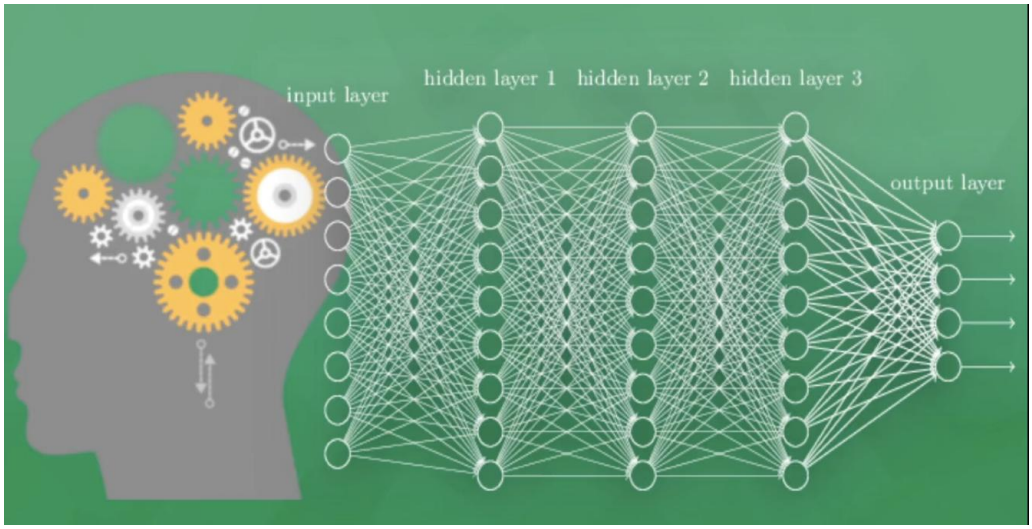
神经网络的专题学习——WittPeng（学自莫烦 python）

一、神经网络

神经网络的构建是在不断优化中完成的，有正向和反向的传播和改正。



二、Tensorflow

TensorFlow 是 Google 开发的一款神经网络的 Python 外部的结构包，也是一个采用数据流图来进行数值计算的开源软件库.TensorFlow 让我们可以先绘制计算结构图，也可以称是一系列可人机交互的计算操作，然后把编辑好的 Python 文件 转换成 更高效的 C++，并在后端进行计算.

CPU 版本的：

```
# python 2+ 的用户:
$ pip install tensorflow
# python 3+ 的用户:
$ pip3 install tensorflow
```

tensorflow 做的就是不断在拟合数据，而越拟合也代表神经网络的效果越好。tensorflow 的本意就是 tensor（张量）在 flow（流动），官网有个动画可以查看一下。节点（Nodes）在图中表示数学操作,图中的线（edges）则表示在节点间相互联系的多维数据数组，即张量（tensor). 训练模型时 tensor 会不断的从数据流图中的一个节点 flow 到另一节点。

```
import tensorflow as tf
import numpy as np

#creat data
x_data=np.random.rand(100).astype(np.float32)
y_data=x_data*0.1+0.3

#create tensorflow struct begin#
Weights = tf.Variable(tf.random_uniform([1],-1.0,1.0))#1wei,between[-1],1
biases = tf.Variable(tf.zeros([1]))
#The neural network is constantly learning the parameters in the right direction.
y = Weights*x_data + biases
```

```
#the loss will be more little
loss = tf.reduce_mean(tf.square(y - y_data))
#Back propagation error
optimizer = tf.train.GradientDescentOptimizer(0.5)
train = optimizer.minimize(loss)

init = tf.initialize_all_variables()
#create tensorflow struct end#

sess = tf.Session()
sess.run(init)     #Very important


for step in range(222):
    sess.run(train)
    if step % 20 ==0:
        print step,sess.run(Weights),sess.run(biases)
```

输出结果：

```
0 [0.2997378] [0.2618141]
20 [0.14730324] [0.27542576]
40 [0.11294082] [0.2932772]
60 [0.10354023] [0.29816085]
80 [0.1009685] [0.29949686]
100 [0.10026497] [0.29986235]
120 [0.10007248] [0.29996237]
140 [0.10001983] [0.2999897]
160 [0.10000544] [0.29999718]
180 [0.10000146] [0.29999924]
200 [0.10000041] [0.2999998]
220 [0.10000011] [0.29999995]
```

可以看到，误差越来越小，越来越接近真实的斜率和截距。

关于上方出现的 Session，顾名思义这是控制函数运行、输出等的会话。

```
import tensorflow as tf

matrix1 = tf.constant([[3,3]])
matrix2 = tf.constant([[2],
                       [2]])
product = tf.matmul(matrix1,matrix2)

#method 1
sess = tf.Session()
result = sess.run(product)
print "method1's result"
```

```
print result
sess.close()

#method 2
with tf.Session() as sess:
    result2 = sess.run(product)
    print "method2's result"
    print result2
```

输出：

```
method1's result
[[12]]
method2's result
[[12]]
```

继续学习 Variable 变量.Variable的使用比较特殊：

```
import tensorflow as tf

state = tf.Variable(0,name='counter')
print state.name
```

输出：

```
counter:0
```

继续，现在进行加 1 操作，并设置输出：

```
import tensorflow as tf

state = tf.Variable(0,name='counter')
one = tf.constant(1)

new_value = tf.add(state , one)
update = tf.assign(state,new_value)

init =tf.initialize_all_variables()

with tf.Session() as sess:
    sess.run(init)
    for _ in range(3):
        sess.run(update)
        print sess.run(state)
```

输出：

```
1
2
3
```

Placeholder 传入值

实例：

```
import tensorflow as tf

input1 = tf.placeholder(tf.float32)
input2 = tf.placeholder(tf.float32)

output = tf.multiply(input1,input2)

with tf.Session() as sess:
    print sess.run(output,feed_dict={input1:[7.],input2:[2.]})
```

输出：

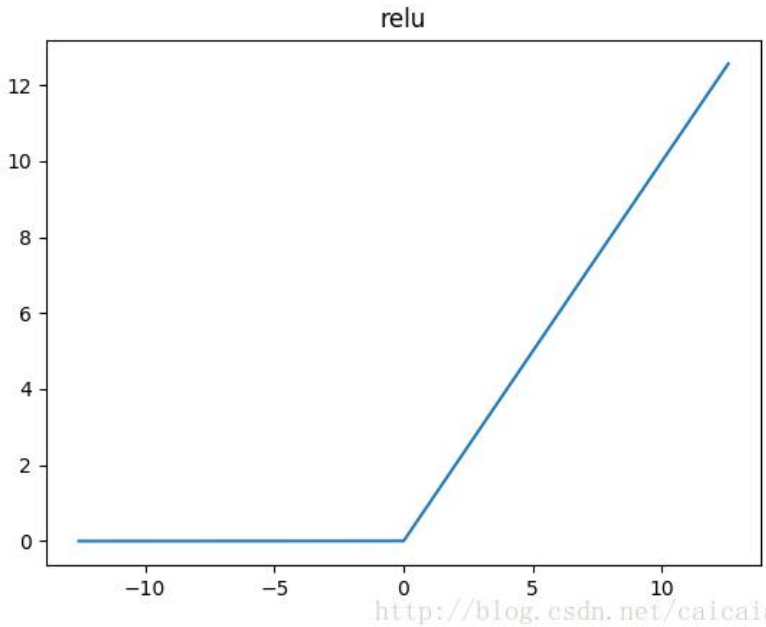
```
[14.]
```

使用 placeholder 即为在运行的时候再将值传入函数。

激励函数的学习：

激励函数的必要性：激励函数是为了解决生活中的非线性方程关系，如 relu、sigmoid、tanh。卷积神经网络中推荐 relu，循环神经网络推荐 relu、tanh。激励函数可以有一定的筛选和关注程度的调节的作用。

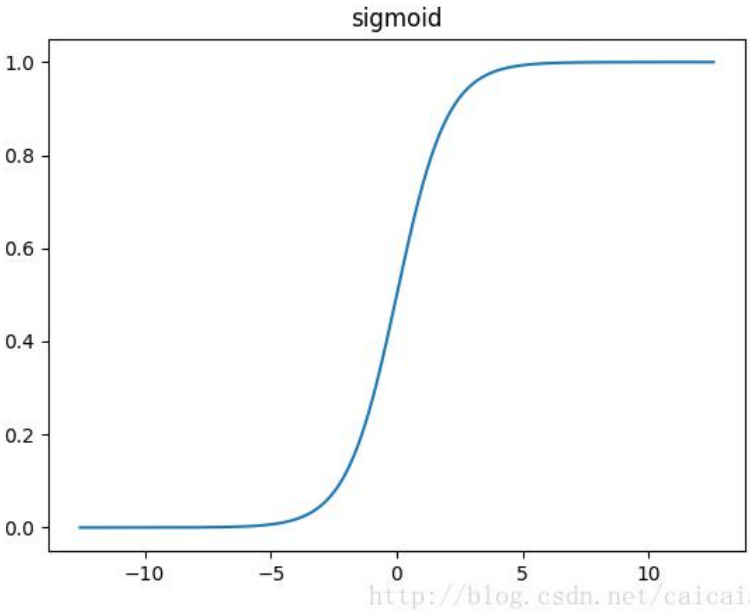
Relu



如果值小于 0 就一直为 0，大于 0 就是输出那个值。从我们的大脑中神经元来类比为：我们大脑中某神经元对于某信号的刺激太小的话，就一直处于睡眠状态，而如果此输入的信号激起了此神经元，则刺激的强度就跟输入信息的强度成正比。

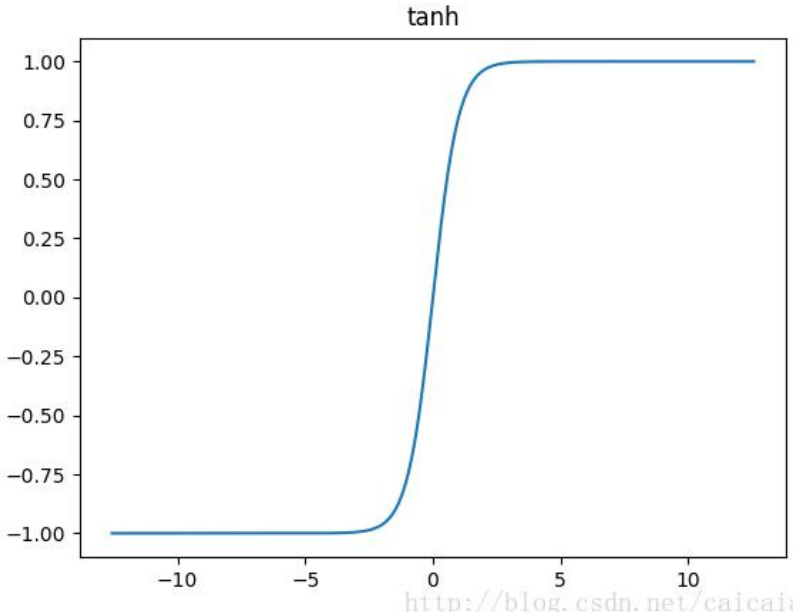
Sigmoid

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



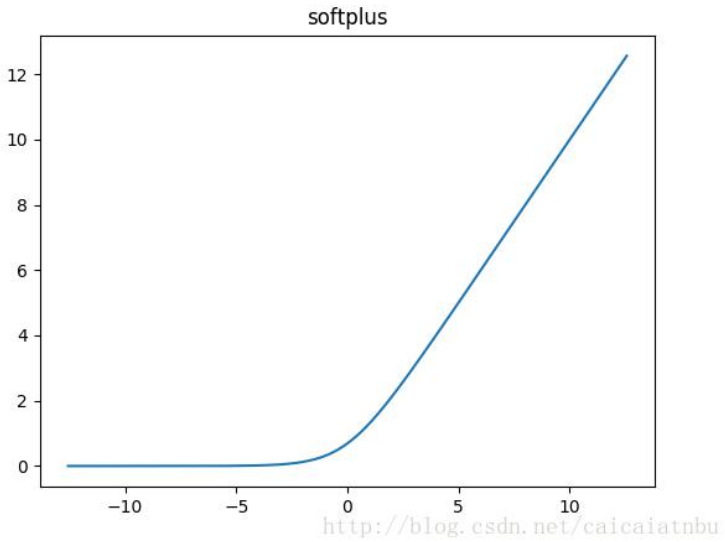
tanh

$$\tanh(x)=\frac{\sinh x}{\cosh x}=\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}=\frac{1-e^{-2x}}{1+e^{-2x}}$$



softplus 函数可以看作是 relu 函数的平滑版本，公式和函数图像如下：

$$f(x)=\log\left(1+e^{x}\right)$$



3.建造神经网络

添加层 def add_layer()

函数定义如下：

```
import tensorflow as tf

def add_layer(inputs,in_size,out_size,activation_function=None):
    Weights = tf.Variable(tf.random_normal([in_size,out_size]))
    biases = tf.Variable(tf.zeros([1,out_size])+0.1)
    Wx_plus_b = tf.matmul(inputs,Weights)+biases
    if activation_function is None:
        outputs = Wx_plus_b
    else:
        outputs = activation_function(Wx_plus_b)
    return outputs
```

建造神经网络

```
import tensorflow as tf
import numpy as np

def add_layer(inputs,in_size,out_size,activation_function=None):
    Weights = tf.Variable(tf.random_normal([in_size,out_size]))
    biases = tf.Variable(tf.zeros([1,out_size])+0.1)
    Wx_plus_b = tf.matmul(inputs,Weights)+biases
    if activation_function is None:
        outputs = Wx_plus_b
    else:
        outputs = activation_function(Wx_plus_b)
    return outputs
```

```
#Only one neuron in the input and output layers
x_data = np.linspace(-1,1,300, dtype=np.float32)[:, np.newaxis]
noise = np.random.normal(0, 0.05, x_data.shape).astype(np.float32)
y_data = np.square(x_data) - 0.5 + noise

xs = tf.placeholder(tf.float32,[None,1])
ys = tf.placeholder(tf.float32,[None,1])

#The following set the hidden layer has 10 nodes.
l1 = add_layer(xs,1,10,activation_function = tf.nn.relu)
prediction = add_layer(l1,10,1,activation_function = None)

loss = tf.reduce_mean(tf.reduce_sum(tf.square(ys - prediction),reduction_indices =
[1]))
train_step = tf.train.GradientDescentOptimizer(0.1).minimize(loss)



init = tf.initialize_all_variables()
sess = tf.Session()
sess.run(init)

for i in range(1000):
    sess.run(train_step,feed_dict={xs:x_data,ys:y_data})
    if i % 1000 :
        print sess.run(loss,feed_dict={xs:x_data,ys:y_data})
```

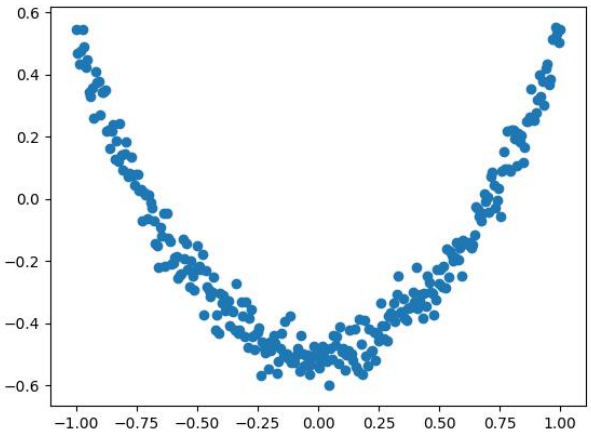
输出：

```
0.31339929
0.19001633
0.12825823
0.093823485
0.07296839
……
0.0028053746
0.0028052246
0.002805075
0.0028049266
0.0028047797
0.0028046337
0.0028044889
0.0028043608
0.002804234
0.002804108
0.0028039825
```

实现结果可视化：

输出训练前的数据

```
fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.scatter(x_data,y_data)
plt.show()
```



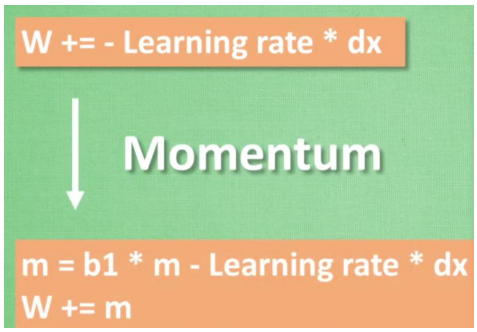

```
import tensorflow as tf
import numpy as np
import matplotlib
matplotlib.use('TkAgg')
import matplotlib
import matplotlib.pyplot as plt
from pylab import *

def add_layer(inputs,in_size,out_size,activation_function=None):
    Weights = tf.Variable(tf.random_normal([in_size,out_size]))
    biases = tf.Variable(tf.zeros([1,out_size])+0.1)
    Wx_plus_b = tf.matmul(inputs,Weights)+biases
    if activation_function is None:
        outputs = Wx_plus_b
    else:
        outputs = activation_function(Wx_plus_b)
    return outputs

#Only one neuron in the input and output layers
x_data = np.linspace(-1,1,300, dtype=np.float32)[:, np.newaxis]
noise = np.random.normal(0, 0.05, x_data.shape).astype(np.float32)
y_data = np.square(x_data) - 0.5 + noise
```

```
xs = tf.placeholder(tf.float32,[None,1])
ys = tf.placeholder(tf.float32,[None,1])

#The following set the hidden layer has 10 nodes.
l1 = add_layer(xs,1,10,activation_function = tf.nn.relu)
prediction = add_layer(l1,10,1,activation_function = None)

loss = tf.reduce_mean(tf.reduce_sum(tf.square(ys - prediction),reduction_indices =
[1]))
train_step = tf.train.GradientDescentOptimizer(0.1).minimize(loss)


init = tf.initialize_all_variables()
sess = tf.Session()
sess.run(init)

for i in range(1000):
    sess.run(train_step,feed_dict={xs:x_data,ys:y_data})
    if i % 1000 :
        print sess.run(loss,feed_dict={xs:x_data,ys:y_data})

fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.scatter(x_data,y_data)
plt.show(block=False)

for i in range(1000):
    sess.run(train_step,feed_dict ={xs:x_data,ys:y_data})
    if i % 50 ==0:
        try:
            ax.lines.remove(lines[0])
        except Exception:
            pass
        prediction_value = sess.run(prediction,feed_dict={xs:x_data,ys:y_data})
        lines = ax.plot(x_data,prediction_value,'r-',lw=5)
        plot.pause(0.1)
```

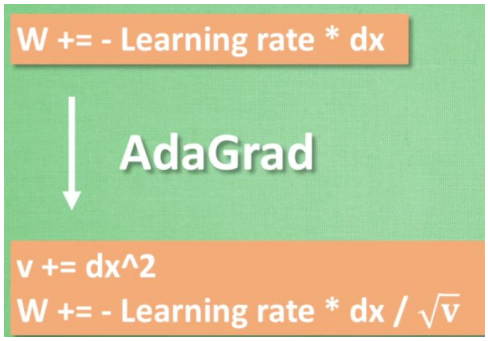
怎样加速神经网络训练?

每次训练都使用批量数据，大多都在神经网络参数的改动上下功夫。

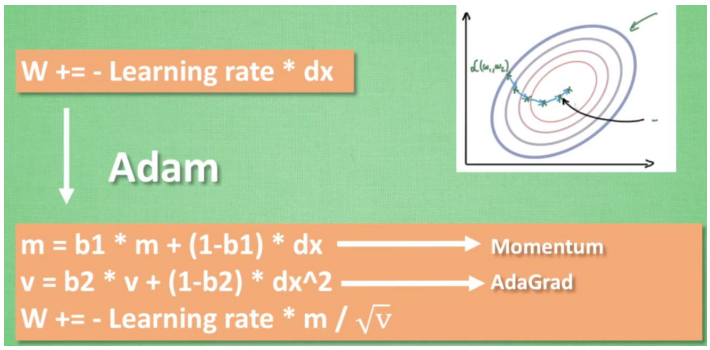
如：Moment



或者在学习方法上下功夫：



或者结合：



对 optimizer(优化器)的学习：

最常用的是 GradientDescentOptimizer（SGD），根据输入进行相应的变化。要花更多的时间来达到最终的目的。

接着是 MomentumOptimizer，考虑了不仅是面前一步的结果，还在考虑之前的结果，这样能够更快的达到目的，达到优化的效果。

- Gating Gradients
- Slots
- class tf.train.GradientDescentOptimizer
- class tf.train.AdadeltaOptimizer
- class tf.train.AdagradOptimizer
- class tf.train.AdagradDAOptimizer
- class tf.train.MomentumOptimizer
- class tf.train.AdamOptimizer
- class tf.train.FtrlOptimizer
- class tf.train.RMSPropOptimizer

可视化的更好方法：

```
import tensorflow as tf
import numpy as np
import matplotlib
matplotlib.use('TkAgg')
import matplotlib
import matplotlib.pyplot as plt
from pylab import *
def add_layer(inputs, in_size, out_size, activation_function=None):
    # add one more layer and return the output of this layer
    with tf.name_scope('layer'):
        with tf.name_scope('weights'):
            Weights = tf.Variable(
                                    tf.random_normal([in_size, out_size]),
                                    name='W')
        with tf.name_scope('biases'):
            biases = tf.Variable(
                                   tf.zeros([1, out_size]) + 0.1,
                                   name='b')
        with tf.name_scope('Wx_plus_b'):
            Wx_plus_b = tf.add(
                                  tf.matmul(inputs, Weights),
                                  biases)
        if activation_function is None:
            outputs = Wx_plus_b
        else:
            outputs = activation_function(Wx_plus_b, )
        return outputs
#Only one neuron in the input and output layers
x_data = np.linspace(-1,1,300, dtype=np.float32)[:, np.newaxis]
noise = np.random.normal(0, 0.05, x_data.shape).astype(np.float32)
y_data = np.square(x_data) - 0.5 + noise
```

```
    with tf.name_scope('inputs'):
        xs = tf.placeholder(tf.float32,[None,1],name = 'x_input')
        ys = tf.placeholder(tf.float32,[None,1],name = 'y_input')

    #The following set the hidden layer has 10 nodes.
    l1 = add_layer(xs,1,10,activation_function = tf.nn.relu)
    prediction = add_layer(l1,10,1,activation_function = None)
    with tf.name_scope('loss'):
        loss = tf.reduce_mean(tf.reduce_sum(tf.square(ys - prediction),reduction_indices=[1]))
    with tf.name_scope('train'):
        train_step = tf.train.GradientDescentOptimizer(0.1).minimize(loss)
    sess = tf.Session()
    wirter = tf.summary.FileWriter("logs/",sess.graph)
    sess.run(tf.global_variables_initializer())
```

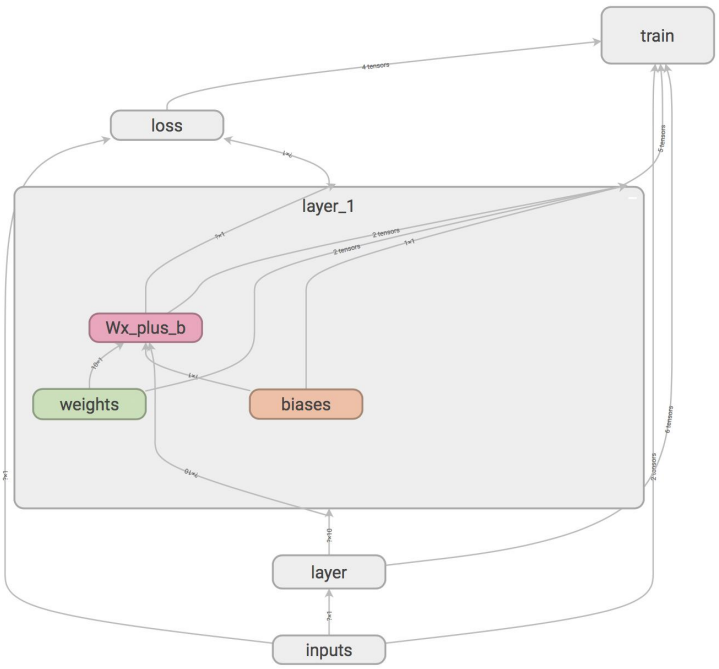
生成的文件，需要调用 terminal，切换 python 所在文件夹后后，运行其。

```
    tensorboard --logdir logs
```

输出结果为：

```
    (python27) zhanglipengdeMacBook-Pro:Desktop zhanglipeng$ tensorboard --logdir logs
    TensorBoard 1.12.2 at http://zhanglipengdeMacBook-Pro.local:6006 (Press CTRL+C to quit)
```

（真的很强～）



接着我们学习怎么显示更多的东西：

```
import tensorflow as tf
import numpy as np
import matplotlib
matplotlib.use('TkAgg')
import matplotlib
import matplotlib.pyplot as plt
from pylab import *

def add_layer(inputs, in_size, out_size,n_layer, activation_function=None):
    # add one more layer and return the output of this layer
    layer_name = 'layer%s' % n_layer
    with tf.name_scope('layer_name'):
        with tf.name_scope('weights'):
            Weights = tf.Variable(
                                    tf.random_normal([in_size, out_size]),
                                    name='W')
            tf.summary.histogram(layer_name+'/weights',Weights)
        with tf.name_scope('biases'):
            biases = tf.Variable(
                                   tf.zeros([1, out_size]) + 0.1,
                                   name='b')
            tf.summary.histogram(layer_name + '/biases', biases) # Tensorflow >= 0.12

        with tf.name_scope('Wx_plus_b'):
            Wx_plus_b = tf.add(
                                  tf.matmul(inputs, Weights),
                                  biases)
        if activation_function is None:
            outputs = Wx_plus_b
        else:
            outputs = activation_function(Wx_plus_b, )
        tf.summary.histogram(layer_name + '/outputs', outputs) # Tensorflow >= 0.12
        return outputs

#Only one neuron in the input and output layers
x_data = np.linspace(-1,1,300, dtype=np.float32)[:, np.newaxis]
noise = np.random.normal(0, 0.05, x_data.shape).astype(np.float32)
y_data = np.square(x_data) - 0.5 + noise

with tf.name_scope('inputs'):
```

```
        xs = tf.placeholder(tf.float32,[None,1],name = 'x_input')
        ys = tf.placeholder(tf.float32,[None,1],name = 'y_input')

    #The following set the hidden layer has 10 nodes.
    l1 = add_layer(xs,1,10,n_layer = 1,activation_function = tf.nn.relu)
    prediction = add_layer(l1,10,1,n_layer = 2,activation_function = None)

    with tf.name_scope('loss'):
        loss = tf.reduce_mean(tf.reduce_sum(tf.square(ys - prediction),reduction_indices=[1]))
        tf.summary.scalar('loss', loss) # tensorflow >= 0.12

    with tf.name_scope('train'):
        train_step = tf.train.GradientDescentOptimizer(0.1).minimize(loss)

    sess= tf.Session()

    # merged= tf.merge_all_summaries()      # tensorflow < 0.12
    merged = tf.summary.merge_all() # tensorflow >= 0.12
    writer = tf.summary.FileWriter("logs/", sess.graph) # tensorflow >=0.12
    sess.run(tf.global_variables_initializer())
    for i in range(1000):
        sess.run(train_step, feed_dict={xs:x_data, ys:y_data})
        if i%50 == 0:
            rs = sess.run(merged,feed_dict={xs:x_data,ys:y_data})
            writer.add_summary(rs, i)
```

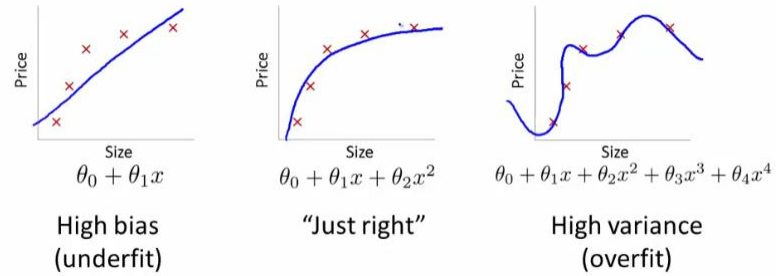
clssification 的学习：

此次神经网络学习的目的，不同于以往 回归，这边主要是解决分类的问题。使用的是 Mnist 数据集，手写数字的识别。

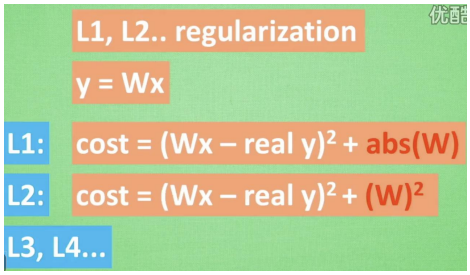
```
        import tensorflow as tf
        from tensorflow.examples.tutorials.mnist import input_data

        mnist = input_data.read_data_sets('MNIST_data', one_hot=True)

        def add_layer(inputs,in_size,out_size,activation_function=None):
            Weights = tf.Variable(tf.random_normal([in_size,out_size]))
            biases = tf.Variable(tf.zeros([1,out_size])+0.1)
            Wx_plus_b = tf.matmul(inputs,Weights)+biases
            if activation_function is None:
                outputs = Wx_plus_b
            else:
                outputs = activation_function(Wx_plus_b)
            return outputs
```

```
def compute_accuracy(v_xs,v_ys):
    global prediction
    y_pre = sess.run(prediction,feed_dict={xs:v_xs})
    correct_prediction = tf_equal(tf.cast(correct_prediction,tf.float32))
    result = sess.run(accuracy,feed_dict={xs:v_xs,ys:v_ys})
    return result


xs = tf.placeholder(tf.float32,[None,784])
ys = tf.placeholder(tf.float32,[None,10])

prediction = add_layer(xs,784,10,activation_function_function = tf.nn.softmax)

cross_entropy = tf.reduce_mean(-tf.reduce_sum(ys * tf.log(prediction),
                                              reduction_indices=[1]))       #
loss

train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)
sess = tf.Session()
sess.run(tf.initialize_all_variables())

for i in range(1000):
    batch_xs,batch_ys=mnist.train.next_batch(100)
    if i % 50 == 0:
        print compute_accuracy(mnist.test.images,mnist.test.labels)
```

过拟合是机器学习的自负现象，过于追求小误差，在更多数据面前反而表现出了不好的效果。



第一个方法是增加数据量，第二个方法是进行正规化

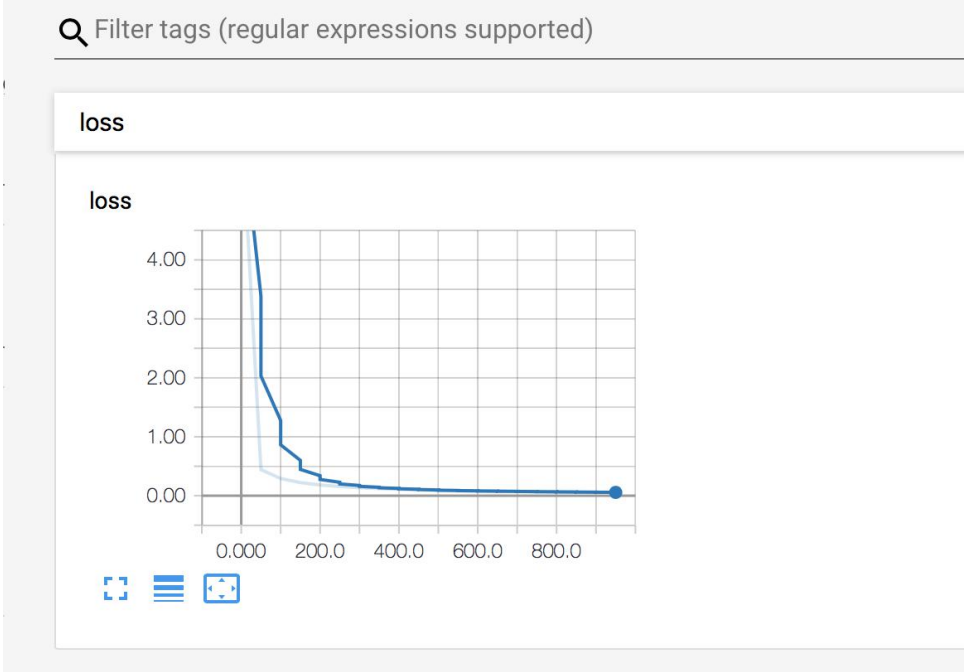


第三个方法就是 dropout，随机 drop 来避免过于依赖某一个点。

```
import tensorflow as tf
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelBinarizer

#load_data
digits = load_digits()
X = digits.data
y = digits.target
y = LabelBinarizer().fit_transform(y)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.3)

def add_layer(inputs,in_size,out_size,layer_name,activation_function=None):
    Weights = tf.Variable(tf.random_normal([in_size,out_size]))
    biases = tf.Variable(tf.zeros([1,out_size])+0.1)
    Wx_plus_b = tf.matmul(inputs,Weights)+biases
    if activation_function is None:
        outputs = Wx_plus_b
    else:
        outputs = activation_function(Wx_plus_b)
    tf.summary.histogram(layer_name + '/outputs',outputs)
    return outputs

def compute_accuracy(v_xs,v_ys):
    global prediction
    y_pre = sess.run(prediction,feed_dict={xs:v_xs})
    correct_prediction = tf_equal(tf.cast(correct_prediction,tf.float32))
    result = sess.run(accuracy,feed_dict={xs:v_xs,ys:v_ys})
    return result

xs = tf.placeholder(tf.float32,[None,64])
ys = tf.placeholder(tf.float32,[None,10])

# add output layer
l1 = add_layer(xs, 64, 50, 'l1', activation_function=tf.nn.tanh)
prediction = add_layer(l1, 50, 10, 'l2', activation_function=tf.nn.softmax)

cross_entropy = tf.reduce_mean(-tf.reduce_sum(ys *
tf.log(prediction),reduction_indices=[1])) # loss
tf.summary.scalar('loss',cross_entropy)
train_step = tf.train.GradientDescentOptimizer(0.6).minimize(cross_entropy)

sess = tf.Session()
```

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

train_writer = tf.summary.FileWriter('logs/train',sess.graph)
test_writer = tf.summary.FileWriter('logs/test',sess.graph)

sess.run(tf.global_variables_initializer())

for i in range(1000):
    sess.run(train_step,feed_dict={xs:X_train,ys:y_train})
    if i % 50 == 0:
        train_result = sess.run(merged, feed_dict={xs: X_train, ys: y_train})
        test_result = sess.run(merged, feed_dict={xs: X_train, ys: y_train})
        train_writer.add_summary(train_result,i)
        train_writer.add_summary(test_result,i)
```

输出图形的查看：



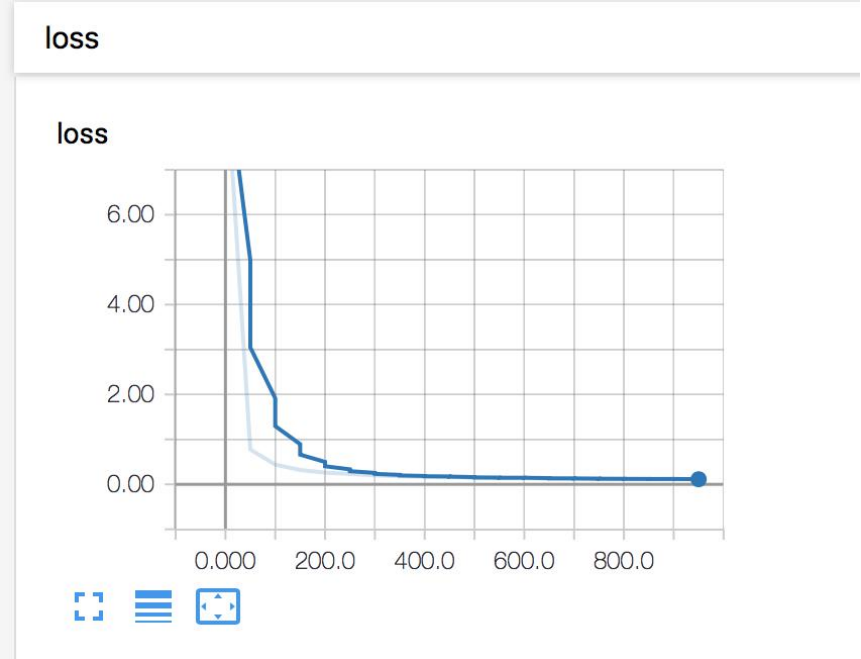
可以看到，出现了过拟合现象。

改正代码：

```
import tensorflow as tf
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelBinarizer

#load_data
digits = load_digits()
X = digits.data
y = digits.target
y = LabelBinarizer().fit_transform(y)
```

```
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.3)

    def
add_layer(inputs,in_size,out_size,layer_name,activation_function=None):
        Weights = tf.Variable(tf.random_normal([in_size,out_size]))
        biases = tf.Variable(tf.zeros([1,out_size])+0.1)
        Wx_plus_b = tf.matmul(inputs,Weights)+biases
        Wx_plus_b = tf.nn.dropout(Wx_plus_b,keep_prob)
        if activation_function is None:
            outputs = Wx_plus_b
        else:
            outputs = activation_function(Wx_plus_b)
        tf.summary.histogram(layer_name + '/outputs',outputs)
        return outputs

    def compute_accuracy(v_xs,v_ys):
        global prediction
        y_pre = sess.run(prediction,feed_dict={xs:v_xs})
        correct_prediction = tf_equal(tf.cast(correct_prediction,tf.float32))
        result = sess.run(accuracy,feed_dict={xs:v_xs,ys:v_ys})
        return result

    keep_prob = tf.placeholder(tf.float32)
    xs = tf.placeholder(tf.float32,[None,64])
    ys = tf.placeholder(tf.float32,[None,10])

    # add output layer
    l1 = add_layer(xs, 64, 50, 'l1', activation_function=tf.nn.tanh)
    prediction          =          add_layer(l1,          50,          10,          'l2',
activation_function=tf.nn.softmax)

    cross_entropy          =          tf.reduce_mean(-tf.reduce_sum(ys          *
tf.log(prediction),reduction_indices=[1])) # loss
    tf.summary.scalar('loss',cross_entropy)
    train_step                                                                    =
tf.train.GradientDescentOptimizer(0.6).minimize(cross_entropy)

    sess = tf.Session()
    merged =    tf.summary.merge_all()

    train_writer = tf.summary.FileWriter('logs/train',sess.graph)
    test_writer = tf.summary.FileWriter('logs/test',sess.graph)

    sess.run(tf.global_variables_initializer())
```

```
for i in range(1000):
    sess.run(train_step,feed_dict={xs:X_train,ys:y_train,keep_prob:0.5})
     if i % 50 == 0:
          train_result = sess.run(merged, feed_dict={xs: X_train, ys:
y_train,keep_prob:1})
          test_result = sess.run(merged, feed_dict={xs: X_train, ys:
y_train,keep_prob:1})
          train_writer.add_summary(train_result,i)
          train_writer.add_summary(test_result,i)
```



好了很多了哈哈。

卷积神经网络 CNN

```
from __future__ import division, print_function, absolute_import
import os
os.environ["KMP_DUPLICATE_LIB_OK"]="TRUE"
import tflearn
import tflearn.data_utils as du

# Data loading and preprocessing
import tflearn.datasets.mnist as mnist
X, Y, testX, testY = mnist.load_data(one_hot=True)
X = X.reshape([-1, 28, 28, 1])
testX = testX.reshape([-1, 28, 28, 1])
X, mean = du.featurewise_zero_center(X)
testX = du.featurewise_zero_center(testX, mean)

# Building Residual Network
```

```
net = tflearn.input_data(shape=[None, 28, 28, 1])
net = tflearn.conv_2d(net, 64, 3, activation='relu', bias=False)
# Residual blocks
net = tflearn.residual_bottleneck(net, 3, 16, 64)
net = tflearn.residual_bottleneck(net, 1, 32, 128, downsample=True)
net = tflearn.residual_bottleneck(net, 2, 32, 128)
net = tflearn.residual_bottleneck(net, 1, 64, 256, downsample=True)
net = tflearn.residual_bottleneck(net, 2, 64, 256)
net = tflearn.batch_normalization(net)
net = tflearn.activation(net, 'relu')
net = tflearn.global_avg_pool(net)
# Regression
net = tflearn.fully_connected(net, 10, activation='softmax')
net = tflearn.regression(net, optimizer='momentum',
                                    loss='categorical_crossentropy',
                                    learning_rate=0.1)
# Training
model = tflearn.DNN(net, checkpoint_path='model_resnet_mnist',
                            max_checkpoints=10, tensorboard_verbose=0)
model.fit(X, Y, n_epoch=100, validation_set=(testX, testY),
                show_metric=True, batch_size=256, run_id='resnet_mnist')
```

Saver 保存神经网络

```
import tensorflow as tf
import numpy as np

#remember to define the same dtype and shape when restore
'''
W = tf.Variable([[1,2,3],[3,4,5]],dtype=tf.float32,name='weights')
b = tf.Variable([[1,2,3]],dtype=tf.float32,name='biases')

init = tf.global_variables_initializer()

saver = tf.train.Saver()

with tf.Session() as sess:
    sess.run(init)
    save_path = saver.save(sess,"my_net/save_net.ckpt")
    print "Save to path:",save_path
'''
#restore,redefine the same shape and same type for your variables
W = tf.Variable(np.arange(6).reshape((2,3)),dtype=tf.float32,name='weights')
b = tf.Variable(np.arange(3).reshape((1,3)),dtype=tf.float32,name='biases')
```

```
#not need to init step
saver = tf.train.Saver()
with tf.Session() as sess:
    saver.restore(sess,"my_net/save_net.ckpt")
    print "weights",sess.run(W)
    print "biases",sess.run(b)
```

循环神经网络 RNN

首先要明确 LSTM，可以用来防止 RNN 梯度爆炸。from __future__ import division, print_function, absolute_import

```
import os
os.environ["KMP_DUPLICATE_LIB_OK"]="TRUE"
import tflearn
import tflearn.data_utils as du
import tflearn.datasets.mnist as input_data
import tensorflow as tf

mnist = input_data.read_data_sets("MNIST_data",one_hot=True)

lr = 0.001
training_iters = 100000
batch_size = 128

n_inputs = 28
n_steps = 28
n_hidden_units = 128
n_classes = 10

x = tf.placeholder(tf.float32,[None,n_steps,n_inputs])
y = tf.placeholder(tf.float32,[None,n_classes])

weights = {
    # shape (28, 128)
    'in': tf.Variable(tf.random_normal([n_inputs, n_hidden_units])),
    # shape (128, 10)
    'out': tf.Variable(tf.random_normal([n_hidden_units, n_classes]))
}
biases = {
    # shape (128, )
    'in': tf.Variable(tf.constant(0.1, shape=[n_hidden_units, ])),
    # shape (10, )
    'out': tf.Variable(tf.constant(0.1, shape=[n_classes, ]))
}
```

```
    def RNN(X,weights,biases):
        #X(128batch,28steps,28inputs)
        #-->(128*28,28inputs)
        X = tf.reshape(X,[-1,n_inputs])
        #-->(128batch*28steps,128hiddden)
        X_in = tf.matmul(X,weights['in'])+biases['in']
        #-->(128batch,28steps,28inputs)
        X_in = tf.reshape(X_in,[-1,n_steps,n_hidden_units])

        lstm_cell = tf.nn.rnn_cell.BasicLSTMCell(n_hidden_units,forget_bias=1.0,state_is_tuple = True)
        init_state = lstm_cell.zero_state(batch_size,dtype=tf.float32)

        outputs, states = tf.nn.dynamic_rnn(lstm_cell, X_in, initial_state=init_state, time_major=False)

        results = tf.matmul(states[1],weights['out'])+biases['out']
        return results

    pred = RNN(x, weights, biases)
    cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=pred,logits=y))
    train_op = tf.train.AdamOptimizer(lr).minimize(cost)

    correct_pred = tf.equal(tf.argmax(pred, 1), tf.argmax(y, 1))
    accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))


    init = tf.global_variables_initializer()

    with tf.Session() as sess:
        sess.run(init)
        step = 0
        while step * batch_size < training_iters:
            batch_xs, batch_ys = mnist.train.next_batch(batch_size)
            batch_xs = batch_xs.reshape([batch_size, n_steps, n_inputs])
            sess.run([train_op], feed_dict={ x: batch_xs,y: batch_ys,})
            if step % 20 == 0:
                print(sess.run(accuracy, feed_dict={x: batch_xs,y: batch_ys,}))
            step += 1
```

迁移学习

利用已有的网络，保持已有的神经层理解能力，在一个新的要求中使用新的输出。迁移的模型是否有价值保留也有待商榷。使用已经有训练过的模型，改头换面，使其完成新的任务：