# 第二周 Tensorflow的基本概念

## 1、基本概念

* 使用图（graphs）来表示计算任务
* 在被称之为会话（Session）的上下文（context）中执行图
* 使用tensor表示数据
* 通过变量Variable维护状态
* 使用feed和fetch可以为任意的操作赋值或者从其中获取数据

Tensorflow是一个编程系统，使用图（graphs）来表示计算任务，图（graphs）中的节点称之为operation，（add，mat之类），一个op获得0个或者多个tensor，执行计算，产生0个或多个tensor，tensor看作是一个n维的数组或列表。图必须在会话里被启动。



## 2、基本代码（变量、常量、会话、op节点、图）

# 声明变量

x = tf.Variable([1,2])

# 声明常量

a = tf.constant([3,3])

# 增加一个减法op

sub = tf.subtract(x,a)

# 增加一个加法op

add = tf.add(a,sub)

# 全局变量初始化

init = tf.global\_variables\_initializer()

# 在图内运行代码

with tf.Session() as sess:

sess.run(init)

print(x.value)

print(sess.run(sub))

print(sess.run(add))

# 赋值op

#update = tf.assign(state,new\_value)

## 3、Fetch and Feed

Fetch可以在一个会话中同时执行多个op,也就是说可以同事进行多个节点的计算

Feed 可以每次为网络传入不同的数据

代码示例：

# Fetch

input1 = tf.constant(3.0)

input2 = tf.constant(2.0)

input3 = tf.constant(5.0)

add = tf.add(input2,input3)

mul = tf.multiply(input1,add)

with tf.Session() as sess:

result = sess.run([mul, add])

print((result))

输出了两个数字

# Feed

# 创建占位符

input1 = tf.placeholder(tf.float32)

input2 = tf.placeholder(tf.float32)

output = tf.multiply(input1, input2)

with tf.Session() as sess:

# feed数据以字典形式传入

print(sess.run(output, feed\_dict={input1:[2.0],input2:[6.0]}))

有了占位符和feed，网络可以每次传入不同的数据

## 4、简单示例

import tensorflow as tf

import numpy as np

# 生成随机点

x\_data = np.random.rand(100)

y\_data = x\_data\*0.1 + 0.2

# 构建线性模型

b = tf.Variable(0.)

k = tf.Variable(0.)

y = k\*x\_data + b

# 定义损失函数

loss = tf.reduce\_mean(tf.square(y\_data-y))

# 定义一个梯度下降法进行训练的优化器,学习率是0.2

optimizer = tf.train.GradientDescentOptimizer(0.2)

# 最小化代价函数

train = optimizer.minimize(loss)

# 变量初始化

init = tf.global\_variables\_initializer()

with tf.Session() as sess:

sess.run(init)

for step in range(201):

sess.run(train)

if step%20 == 0:

print(step+1, sess.run([k,b]))

# 第三周 Tensorflow非线性回归以及分类的简单使用，softmax介绍

1、非线性回归简单示例

import tensorflow as tf

import numpy as np

import matplotlib.pyplot as plt

# 生成200个随机点，并改变其形状为200\*1

x\_data = np.linspace(-0.5, 0.5, 200)[:,np.newaxis]

noise = np.random.normal(0,0.02,x\_data.shape)

y\_data = np.square(x\_data) + noise

#查看一下数据形状

print(x\_data.shape)

type(x\_data)

print(noise.shape)

# 定义两个placeholder

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

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

# 定义中间层

Weights\_L1 = tf.Variable(tf.random\_normal([1,10]))

bias\_L1 = tf.Variable(tf.zeros([1,10]))

Wx\_plus\_b\_L1 = tf.matmul(x, Weights\_L1) + bias\_L1

# 激活函数

L1 = tf.nn.tanh(Wx\_plus\_b\_L1)

# 定义输出层

Weights\_L2 = tf.Variable(tf.random\_normal([10,1]))

bias\_L2 = tf.Variable(tf.zeros([1,1]))

Wx\_plus\_b\_L2 = tf.matmul(L1,Weights\_L2) + bias\_L2

prediction = tf.nn.tanh(Wx\_plus\_b\_L2)

# 二次代价函数（损失函数）

loss = tf.reduce\_mean(tf.square(y-prediction))

# 梯度下降法

train\_step = tf.train.GradientDescentOptimizer(0.1).minimize(loss)

with tf.Session() as sess:

# 变量的初始化

sess.run(tf.global\_variables\_initializer())

for \_ in range(2000):

sess.run(train\_step, feed\_dict={x:x\_data, y:y\_data})

# 获得预测值

prediction\_value = sess.run(prediction,feed\_dict={x:x\_data})

plt.figure()

plt.scatter(x\_data, y\_data)

plt.plot(x\_data, prediction\_value,'r-', lw=5)

plt.show()

# 第四周 交叉熵(cross-entropy)，过拟合，dropout以及Tensorflow中各种优化器的介绍

## 1、代价函数的选择（关于交叉熵代价函数和对数释然代价函数

如果输出神经元是线性的，二次代价函数是合适的选择，

如果输出神经元是S型函数，适用交叉熵代价函数。（这里指的是输出层被sigmoid函数激活的情况）

如果输出层被sigmoid激活，可采用交叉熵代价函数，如果使用softmax作为网络最后一层，此时常用对数释然代价函数。

对数似然代价函数与softmax的组合，跟交叉熵与sigmoid函数的组合非常相似。在二分类问题中，对数释然代价函数简化为交叉熵代价函数的形势。

在Tensorflow中

Tf.sigmoid\_cross\_entropy\_with\_logits()表示跟sigmoid搭配使用的交叉熵

Tf.softmax\_corss\_entropy\_with\_logits()表示跟softmax搭配使用的交叉熵。

## 2、过拟合

解决过拟合的操作

增加数据集、

正则化方法、减小loss，从而减小学习率

Dropout、随机丢弃某些神经节点，使网络不能过度依赖某些节点

## 3、Optimizer

各种优化器对比

标准梯度下降法：

计算所有样本汇总误差，根据总误差来更新权值；

缺点：太慢；

随机梯度下降法：

随机抽取一个样本来计算误差，然后更新权值；

缺点：对噪声过于敏感

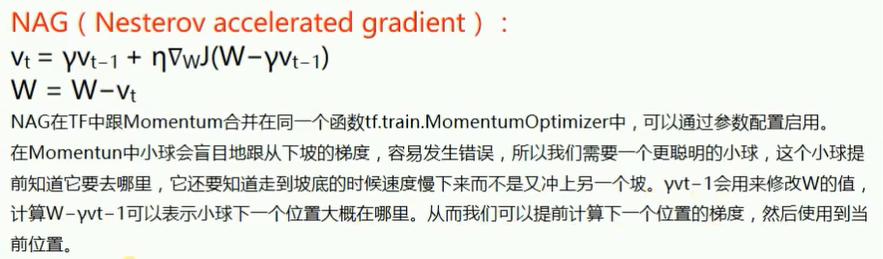
批量梯度下降法：

算是一种折中方案，从总样本中选取一个批次，计算该批次数据误差，来更新权值；

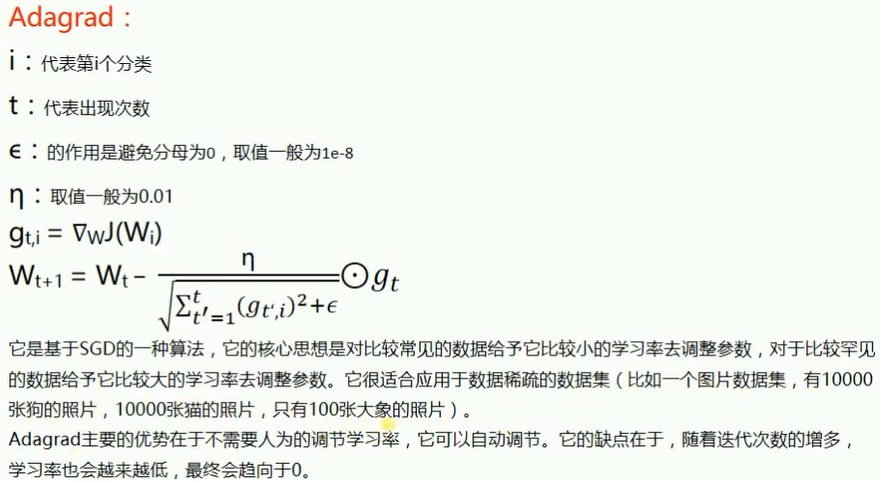
### SGD Momentum



### NAG

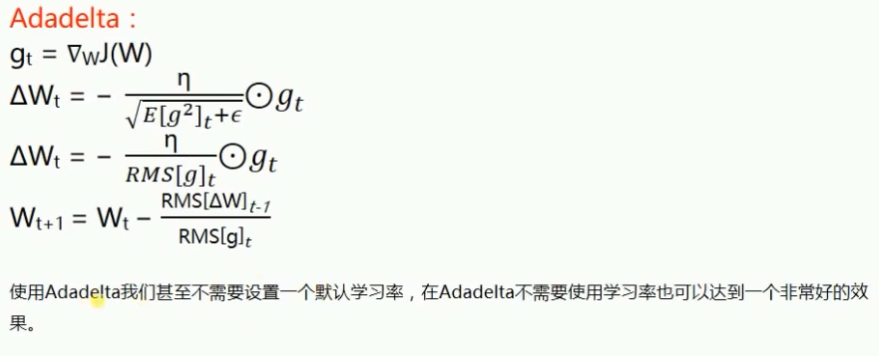


### Adagrad

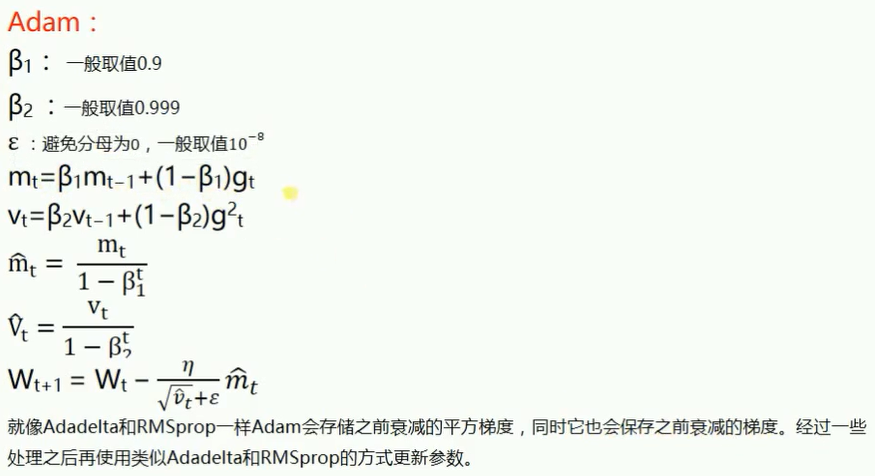


### RMSprop

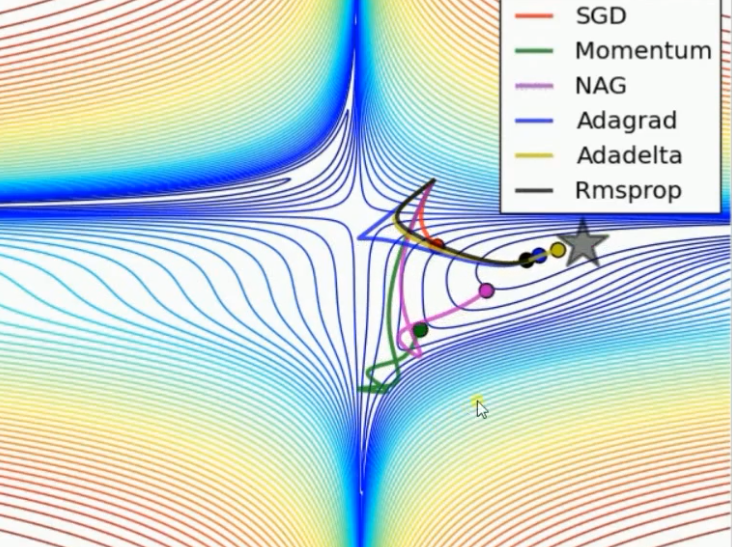
### Adadelta



### Adam



### 效果比较1



效果图：

排名：

Adadelta

Adagrade

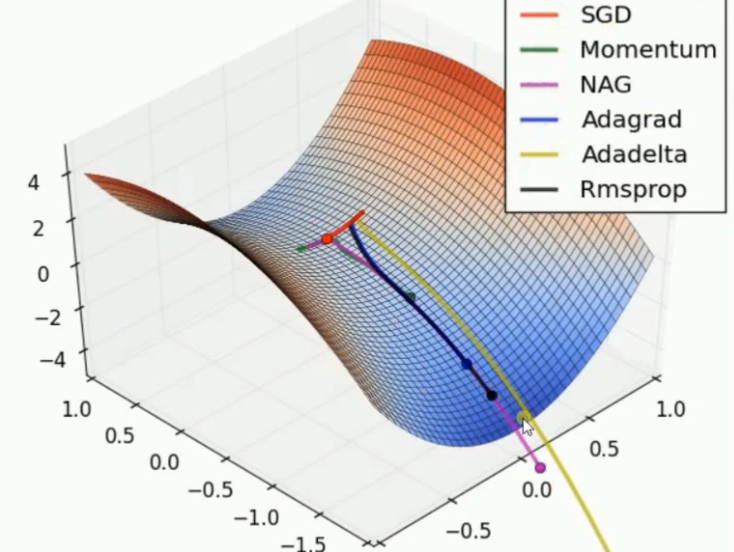
Rmsprop

NAG

Momentum

SGD

### 效果比较：鞍点问题



排名：

Adadelta以很快的速度冲下来

NAG在最初的迷茫之后快速走下鞍点，速度很快

Momentum也在迷茫之后走下鞍点，但是没有NAG

Rmsprop没有迷茫，但是下降速度有点慢

Adagrad 也没有迷茫，但是速度更慢

SGD，直接在鞍点下不来了

### 效果比较：总结

优化器各有优缺点

别的优化器收敛速度会比较快，但是SGD最后的结果一般来说很好