创建张量

【示例】固定张量的创建

```
import tensorflow as tf
tensor1 = tf.constant(5.0)
tensor2 = tf.constant([1,2,3,4,5,6])
tensor3 = tf.constant([[1,2,3],[4,5,6]])
print('tensor1:',tensor1)
print('tensor2:',tensor2)
print('tensor3:',tensor3)
```

【示例】随机张量的创建

```
import tensorflow as tf
print('随机张量的创建')
r = tf.ones(shape=[1,2,3])
print(r)
z = tf.zeros(shape=[3,4])
print(z)
print('创建均匀分布随机值')
r1 = tf.random_uniform([2,3],minval=0,maxval=4)
sess = tf.Session()
```

```
print(sess.run(r1))  
print('创建均值和标准差的正太分布随机值')  
r2 = tf.random_normal([2,3],mean=5,stddev=4)  
print(sess.run(r2))  
sess.close()
```

张量的形状变换

TensorFlow 的张量具有两种形状变换，动态形状和静态形状。

静态形状转换调用 `set_shape()`，转换的时候必须和初始创建张量时形状相同，对于已经固定好静态形状的张量，不能再次设置静态形状。

【示例】更改静态形状

```
a_p = tf.placeholder(dtype=tf.float32,shape=[None,None])  
b_p = tf.placeholder(dtype=tf.float32,shape=[None,5])  
print('静态形状修改')  
print('修改前形状: ')  
print('a_p:',a_p)  
print('b_p:',b_p)  
print('修改后形状')  
a_p.set_shape([2,5])  
print('a_p:',a_p)  
b_p.set_shape([5,5])  
print('b_p:',b_p)
```

动态形状转换调用 `tf.reshape(tensor,shape)`，动态转换形状张量的元素个数必须和修改前要相同。

【示例】更改动态形状

```
c_p = tf.placeholder(dtype=tf.float32,shape=[3,4])  
print('动态修改前: ')  
print('c_p:',c_p)  
print('动态修改后')  
c_p = tf.reshape(c_p,shape=[2,6])  
print('c_p:',c_p)  
c_p = tf.reshape(c_p,shape=[6,2])  
print('c_p:', c_p)
```

```
c_p = tf.reshape(c_p,shape=[4,3])
print('c_p:', c_p)
```

【示例】矩阵运算

```
import tensorflow as tf
import numpy as np
sess = tf.Session()
A = tf.random_uniform([3,2])
B = tf.fill([2,4],3.5)
C = tf.random_normal([3,4])
print('A\n',sess.run(A))
print('B\n',sess.run(B))
print('sess.run(tf.matmul(A,B))\n',sess.run(tf.matmul(A,B)))
print('C\n',sess.run(C))
print('加法运算: ')
print(sess.run(tf.matmul(A,B)+C))
```

变量

为了训练模型，需要能够修改图以调节一些对象，比如权重值、偏置量。简单来说，变量让你能够给图添加可训练的参数，它们在创建时就带有类型属性和初始值。

变量的创建语法格式如下：

```
tf.Variable(initial_value=None,
            trainable=True,
            collections=None,
            validate_shape=True,
            caching_device=None,
            name=None,)
```

其中参数 `initial_value` 是初始化的值，`trainable` 是否被训练。

【示例】变量的使用

```
import tensorflow as tf
x = tf.constant([10,20,30,40])
y = tf.Variable(x*2+10)
#初始化变量
model = tf.global_variables_initializer()
with tf.Session() as sess:
```



```
#运行初始化  
sess.run(model)  
print('y:',sess.run(y))
```

实现线性回归

根据数据建立回归模型， $y=w_1x_1+w_2x_2+...+b$ ，通过真实值与预测值之间建立误差，使用梯度下降优化得到损失最小对应的权重和偏置。最终确定模型的权重和偏置参数，最后可以用这些参数进行预测。

线性回归案例实现

假定随机指定 100 个点，只有一个特征。x 和 y 之间的关系满足 $y=kx+b$ 。

实现步骤：

(1) 准备数据

数据分布为 $y=0.8*x+0.7$ 。这里将数据分布的规律确定，是为了使用我们训练处的参数和真实的参数（即 0.8 和 0.7）比较是否训练准确

```
X = tf.random_normal(shape=[100,1])  
y_true = tf.matmul(X,[[0.8]])+0.7
```

(2) 构建模型

模型要满足 $y=weight*x+bias$ ，数据 x 的形状为(100,1)，与权重相乘后的形状为(100,1)，即模型参数权重 weight 的形状为 (1,1)。偏置 bias 的形状可以和权重形状相同也可以是一个标量。

```
weight = tf.Variable(initial_value=tf.random_normal(shape=[1,1]))  
bias = tf.Variable(initial_value=tf.random_normal(shape=[1,1]))  
y_predict = tf.matmul(X,weight)+bias
```

(3) 构建损失函数

线性回归损失函数使用均方误差。

```
error = tf.reduce_mean(tf.square(y_predict-y_true))
```

(4) 优化损失

使用梯度下降优化，语法格式如下：

```
tf.train.GradientDescentOptimizer(learning_rate=0.01).minimize(error)
```

其中 learning_rate 是学习率，一般为 0-1 之间比较小的值。因为要让损失最小，所以调用梯度下降优化器的 minimize() 方法。

【示例】线性回归案例实现

```
import tensorflow as tf

def linear_regression():
    #准备数据
    X = tf.random_normal(shape=[100,1])
    y_true = tf.matmul(X,[[0.8]])+0.7
    #构造模型
    #构造模型参数权重 weight 和偏移 bias
    weight = tf.Variable(initial_value=tf.random_normal(shape=[1,1]))
    bias = tf.Variable(initial_value=tf.random_normal(shape=[1,1]))
    y_predict = tf.matmul(X,weight)+bias
    #构造损失函数
    error = tf.reduce_mean(tf.square(y_predict-y_true))
    #优化损失
    optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.01).minimize(error)
    #显示初始化变量
    init = tf.global_variables_initializer()
    #开始会话
    with tf.Session() as sess:
        #运行初始化变量
        sess.run(init)
        print('查看训练前模型参数:权重: %f, 偏量: %f, 损失: %f'%(weight.eval(),bias.eval(),error.eval()))

        #开始训练
        for i in range(1000):
            sess.run(optimizer)
            print('训练第%d次后模型参数:权重: %f, 偏量: %f, 损失: %f %' %
                  ((i+1),weight.eval(), bias.eval(), error.eval()))

if __name__ == '__main__':
    linear_regression()
```

添加变量显示

在 TensorBoard 中观察损失模型的参数，损失值等变量的变化。

(1) 创建事件文件

```
file_writer = tf.summary.FileWriter('e:/events/test',graph=sess.graph)
```

(2) 收集变量

收集对于损失函数和准确率等单值变量使用 `tf.summary.scalar(name='',tensor)`，收集高维度变量参数使用 `tf.summary.histogram(name='',tensor)`，收集输入的图片张量能显示图片使用 `tf.summary.image(name='',tensor)`，其中 `name` 为变量的名字，`tensor` 为值。使用示例如下：

```
tf.summary.scalar('error',error)
tf.summary.histogram('weights',weight)
tf.summary.histogram('bias',bias)
```

(3) 合并变量

```
merged = tf.summary.merge_all()
```

(4) 运行合并变量

```
summary = sess.run(merged)
```

(5) 将 summary 写入事件文件

```
file_writer.add_summary(summary,i)
```

增加命名空间

为了使代码结构更加清晰，TensorBoard 图结构清楚，可以增加命名空间。

【示例】增加命名空间

```
import tensorflow as tf
def linear_regression():
    with tf.variable_scope('prepare_data'):
        #准备数据
        X = tf.random_normal(shape=[100,1],name='feature')
        y_true = tf.matmul(X,[[0.8]])+0.7
    with tf.variable_scope('create_model'):
        #构造模型
        #构造模型参数权重 weight 和偏移 bias
```

```

weight = tf.Variable(initial_value=tf.random_normal(shape=[1,1]),name='Weights')
bias = tf.Variable(initial_value=tf.random_normal(shape=[1,1]),name='bias')
y_predict = tf.matmul(X,weight)+bias
with tf.variable_scope('loss_function'):
    #构造损失函数
    error = tf.reduce_mean(tf.square(y_predict-y_true))
with tf.variable_scope('optimizer'):
    #优化损失
    optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.01).minimize(error)

#2.收集变量
tf.summary.scalar('error',error)
tf.summary.histogram('weights',weight)
tf.summary.histogram('bias',bias)

#3.合并变量
merged = tf.summary.merge_all()
#显示初始化变量
init = tf.global_variables_initializer()
#开始会话
with tf.Session() as sess:
    #运行初始化变量
    sess.run(init)
    #1.创建事件文件
    file_writer = tf.summary.FileWriter('e:/events/test',graph=sess.graph)
    print('查看训练前模型参数:权重: %f, 偏量: %f, 损失: %f'%(weight.eval(),bias.eval(),error.eval()))

    #开始训练
    for i in range(1000):
        sess.run(optimizer)
        print('训练第%d 次后模型参数:权重: %f, 偏量: %f, 损失: %f %'
              ((i+1),weight.eval(), bias.eval(), error.eval()))

    #4.运行合并变量

```

```
summary = sess.run(merged)

#5.将每次迭代后的变量写入事件文件

file_writer.add_summary(summary,i)


if __name__ == '__main__':
    linear_regression()
```

保存读取模型

(1) 保存模型

```
#创建 saver 对象
saver = tf.train.Saver()

saver.save(sess,'./ckpt/linear_regression.ckpt')
```

(2) 读取模型

```
#判断模型是否存在

ckpt = tf.train.get_checkpoint_state('./ckpt/')

if ckpt and ckpt.model_checkpoint_path:
    saver.restore(sess,'./ckpt/linear_regression.ckpt')

print('训练后模型参数:权重: %f, 偏量: %f, 损失: %f % (weight.eval(), bias.eval(),
error.eval()))
```

MNIST 数据集

MNIST 数据集简介

MNIST 数据集是机器学习领域中非常经典的一个数据集,由 60000 个训练样本和 10000 个测试样本组成, 每个样本都是一张 28 * 28 像素的灰度手写数字图片, 如下图所示:



MNIST 数据集可在 <http://yann.lecun.com/exdb/mnist/> 获取，它包含了四个部分：训练集、训练集标签、测试集、测试集标签。

文件名称	大小	内容
train-images-idx3-ubyte.gz	9,681 kb	55000 张训练集，5000 张验证集
train-labels-idx1-ubyte.gz	29 kb	训练集图片对应的标签
t10k-images-idx3-ubyte.gz	1,611 kb	10000 张测试集
t10k-labels-idx1-ubyte.gz	5 kb	测试集图片对应的标签

MNIST 中的每个图像都具有相应的标签，0 到 9 之间的数字表示图像中绘制的数字，用的是 one-hot 编码 `mn[0,0,0,0,0,0,1,0,0,0]`，`mnist.train.labels[55000,10]`。

加载 MNIST 数据集

直接下载下来的数据是无法通过解压或者应用程序打开的，因为这些文件不是任何标准的图像格式而是以字节的形式进行存储的，所以必须编写程序来打开它。

【示例】使用 TensorFlow 来读取数据及标签

```
from tensorflow.examples.tutorials.mnist import input_data
import matplotlib.pyplot as plt

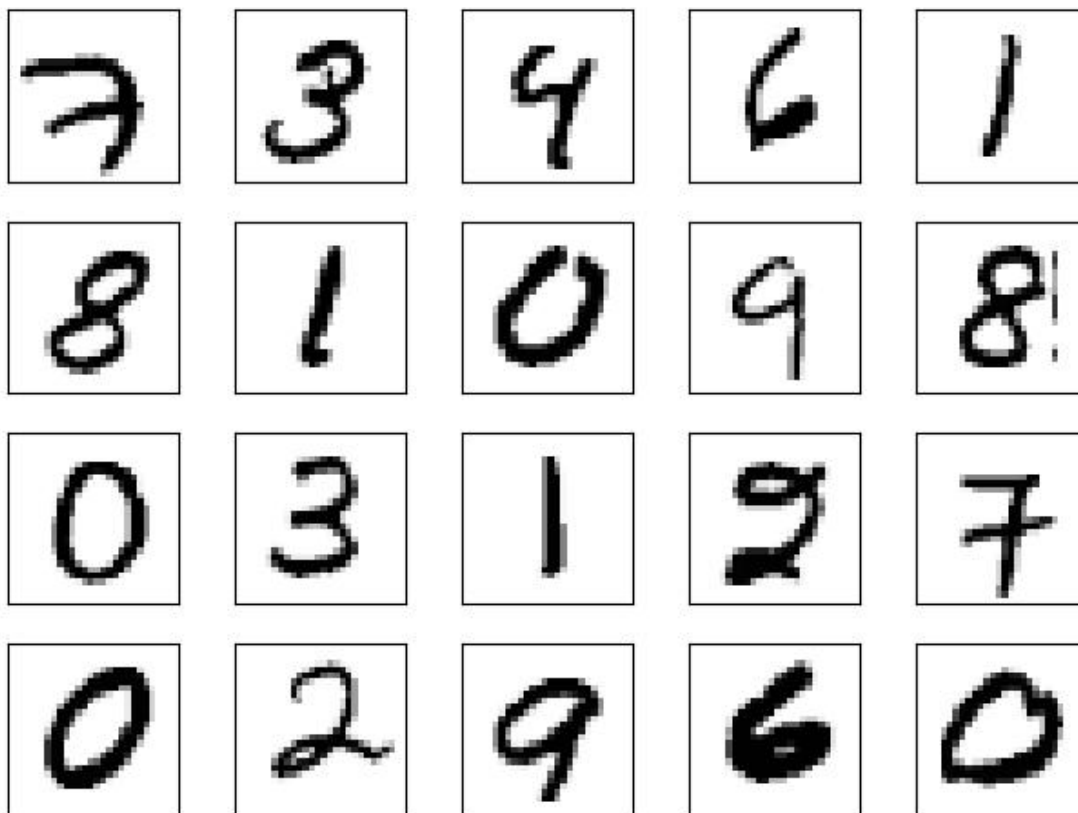
mnist = input_data.read_data_sets('E:\\soft\\MNIST_DATA', one_hot=True)

# 第一个参数指的是存放数据的文件夹路径，one_hot=True 为采用 one_hot 的编码方式
# 编码标签
```

```
#load data
train_X = mnist.train.images           #训练集样本
validation_X = mnist.validation.images  #验证集样本
test_X = mnist.test.images             #测试集样本

#labels
train_Y = mnist.train.labels           #训练集标签
validation_Y = mnist.validation.labels  #验证集标签
test_Y = mnist.test.labels             #测试集标签
print(train_X.shape,train_Y.shape)     #输出训练集样本和标签的大小
#查看数据，例如训练集中第一个样本的内容和标签
print(train_X[0])                      #是一个包含 784 个元素且值在[0,1]之间的向量
print(train_Y[0])
#获取数据集中 100 行
image,label = mnist.train.next_batch(100)
print('image.shape:',image.shape,'label.shape:',label.shape)
#可视化样本，下面是输出了训练集中前 20 个样本
fig, ax = plt.subplots(nrows=4,ncols=5,sharex='all',sharey='all')
ax = ax.flatten()
for i in range(20):
    img = train_X[i].reshape(28, 28)
    ax[i].imshow(img,cmap='Greys')
ax[0].set_xticks([])
ax[0].set_yticks([])
# 自动调整子图参数，使之填充整个图像区域
plt.tight_layout()
plt.show()
```

训练集中前 20 个样本图形如下：



手写数字识别

【示例】手写数字识别

```
import tensorflow as tf
import os

os.environ['TF_CPP_MIN_LOG_LEVEL']='2' #不想让警告的信息输出可以添加
from tensorflow.examples.tutorials.mnist import input_data

def mnist_demo():
    #1.准备数据

    mnist = input_data.read_data_sets('E:\\soft\\MNIST_DATA',one_hot=True)
    x = tf.placeholder(dtype = tf.float32,shape=[None,784])
    y_true = tf.placeholder(dtype= tf.float32,shape=[None,10])

    #2.构建模型
    weight = tf.Variable(initial_value=tf.random_normal(shape=[784,10]))
    bias = tf.Variable(initial_value=tf.random_normal(shape=[10]))
    y_predict = tf.matmul(x,weight)+bias

    #3.构造损失函数
```

```
error
=
tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=y_true, logits=y_predict))
#优化损失
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.01).minimize(error)
#初始化变量
init = tf.global_variables_initializer()
#开始会话
with tf.Session() as sess:
    sess.run(init)
    image, label = mnist.train.next_batch(100)
    print('训练前损失: %f'%(sess.run(error, feed_dict={x:image, y_true:label})))

    #开始训练
    for i in range(1000):
        op, loss = sess.run([optimizer, error], feed_dict={x:image, y_true:label})
        print('第%d 次训练的损失 loss 为%f'%(i+1, loss))

if __name__ == '__main__':
    mnist_demo()
```