Chapter 1

Tensorflow 基础

1.1 Tensorflow 基础函数

1.1.1 Variable

```
#tensorflow 1.2.1
import tensorflow as tf
var = tf.Variable(0)
add_operation = tf.add(var,1)
update_operation = tf.assign(var,add_operation)
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for _ in range(3):
        sess.run(update_operation)
        print(sess.run(var))
```

1.1.2 placeholder 使用

```
#tensorflow 1.2
import tensorflow as tf
x1 = tf.placeholder(dtype=tf.float32,shape=None)
y1 = tf.placeholder(dtype=tf.float32,shape=None)
z1 = x1+y1
x2 = tf.placeholder(dtype=tf.float32,shape=[2,1])
y2 = tf.placeholder(dtype=tf.float32,shape=[1,2])
z2 = tf.matmul(x2,y2)
with tf.Session() as sess:
    z1_value = sess.run(z1,feed_dict={x1:1,y1:2})
```

```
z1_value,z2_value = sess.run([z1,z2],feed_dict={x1:1,y1:2,x2:[[2],[2]],
                                                y2:[[3,3]]})
print(z1_value)
print(z2_value)
```

1.1.3 batch normalization

- 数据 x 为 Tensor。
- mean: 为 x 的均值，也是一个 Tensor。
- var: 为 x 的方差，也为一个 Tensor。
- offset: 一个偏移，也是一个 Tensor。
- scale: 缩放倍数，也是一个 Tensor。
- variable_epsilon, 一个不为 0 的浮点数。
- name: 操作的名字，可选。

batch normalization 计算方式是:

$$x = (x - \bar{x})/\sqrt{Var(x) + variable_{epsilon}} \tag{1.1}$$

$$x = x \times scale + offset \tag{1.2}$$

$$\tag{1.3}$$

$$均值 : \bar{x} = \frac{1}{m}\Sigma_{i=1}^{m} x_i \tag{1.4}$$

$$方差 : \sigma^2 = \frac{1}{m}\Sigma_{i=1}^{m}(x_i - \bar{x}) \tag{1.5}$$

1.1.4 **常见的的激活函数**

- relu
- sigmoid
- tanh
- elu

- bias_add
- relu6
- softplus
- softsign

1.2 relu 函数

1.2.1 relu

relu 函数在自变量 x 小于 0 时值全为 0, 在 x 大于 0 时，值和自变量相等。

```
import tensorflow as tf
import matplotlib.pyplot as plt
x = tf.linspace(-10.,10.,100)
y = tf.nn.relu(x)
with tf.Session() as sess:
  [x,y] = sess.run([x,y])
plt.plot(x,y,'r',6,6,'bo')
plt.title('relu')
ax = plt.gca()
ax.annotate("",
            xy=(6, 6), xycoords='data',
            xytext=(6, 4.5), textcoords='data',
            arrowprops=dict(arrowstyle="->",
                            connectionstyle="arc3"),
            )
ax.annotate("",xy=(6,6),xycoords='data',
            xytext=(10, 6), textcoords='data',
            arrowprops=dict(arrowstyle="->",
                            connectionstyle="arc3"),

)
ax.grid(True)
plt.xlabel('x')
plt.ylabel('relu(x)')
plt.savefig('relu.png',dpi = 600)
```

1.2.2 relu6

relu6 函数和 relu 不同之处在于在 x 大于等于 6 的部分值保持为 6。

```
import tensorflow as tf
import matplotlib.pyplot as plt
x = tf.linspace(-10.,10.,100)
y = tf.nn.relu6(x)
with tf.Session() as sess:
  [x,y] = sess.run([x,y])
plt.plot(x,y,'r',6,6,'bo')
plt.title('relu6')
ax = plt.gca()
ax.annotate("",
            xy=(6, 6), xycoords='data',
            xytext=(6, 4.5), textcoords='data',
            arrowprops=dict(arrowstyle="->",
                            connectionstyle="arc3"),
            )
ax.grid(True)
plt.xlabel('x')
plt.ylabel('relu6(x)')
plt.savefig('relu6.png',dpi = 600)
```

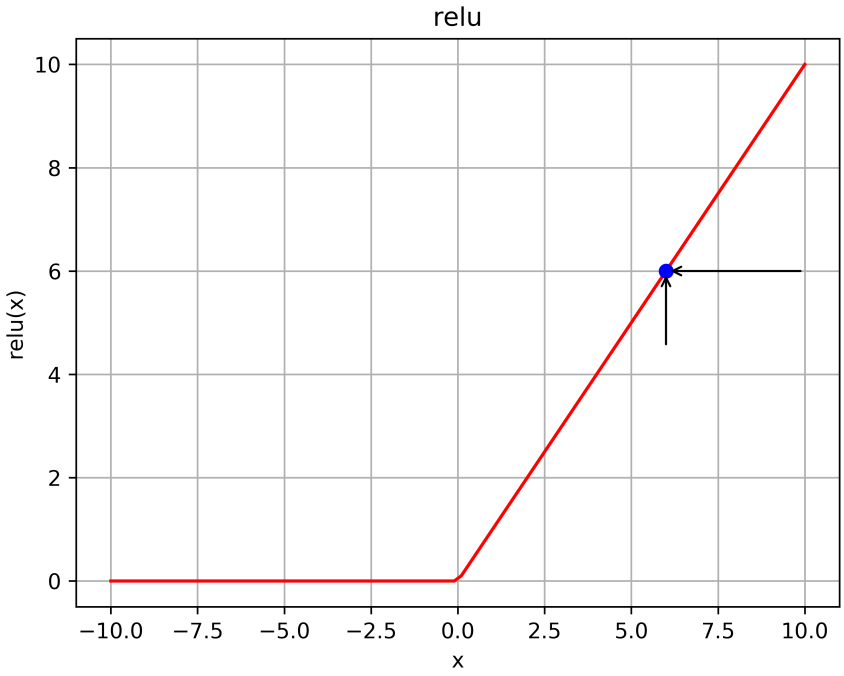


图 1.1: relu

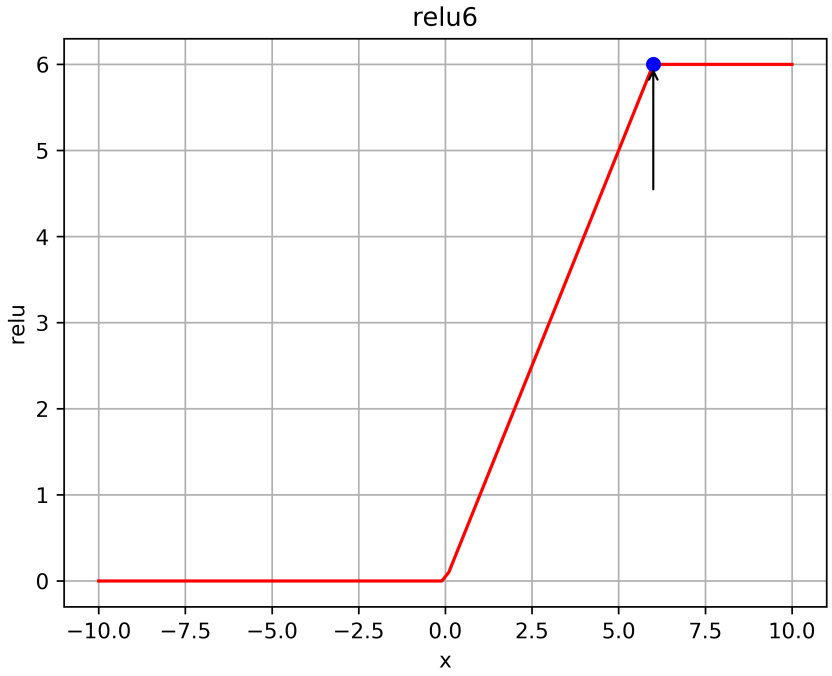


图 1.2: relu6

1.2.3 sigmoid

```
import tensorflow as tf
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
x = tf.linspace(-10.,10.,100)
y1 = tf.nn.sigmoid(x)
y2 = tf.nn.tanh(x)
red_patch = mpatches.Patch(color = 'red',label = 'sigmoid')
blue_patch = mpatches.Patch(color = 'blue',label = 'tanh')
with tf.Session() as sess:
  [x,y1,y2] = sess.run([x,y1,y2])
plt.plot(x,y1,'r',x,y2,'b')
ax = plt.gca()
ax.annotate(r"$tanh(x) = \frac{1-^{-2x}}{1+e^{-x}}$",
      xy=(0,0),xycoords="data",
      xytext=(1,0),textcoords="data",
      arrowprops=dict(arrowstyle="->",
      connectionstyle="arc3"),
```

```
)
ax.annotate(r"$sigmoid(x) = \frac{1}{1+e^{-x}}$",
      xy=(0,0.5),xycoords="data",
      xytext=(1,0.5),textcoords="data",
      arrowprops=dict(arrowstyle="->",
      connectionstyle="arc3"),
)
plt.xlabel('x')
plt.grid(True)
plt.legend(handles = [red\_patch,blue\_patch])
plt.savefig('activate.png',dpi=600)
```

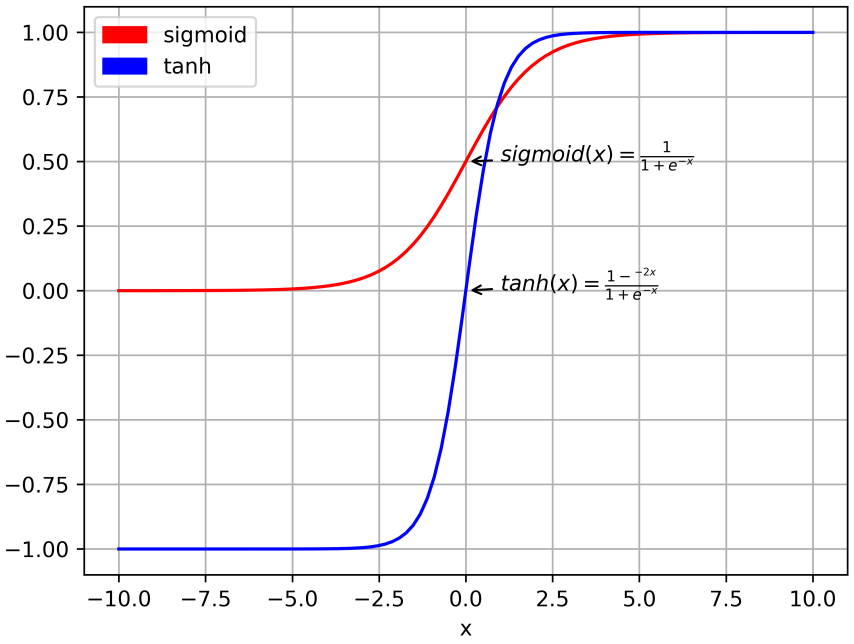


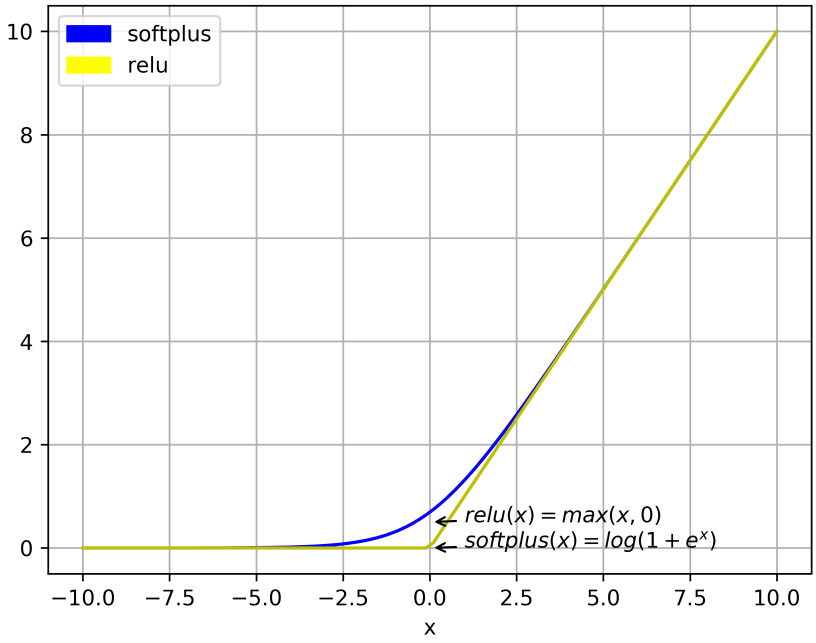
图 1.3: activate_fun

1.2.4 relu 和 softplus

```
import tensorflow as tf
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
x = tf.linspace(-10.,10.,100)
```

```
y2 = tf.nn.softplus(x)
y3 = tf.nn.relu(x)
blue_patch = mpatches.Patch(color = 'blue',label = 'softplus')
yellow_patch = mpatches.Patch(color = 'yellow',label = 'relu')
with tf.Session() as sess:
  [x,y2,y3] = sess.run([x,y2,y3])
plt.plot(x,y2,'b',x,y3,'y')
ax = plt.gca()
plt.xlabel('x')
ax.annotate(r"$softplus(x)=log(1+e^x)$",
      xy=(0,0),xycoords="data",
      xytext=(1,0),textcoords="data",
      arrowprops=dict(arrowstyle="->",
      connectionstyle="arc3"),
)
ax.annotate(r"$relu(x)=max(x,0)$",
      xy=(0,0.5),xycoords="data",
      xytext=(1,0.5),textcoords="data",
      arrowprops=dict(arrowstyle="->",
      connectionstyle="arc3"),
)

plt.grid(True)
plt.legend(handles = [blue_patch,yellow_patch])
plt.savefig('relu_softplus.png',dpi=600)
```



1.2.5 dropout

将神经元以概率 keepi_prob 绝对是否被抑制。如果被抑制该神经元的输出为 0 如果不被抑制，该神经元的输出将被放大到原来的 1/keep_prop。默认情况下，每个神经元是否被抑制是相互独立的。但是是否被抑制也可以通过 noise_shape 来调节。当 noise_shape[i]=shape(x)[i] 时,x 中的元素相互独立。如果 shape(x)=[k,1,1,n], 那么每个批通道都是相互独立的，但是每行每列的数据都是关联的，也就是说要么都为 0, 要么还是原来的值。

```
import tensorflow as tf
a = tf.constant([[-1.,2.,3.,4.]])
with tf.Session() as sess:
    b = tf.nn.dropout(a,0.5,noise_shape=[1,4])
    print(sess.run(b))
    c = tf.nn.dropout(a,0.5,noise_shape=[1,1])
    print(sess.run(c))
```

```
[[-2.  0.  0.  8.]]
[[-0.  0.  0.  0.]]
```

当输入数据特征相差明显时，用 tanh 效果会很好，但在循环过程中会不断扩大特征效果并显示出来。当特征相差不明显时，sigmoid 效果比较好。同时，用 sigmoid 和 tanh 作为激活函数时，需要对输入进行规范化，否则激活厚的值全部进入平坦区，隐藏层的输出会趋同，丧失原来的特征表达，而 relu 会好很多，优势可以不需要输入规范化来避免上述情况。因此，现在大部分卷积神经网络都采用 relu 作为激活函数。

1.3 卷积函数

tf.nn.conv2d(input,filter,padding,stride=None,diation_rate=Nonei 每 name = None,data_format=None)

- input: 一个 tensor，数据类型必须是 float32, 或者是 float64
- filter: 一个 tensor, 数据类型必须和 input 相同。
- strides: 一个长度为 4 的一组证书类型数组，每一维对应 input 中每一维对应移动的步数，strides[1] 对应 input[1] 移动的步数。
- padding: 有两个可选参数'VALID'（输入数据维度和输出数据维度不同）和'SAME'（输入数据维度和输出数据维度相同）
- use_cudnn_on_gpu: 一个可选的布尔值，默认情况下时 True。
- name: 可选，操作的一个名字。

```
import tensorflow as tf
input_data = tf.Variable(tf.random_normal(shape = [10,9,9,3],mean=0,stddev=1
                                          ),dtype = tf.float32)
kernel = tf.Variable(tf.random_normal(shape = [2,2,3,2],mean = 0,stddev=1,
                                          dtype=tf.float32))

y = tf.nn.conv2d(input_data,kernel,strides=[1,1,1,1],padding='SAME')
init = tf.global_variables_initializer()
with tf.Session() as sess:
    sess.run(init)
    print(sess.run(y).shape)
```

输出形状为 [10,9,9,2]。

1.4 池化

池化函数	功能
① tf.nn.avg_pool(value,ksize,strides,padding,data_format='NHWC',name =None)	平均池化
② tf.nn.max_pool(value,ksize,strides,padding,data_format='NHWC',name =None)	最大池化
③ tf.nn.max_pool_with_argmax(input,ksize,strides,padding,Targmax=None,name =None)	最大池化
④ tf.nn.avg_pool3d(input,ksize,strides,padding,name =None)	三维状态
⑤ tf.nn.max_pool3d(input,ksize,strides,padding,name =None)	三维状态
⑥ tf.nn.fractionan_avg_pool(value,pooling_ratio,pseudo_random=None,overlapping=None,deterministic = None,seed2=None,name = None)	三维下的
⑦ tf.nn.avg_max_pool(value,pooling_ratio,pseudo_random=None,overlapping=None,deterministic = None,seed2=None,name = None)	三维状态
⑧ tf.nn.pool(input,window_shape,pool_typing,padding,dilation_rate = None,strides=None,name=None,data_format=None)	执行一个

- value: 一个四维 Tensor, 维度时 [batch,height,width,chennels]。
- ksize: 一个长度不小于 4 的整型数据，每一位上的值对应于输入数据 Tensor 中每一维窗口对应值。
- stride: 一个长度不小于 4 的整型列表。该参数指定窗口在输入数据 Tensor 每一维上的步长。
- padding: 一个字符串，取值为 SAME 或者 VALID。
- data_format:NHWC。

1.5 常见的分类函数

tf.nn.sigmoid_cross_entropy_with_logits(logits,targets,name=None)

- logits:[batch_size,num_classes]
- targets:[batch_size,size]
- 输出：loss[batch_size,num_classes]

最后已成不需要进行 sigmoid 操作。

tf.nn.softmax(logits,dim=-1,name=None): 计算 Softmax

$$softmax = \frac{x^{logits}}{reduce\_sum(e^{logits}, dim)}$$

tf.nn.log_softmax(logits,dim=-1,name = None) 计算 log softmax

$$logsoftmax = logits - log(reduce\_softmax(exp(logits), dim))$$

tf.nn.softmax_cross_entropy_with_logits(_setinel=None,labels=None,logits=None,dim=-1,name=None) 输出 loss:[batch_size] 保存的时 batch 中每个样本的交叉熵。
tf.nn.sparse_softmax_cross_entropy_with_logic(logits,labels,name=None)

- logits: 神经网络最后一层的结果。
- 输入 logits:[batch_size,num_classes],labels:[batch_size], 必须在 [0,num_classes]
- loss[batch], 保存的是 batch 每个样本的交叉熵。

1.6 优化方法

- tf.train.GradientDescentOptimizer
- tf.train.AdadeltaOptimizer
- tf.train.AdagradDAOptimizer
- tf.train.AdagradOptimizer
- tf.train.MomentumOptimizer
- tf.train.AdamOptimizer
- tf.train.FtrlOptimizer
- tf.train.RMSPropOptimizer

1.6.1 BGD

BGD(batch gradient descent) 批量梯度下降。这种方法是利用现有的参数对训练集中的每一个输入生成一个估计输出 y_i, 然后跟实际的输出 y_i 比较，统计所有的误差，求平均后的到平均误差作为更新参数的依据。啊他的迭代过程是:

1. 提取巡检集中所有内容 $\{x_1, \ldots, x_n\}$, 以及相关的输出 y_i;

2. 计算梯度和误差并更新参数。

这种方法的优点是：使用所有数据计算，都保证收敛，并且并不需要减少学习率缺点是每一步需要使用所有的训练数据，随着训练的进行，速度会变慢。那么如果将训练数据拆分成一个个 batch, 每次抽取一个 batch 数据更新参数，是不是能加速训练？这就是 SGD。

1.6.2 SGD

SGD(stochastic gradient descent): 随机梯度下降。这种方法的主要思想是将数据集才分成一个个的 batch，随机抽取一个 batch 计算并更新参数，所以也称为 MBGD(minibatch gradient descent) SGD 在每次迭代计算 mini-batch 的梯度，然后队参数进行更新。和 BGD 相比，SGD 在训练数据集很大时也能以较快的速度收敛，但是它有两个缺点：

1. 需要手动调整学习率，但是选择合适的学习率比较困难。尤其在训练时，我们常常想队常出现的特征更新速度快点，队不长出现的特征更新速度慢些，而 SGD 对更新参数时对所有参数采用一样的学习率，因此无法满足要求。

2. SGD: 容易收敛到局部最优。

1.6.3 momentum

Momentum 时模拟物理学中的动量概念，更新时在一定程度上保留之前的更新方向，利用当前批次再次微调本次更新参数，因此引入了一个新的变量 v，作为前几次梯度的累加。因此，momentum 能够更新学习率，在下降初期，前后梯度方向一致时能加速学习：在下降的中后期，在局部最小值附近来回振荡，能够抑制振荡加快收敛。

1.6.4 Nesterov Momentum

标准的 Monentum 法首先计算一个梯度，然后子啊加速更新梯度的方向进行一个大的跳跃 Nesterov 首先在原来加速的梯度方向进行一个大的跳跃，然后在改为值设置计算梯度值，然后用这个梯度值修正最终的更新方向。

1.6.5 Adagrad

Adagrade 能够自适应的为哥哥参数分配不同的学习率，能够控制每个维度的梯度方向，这种方法的优点是能实现学习率的自动更改，如果本次更新时梯度大，学习率就衰减得快，如果这次更新时梯度小，学习率衰减得就慢些。

1.6.6 RMSprop

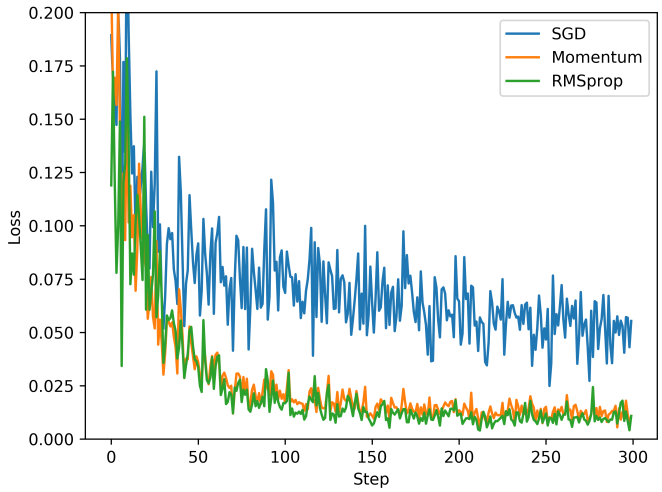
和 Momentum 类似，通过引入衰减系数使得每个回合都衰减一定比例。在实践中，对循环神经网络效果很好。

1.6.7 Adam

名称来自自适应矩阵 (adaptive moment estimation).Adam 更均损失函数针对每个参数的一阶矩，二阶矩估计动态调整每个参数的学习率。

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
tf.set_random_seed(0)
np.random.seed(0)
LR = 0.01
BATCH_SIZE = 32
x = np.linspace(-1,1,100).reshape(-1,1)
noise = np.random.normal(0,0.1,size=x.shape)
y = np.power(x,2)+noise
class Net:
    def __init__(self,opt,**kwargs):
        self.x = tf.placeholder(tf.float32,[None,1])
        self.y = tf.placeholder(tf.float32,[None,1])
        l = tf.layers.dense(self.x,20,tf.nn.relu)
        out = tf.layers.dense(l,1)
        self.loss = tf.losses.mean_squared_error(self.y,out)
        self.train = opt(LR,**kwargs).minimize(self.loss)
net_SGD = Net(tf.train.GradientDescentOptimizer)
net_momentum = Net(tf.train.MomentumOptimizer,momentum=0.9)
net_RMSprop = Net(tf.train.RMSPropOptimizer)
net_Adam = Net(tf.train.AdamOptimizer)
nets = [net_SGD,net_momentum,net_RMSprop,net_Adam]
sess = tf.Session()
sess.run(tf.global_variables_initializer())
losses_his = [[],[],[]]
```

```
for step in range(300):
    index = np.random.randint(0,x.shape[0],BATCH_SIZE)
    b_x = x[index]
    b_y = y[index]
    for net,l_his in zip(nets,losses_his):
        _,l = sess.run([net.train,net.loss],{net.x:b_x,net.y:b_y})
        l_his.append(l)
labels = ['SGD','Momentum','RMSprop','Adam']
for i,l_his in enumerate(losses_his):
    plt.plot(l_his,label=labels[i])
plt.legend(loc='best')
plt.xlabel('Step')
plt.ylabel('Loss')
plt.ylim(0,0.2)
plt.savefig('Opt.png',dpi=600)
```



1.6.8 常用的度量函数 tf.metrics

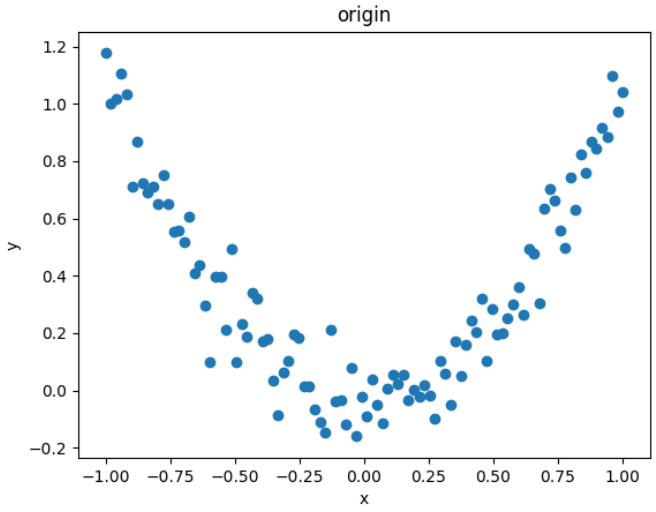
accuracy(labels,predictions,weights=None,metrics_collections,updates_collections=None,name=None)

- labels:Tensor，和 predictions 的形状相同，代表真实值。
- predictions:Tensor，代表预测值。
- weights:Tensor，rank 可以为 0 或者 labels 的 rank，必须能和 label 广播（所有的维度必须是 1, 或者和 labels 维度相同）

- metrics_collection:accuracy 应该被增加的一个 collectiobn 列表选项。
- update_collections:update_op 应该添加的选项列表。
- name:Variable_scope 名字选项。
- accuracy: 返回值 Tensor，代表精度，总共预测对的和总数的商。
- update_op: 返回值适当增加 total 和 count 变量和 accuracy 匹配。
- ValueError: 异常如果 predictions 和 labels 有不同的形状，或者 weight 不是 None 它的形状不合 prediction 匹配，或者 metrics_collections 会哦这 updates_collections 不是一个 list 或者 tuple。

1.6.9 构造简单的神经网络拟合数据

原始数据为 $y = x^2$ 的基础上添加随机噪声。原始数据的散点图如下




```
#tensorflow 1.2.1
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
tf.set_random_seed(0)
np.random.seed(0)
#生成数据
step = 100
x = np.linspace(-1,1,step).reshape(-1,1)
```

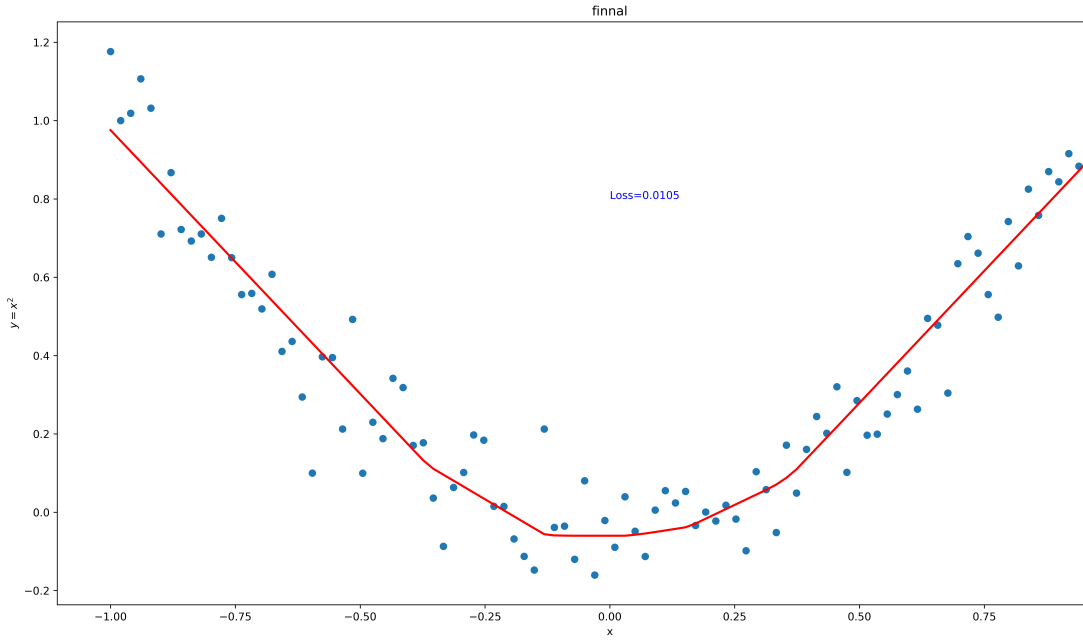
```
noise = np.random.normal(0,0.1,size=x.shape)
y = np.power(x,2)+noise

tf_x = tf.placeholder(tf.float32,x.shape)
tf_y = tf.placeholder(tf.float32,x.shape)
l1 =  tf.layers.dense(tf_x,10,tf.nn.relu)
output = tf.layers.dense(l1,1)

loss = tf.losses.mean_squared_error(tf_y,output)
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.5)
train_op = optimizer.minimize(loss)

sess = tf.Session()
sess.run(tf.global_variables_initializer())
plt.ion()
for step in range(100):
    _,l,pred = sess.run([train_op,loss,output],{tf_x:x,tf_y:y})
    if step%5==0:
        plt.cla()
        plt.scatter(x,y)
        plt.title(r'$y=x^2+noise$')
        plt.plot(x,pred,'r-',lw=2)
        plt.text(0,0.8,'Loss=%.4f'%l,fontdict={'size':10,'color':'blue'})
        plt.xlabel("x")
        plt.ylabel(r"$y=x^2$")
        plt.pause(0.1)
plt.ioff()
plt.show()
```

最终拟合数据:


finnal
Loss=0.0105
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
$y = x^2$
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
x

Chapter 2

数据的存储和加载

```
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
tf.set_random_seed(0)
np.random.seed(0)
x = np.linspace(-1,1,100).reshape(-1,1)
noise = np.random.normal(0,0.1,size=x.shape)
y = np.power(x,2)+noise
def gendata():
    t = np.linspace(-1,1,100).reshape(-1,1)
def save():
    print('This is save')
    tf_x = tf.placeholder(tf.float32,x.shape)
    tf_y = tf.placeholder(tf.float32,y.shape)
    l = tf.layers.dense(tf_x,10,tf.nn.relu)
    o = tf.layers.dense(l,1)
    loss = tf.losses.mean_squared_error(tf_y,o)
    train_op = tf.train.GradientDescentOptimizer(learning_rate=0.5).minimize
                                            (loss)
    sess = tf.Session()
    sess.run(tf.global_variables_initializer())
    saver = tf.train.Saver()
    for step in range(100):
        sess.run(train_op,{tf_x:x,tf_y:y})
    saver.save(sess,'params',write_meta_graph=False)
    pred,l = sess.run([o,loss],{tf_x:x,tf_y:y})
    plt.figure(1,figsize=(10,5))
    plt.subplot(121)
    plt.scatter(x,y)
```

```
    plt.plot(x,pred,'r-',lw=5)
    plt.text(-1,1.2,'save loss=%.4f'%l,fontdict={'size':15,'color':'red'})
def reload():
    print('This is reload')
    tf_x = tf.placeholder(tf.float32,x.shape)
    tf_y = tf.placeholder(tf.float32,y.shape)
    l_ = tf.layers.dense(tf_x,10,tf.nn.relu)
    o_ = tf.layers.dense(l_,1)
    loss_ = tf.losses.mean_squared_error(tf_y,o_)
    sess = tf.Session()
    saver = tf.train.Saver()
    saver.restore(sess,'params')
    pred,l = sess.run([o_,loss_],{tf_x:x,tf_y:y})
    plt.subplot(122)
    plt.scatter(x,y)
    plt.plot(x,pred,'r-',lw=5)
    plt.text(-1,1.2,'Reload Loss=%.4f'%l,fontdict={'size':15,'color':'red'})
    plt.show()
save()
tf.reset_default_graph()
reload()
```

2.1 模型存储和加载

- 生成 checkpoint 文件，扩展名一般为.ckpt, 通过在 tf.train.Saver 对象上调用 Saver.saver() 生成。它包含权重和其他程序中定义的变量，不包含 图的结构。如果需要在另一个程序中十一哦嗯，需要吃哦嗯见图形结构，并告诉 Tensorflow 如何处理这些权重。
- 生成 (graph proto file)，这是一个二进制文件，扩展名一般是.pb, 用 tf.train.write_graph() 保存每，只包含图形结构，不包含全职哦嗯，然后使用 tf.import_graph_def() 加载 图形。

Chapter 3

Tensorflow API

3.1 tf

tf.squeeze(input,axis=None,name=None,squeeze_dims=None) 说明: 从指定的 Tensor 中移除 1 维度。

- input:tensor, 输入 Tensor。
- axis: 列表, 指定需要移除的位置的列表, 默认为空列表 [], 索引从 0 开始 squeeze 不为 1 的索引会报错。
- name:caozuoide 名字
- squeeze_dims: 否决当前轴的参数。
- 返回一个 Tensor, 形状和 input 相同, 包含和 input 相同的数据, 但是不包含有 1 的元素。
- 异常: squeeze_dims 和 axis 同时指定时会有 ValueError。

't' is a tensor of shape [1, 2, 1, 3, 1, 1]
shape(squeeze(t)) ==> [2, 3]
't' is a tensor of shape [1, 2, 1, 3, 1, 1]
shape(squeeze(t, [2, 4])) ==> [1, 2, 3, 1]