1.基础

constant 英 ['kɒnst(ə)nt]、美 ['kɑnstənt]adj 常数；恒量

initializer [i'niʃəlaizə] 初始化器(global\_variables\_initializer、random\_uniform\_initializer)

global 英 ['gləʊb(ə)l]、美 ['ɡlobl] 总体(global\_step)

reuse 英 [riː'juːz]、美 [,ri'jʊs] 再次使用；重复使用(reuse=True)

scope 英 [skəʊp]、美 [skop] 范围(variable\_scope())

scalar 英 ['skeɪlə]、美 [skelɚ] 标量

vector 英 ['vektə]、美 ['vɛktɚ] 向量

interactive 英 [ɪntər'æktɪv]、美 [,ɪntɚ'æktɪv] 交互式

(InteractiveSession)

proto 英 [prəʊtə]、美 ['proto] 典型(ConfigProto)

matmul [mai:mu] 矩阵相乘

allow 英 [ə'laʊ]、美 [ə'laʊ] 允许(allow\_soft\_placement

、log\_device\_placement)

sigmoid 英 ['sɪgmɒɪd]、美 ['sɪgmɔɪd] S状弯曲

relu [ri:l] 线性整流函数

tanh [tænʃ] [tænʃ] abbr. 双曲正切

cross 英 [krɒs]、美 [krɔs] 交叉

entropy 英 ['entrəpɪ]、美 ['ɛntrəpi] [热] 熵

Adam 英 [ˈædəm]、美 [ˈædəm] 亚当(AdamOptimizer)

optimizer 英 ['ɒptɪmaɪzə] 优化

minimize 英 [ˈmɪnɪˌmaɪz]、美 ['mɪnɪmaɪz] 最小化

gradient 英 ['greɪdɪənt]、美 ['ɡredɪənt] [数][物] 梯度

descent 英 [dɪ'sent]、美 [dɪ'sɛnt] 下降(GradientDescentOptimizer)

train 英 [treɪn]、美 [tren] 训练

normal 英 ['nɔːm(ə)l]、美 ['nɔrml] 正常的；正规的；标准的(正态分布)

truncated 英 [trʌŋ'keɪtɪd]、美 ['trʌŋketɪd] 缩短了的；切去顶端的

uniform 英 ['juːnɪfɔːm]、美 [junəˌfɔrm] 均衡的(均匀分布)

gamma 英 ['gæmə]、美 ['gæmə] 丙等、γ射线

2.深层

biases [baie:si:s] 偏置**，**偏差，偏见

exponential 英 [,ekspə'nenʃ(ə)l]、美 ['ɛkspə'nɛnʃəl] 指数的、 指数

decay 英 [dɪ'keɪ]、美 [dɪ'ke] 衰减(exponential\_decay指数衰减法)

staircase 英 ['steəkeɪs]、美 ['stɛrkes] 楼梯(True或False)

rate 英 [reɪt]、美 [ret] 比率，率；速度(decay\_rate)

reduce 英 [rɪ'djuːs]、美 [rɪ'dʊs] 减少；降低

mean 英 [miːn]、美 [min] 用意（reduce\_mean）

clip 英 [klɪp]、美 [klɪp] 修剪；剪掉(clip\_by\_value)

feed 英 [fiːd]、美 [fid] 喂养；供给

dict [dai:c:t] 词典，字典(feed\_dict)

step 英 [step]、美 [stɛp] 步(STEPS)

soft 英 [sɒft]、美 [sɔft] 软的；柔性()

max [mæks] [mæks] 最大量；最大数(Softmax)

logits [lao:zhen:t] 分对数(sparse\_softmax\_cross\_entropy\_with\_logits)

sparse 英 [spɑːs]、美 [spɑrs] 稀疏的；稀少的

square 英 [skweə]、美 [skwɛr] 平方的(tf.square回归问题均方误差)

moving 英 ['muːvɪŋ]、美 ['muvɪŋ] 移动(ExponentialMovingAverage)

average 英 ['æv(ə)rɪdʒ]、美 ['ævərɪdʒ] 平均

shadow 英 ['ʃædəʊ]、美 ['ʃædo] 阴影；影子(shadow\_variable)

apply 英 [ə'plaɪ]、美 [ə'plaɪ] 应用

assign 英 [ə'saɪn]、美 [ə'saɪn] 赋值

regularization [,raiɡjʊlərɪ'zeʃən] [,rɛɡjʊlərɪ'zeʃən] 规则化

regularizer [raiɡjʊlairɪ'ze] 正则(tf.contrib.layers.l2\_regularizer)

overfitting [o:ve:fi:ting] 过度拟合

program 英 ['prəʊɡræm]、美 ['proɡræm] 程序；计划

3.mnist

examples [ɪg'zæmpl] 例子；列证(mnist.train.num\_examples数据个数)

control 英 [kən'trəʊl]、美 [kən'trol] 控制；管理

dependencies [dɪ'pɛndənsi] [dɪ'pɛndənsi] 依赖性(control\_dependencies)

validation 英 [,vælɪ'deɪʃən]、美 [,vælə'deʃən] 确认

validate 英 ['vælɪdeɪt]、美 ['vælɪdet] 证实，验证

accuracy 英 ['ækjʊrəsɪ]、美 ['ækjərəsi] 准确性

correct 英 [kə'rekt]、美 [kə'rɛkt] 正确的(correct\_prediction)

prediction 英 [prɪ'dɪkʃ(ə)n]、美 [prɪ'dɪkʃən] 预言

hot 英 [hɒt]、美 [hɑt] 紧迫的、增加(one\_hot)

meta 英 ['metə]、美 ['mɛtə] 变化、变换(import\_meta\_graph)

restore 英 [rɪ'stɔː]、美 [rɪ'stɔr] 恢复、还原(加载保存的模型)

convert 英 [kən'vɜːt]、美 [kən'vɝt] 转换(convert\_variables\_to\_constants)

serialize 英 [ˈsɪərɪəlaɪz]、美 ['sɪrɪəlaɪz] 连载(SerializeToString)

except 英 [ɪkˈsept]、美 [ɪk'sɛpt] 把…除外、除了

checkpoint 英 ['tʃekpɒɪnt]、美 ['tʃɛkpɔɪnt] 关卡(get\_checkpoint\_state)

evaluate 英 [ɪ'væljʊeɪt]、美 [ɪ'væljʊ'et] 评价；估价

argmax (tf.argmax(y\_, 1))

4.CNN

convolutional [kan:ve:lu:shen:nl] 卷积的(Convolutional Neural Network，CNN)

neural 英 ['njʊər(ə)l]、美 ['nʊrəl] 神经的；神经系统的

network 英 ['netwɜːk]、美 ['nɛtwɝk] 网络

inception 英 [ɪn'sepʃ(ə)n]、美 [ɪn'sɛpʃən] 起初(Inception v3)

Alex 英 ['ælɪks]( AlexNet)

stride 英 [straɪd]、美 [straɪd] 步长

filter 英 ['fɪltə]、美 ['fɪltɚ] 滤波器；过滤器、过滤

pooling [pu:ling] 合并；池化

padding 英 ['pædɪŋ]、美 ['pædɪŋ] 填补

same 英 [seɪm]、美 [sem] 相同的

valid 英 ['vælɪd]、美 ['vælɪd] 有效的

concat [ken:kai:te] 合并多个数组

current 英 ['kʌr(ə)nt]、美 ['kɜrənt] 现在的

depth 英 [depθ]、美 [dɛpθ] [海洋] 深度

migration 英 [maɪ'greɪʃ(ə)n]、美 [maɪ'ɡreʃən] 迁移

glob 英 [glɒb]、美 [ɡlɑb] 一滴；水珠；一团

shuffle 英 ['ʃʌf(ə)l]、美 ['ʃʌfl] 把……变换位置；调动

5.RNN

recurrent 英 [rɪ'kʌr(ə)nt]、美 [rɪ'kɝənt] 复发的(recurrent neuralnetwork，RNN)

term 英 [tɜːm]、美 [tɝm] 期限(long short-term memory，LSTM)

memory 英 ['mem(ə)rɪ]、美 ['mɛməri] 记忆

bidirectional 英 [baɪdɪ'rekʃ(ə)n(ə)l] 双向的(bidirectional RNN))

dropout 英 ['drɒpaʊt]、美 ['drɑpaʊt] 中途退学；辍学学生

wrapper 英 ['ræpə]、美 ['ræpɚ] 包装(DropoutWrapper)

basic 英 ['beɪsɪk]、美 ['beɪsɪk] 基本、基础(BasicLSTMCell)

cell 英 [sel]、美 [sɛl] 细胞

state 英 [steɪt]、美 [stet] 声明；陈述(lstm.zero\_state初始化)

fully 英 ['fʊlɪ]、美 ['fʊli] ad 充分地；完全地(fully\_connected)

multi 英 ['mʌltɪ]、美 ['mʌlti] pref. 多(MultiRNNCell)

stacked 英 [stækt]、美 [stækt] 成堆的、 堆放(stacked\_lstm)

dynamic 英 [daɪ'næmɪk] 动态；动力(tf.nn.dynamic\_rnn)

6.preprocess

preprocess 英 [priː'prəʊses]、美 [pri'prɑsɛs] 预处理

features 英 [ˈfiːtʃəz]、美 [fitʃɚz] 特征

records 英 ['rekɔːdz]、美 ['rɛkɚdz] 记录(TFRecord)

protocol 英 ['prəʊtəkɒl]、美 ['protə'kɔl] 协议

producer 英 [prə'djuːsə] 生产者(创建一个队列string\_input\_producer)

queue 英 [kjuː]、美 [kju] 队列

parse 英 [pɑːz]、美 [pɑrs] 解析(parse\_single\_example解析一个样例)

single 英 ['sɪŋg(ə)l]、美 ['sɪŋɡl] 单一的

fixed 英 [fɪkst]、美 [fɪkst] 确定的(FixedLenFeature)

runner 英 ['rʌnə] 美 ['rʌnɚ] 跑步者(QueueRunner创建入队操作)

coordinator 英 [kəʊ'ɔ:dɪneɪtə] 协调者(Coordinator)

encode 英 [ɪn'kəʊd; en-]、美 [ɪn'kod] 编码(encode\_jpeg)

decode /diːˈkəʊd/ 解码(decode\_jpeg)

resize 英 [riː'saɪz]、美 [,ri'saɪz] 调整大小(resize\_images)

central 英 ['sentr(ə)l]、美 ['sɛntrəl] 中心的；主要的

crop 英 [krɒp]、美 [krɑp] 收割、收获(central\_crop比例调整图片大小)

adjust 英 [ə'dʒʌst]、美 [ə'dʒʌst] 调整

brightness 英 ['braɪtnɪs]、美 ['braɪtnɪs] 亮度(adjust\_brightness)

contrast 英 ['kɒntrɑːst]、美 ['kɑntræst] 对比(adjust\_contrast)

hue 英 [hjuː] 色彩；色度(adjust\_hue)

saturation 英 [sætʃə'reɪʃ(ə)n] 饱和(adjust\_saturation)

standardization [stændədaɪ'zeɪʃən] 标准化(per\_image\_standardization)

flip 英 [flɪp]、美 [flɪp] 筋斗、翻转(flip\_up\_down)

transpose 英 [træns'pəʊz] 颠倒顺序(transpose\_image对角线翻转)

convert 英 [kən'vɜːt]、美 [kən'vɝt] 转换类型(convert\_image\_dtype)

expand 英 [ɪk'spænd]、美 [ɪk'spænd] 扩张(expand\_dims增加一个维度)

draw 英 [drɔː]、美 [drɔ] 拉；拖(draw\_bounding\_boxes)

bounding [baʊnd] [baʊnd] 边界框

boxes 英 ['bɒksɪz]、美 ['bɔksɪz] 小木箱，盒子

sample 英 ['sɑːmp(ə)l] 抽样(sample\_distorted\_bounding\_box)

distorted 英 [dɪ'stɔːtɪd]、美 [dɪs'tɔrtɪd] 扭曲

slice 英 [slaɪs]、美 [slaɪs] 切下；切开(截取图像)

match 英 [mætʃ]、美 [mætʃ] 匹配(match\_filenames\_once模糊匹配)

enqueue 英 [ɪn'kjuː] 入队(enqueue\_many初始化队列)

dequeue [di'kju:] 出列(队列中的第一个元素出列tf.FIFOQueue.dequeue)

cast 英 [kɑːst]、美 [kæst] 计算(tf.cast转换类型)

shot 英 [ʃɒt]、美 [ʃɑt] 破旧的(make\_one\_shot\_iterator)

iterator 英 [ɪtə'reɪtə] 迭代器(make\_initializable\_iterator)

channels ['tʃænlz] ['tʃænlz] 通道

repeat 英 [rɪ'piːt]、美 [rɪ'pit] 重复；复制(dataset.repeat)

7. tensorboard

board 英 [bɔːd]、美 [bɔrd] 木板(tensorboard)

summary 英 ['sʌm(ə)rɪ]、美 ['sʌməri] 摘要，总结(tf.summary.FileWriter(path, tf.get\_default\_graph()))

histogram 英 ['hɪstəgræm] 柱状图(tf.summary.histogram)

audio 英 ['ɔːdɪəʊ]、美 ['ɔdɪo] 音频的

merge [mɜːdʒ]合并(tf.summary.merge\_all()writer.add\_summary(summary,i)

options [ˈɒpʃnz] 选择项(writer.add\_run\_metadata(run\_metadata, 'step%03d' % i))

trace [treɪs]追溯(tf.RunOptions(trace\_level=tf.RunOptions.FULL\_TRACE)

metadata 英 ['metədeɪtə]、美 ['mɛtədetə] 元数据(tf.RunMetadata())

projector 英 [prə'dʒektə]、美 [prə'dʒɛktɚ] 放映机(ProjectorConfig)

sprite 英 [spraɪt]、美 [spraɪt] 调皮鬼

embedding [ɪm'bedɪŋ] [ɪm'bedɪŋ] 植入

extend 英 [ɪk'stend; ek-]、美 [ɪk'stɛnd] 延伸

visualisation [vɪʒ(j)ʊəlʌɪˈzeɪʃ(ə)n] 可视化

visualize ['vɪʒuəlaɪz] 设想

8. highLevel

slim 英 [slɪm]、美 [slɪm] 苗条的；修长的；微小的；差的

fully 英 ['fʊlɪ]、美 ['fʊli] 充分地；完全地(全连接层)

flatten 英 ['flæt(ə)n]、美 ['flætn] 变平(4维变2维)

learn 英 [lɜːn]、美 [lɝn] 学习

regression 英 [rɪ'greʃ(ə)n] 回归；退化(用封装好的函数定义优化器等)

keras [kai:re:s]

sequential 英 [sɪ'kwenʃ(ə)l]、美 [sɪ'kwɛnʃl] 有顺序的(构建模型)

dense 英 [dens]、美 [dɛns] 浓厚的(构建全连接)

categorical 英 [kætɪ'gɒrɪk(ə)l] 绝对的(转换格式)

concatenate 英 [kən'kætɪneɪt] 连接(拼接输出)

fit 英 [fɪt]、美 [fɪt] 配合(训练模型)

estimator 英 ['estɪmeɪtə]、美 ['ɛstə,metɚ] [统计] 估计量；评价者

spec 英 [spek]、美 [spɛk] 投机；说明书；细则

classifier ['klæsɪfaɪɚ] ['klæsɪfaɪɚ] 分类器；

9. distributed

distributed 英 [dɪ'strɪbjʊtɪd]、美 [dɪ'strɪbjʊtɪd] 分布式的

device 英 [dɪ'vaɪs]、美 [dɪ'vaɪs] 设备

fraction 英 ['frækʃ(ə)n] 分数小部分(per\_process\_gpu\_memory\_fraction)

compute 英 [kəm'pjuːt]、美 [kəm'pjʊt] 计算；推断(compute\_gradients)

apply 英 [ə'plaɪ]、美 [ə'plaɪ] 应用(apply\_gradients)

cluster 英 ['klʌstə] 美 ['klʌstɚ] 群；簇

spec 英 [spek] 美 [spɛk] 细则(clusterSpec)

group 英 [gruːp]、美 [ɡrup] 组(tf.group更新变量和滑动平均模型值

define 英 [dɪ'faɪn] 定义；规定(tf.app.flags.DEFINE\_string)

flags [f:lai:g:s] 标记(FLAGS = tf.app.flags.FLAGS)

framework 英 ['freɪmwɜːk]、美 ['fremwɝk] 构架

replica 英 ['replɪkə] 仿制品；模型(replica\_device\_setter)

setter 英 ['setə]、美 ['sɛtɚ] 制订者

hook 英 [hʊk]、美 [hʊk] 挂钩，吊钩(StopAtStepHook)

monitored ['mɑnətɚ] 监控(MonitoredTrainingSession管理session)

chief 英 [tʃiːf]、美 [tʃif] 首领(is\_chief)

sync 英 [sɪŋk]、美 [sɪŋk] 同步(SyncReplicasOptimizer)

优化算法

一、 激活函数(Activation functions—非线性函数)

1. 激活函数的作用

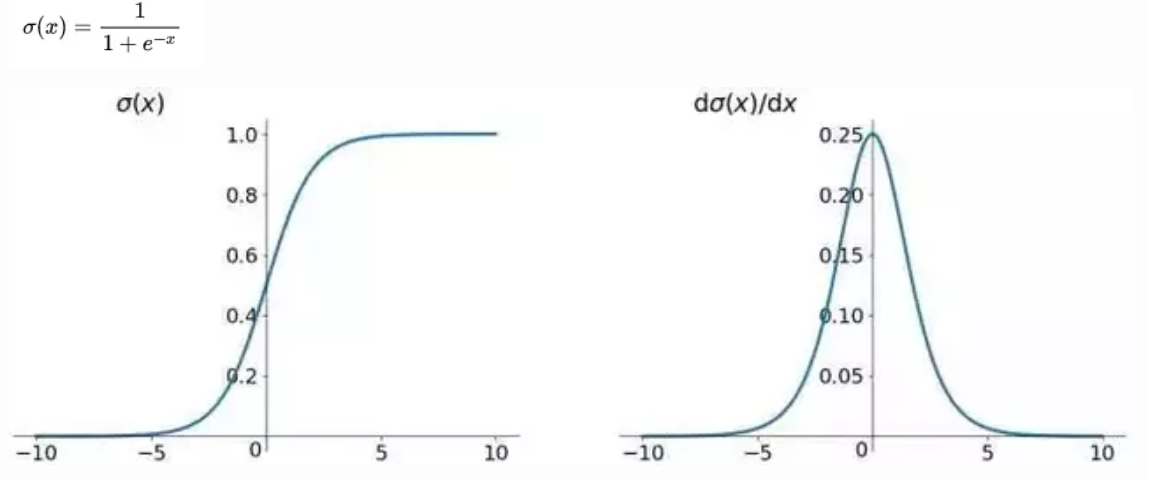
1.1 不带激活函数的单层感知机是一个线性分类器，不能解决线性不可分的问题(异或问题)。线性模型能够解决的问题是有限的，这就是线性模型最大的局限性，也是为什么深度学习要强调非线性。

1.2 激活函数是用来加入非线性因素的，提高神经网络对模型的表达能力，解决线性模型所不能解决的问题。

1.3 神经元中有输入信号时神经元不会立即反应，而是会抑制输入，直到输入增强，超过阈值时，信号才会传递到下一层，这个阈值可以通过激活函数来实现。

2. 常用激活函数

2.1 Sigmoid函数(S型生长曲线，具有指数函数形状)



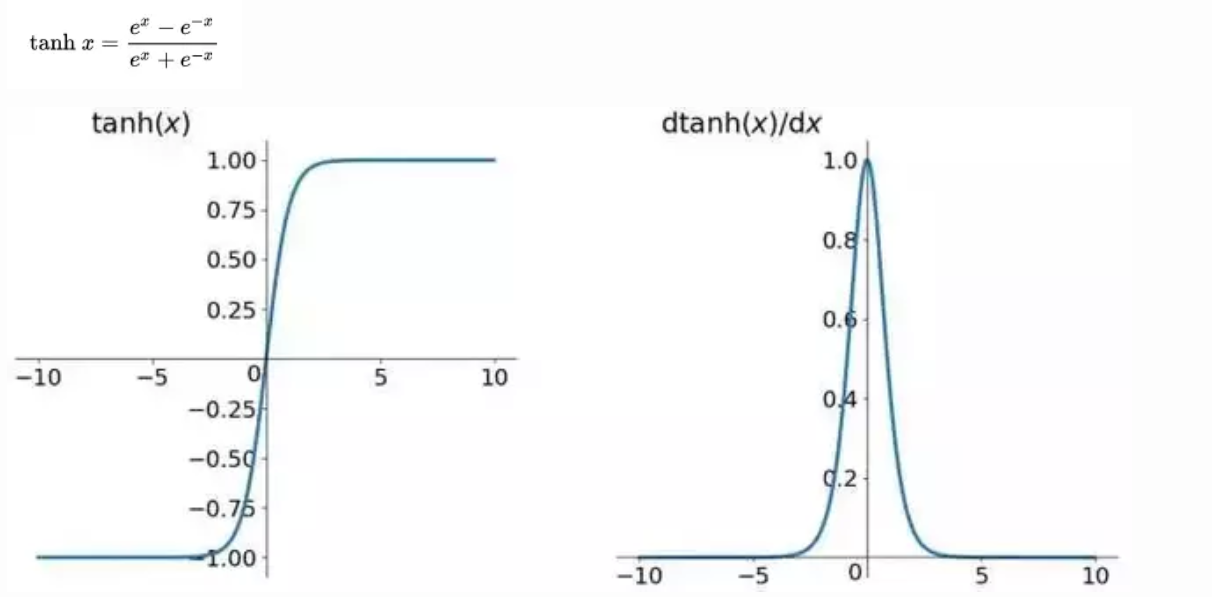
缺点：

1.> 容易出现梯度消失(gradient vanishing): 当神经元的激活在接近0或1处时会饱和，在这些区域梯度几乎为0，这就会导致梯度消失，几乎就有没有信号通过神经传回上一层。

2.> 它的输出不是以0为中心(zero-centered): 因为如果输入神经元的数据总是正数，那么关于W的梯度在反向传播的过程中，将会要么全部是正数，要么全部是负数，这将会导致梯度下降权重更新时出现z字型的下降。这会导致模型训练的收敛速度变慢.

3.> 幂运算相对耗时: 算激活函数时（指数运算），计算量大，反向传播求误差梯度时，求导涉及除法，计算量相对大. 相对于前两项，这其实并不是一个大问题，我们目前是具备相应计算能力的，但面对深度学习中庞大的计算量，最好是能省则省。

2.2 tanh函数(双曲正切函数)

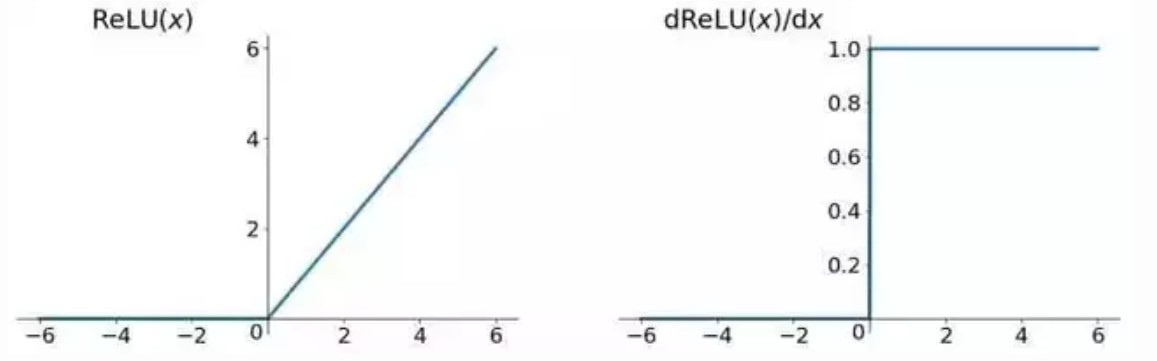


缺点：

1.> tanh读作Hyperbolic Tangent，如上图所示，它解决了zero-centered的输出问题，然而，gradient vanishing的问题和幂运算的问题仍然存在。

2.> tanh是sigmoid函数的一种变体，它的取值范围为【-1，1】，而不是sigmoid函数的【0，1】

2.3 relu函数(线性修正单元)



优点：

1.> 解决了gradient vanishing问题 (在正区间)

2.> 计算速度非常快，只需要判断输入是否大于0

3.> 收敛速度远快于sigmoid和tanh

4.> Relu会使一部分神经元的输出为0，这样就造成了网络的稀疏性，并且减少了参数的相互依存关系，缓解了过拟合问题的发生.

5.> 只需要一个阈值就可以得到激活值，而不用去算一大堆复杂的运算。计算速度快，对网络计算加速具有巨大作用。

6.> 只要R（x）= max（0，x），即如果x <0，R（x）= 0，如果x> = 0，则R（x）= x

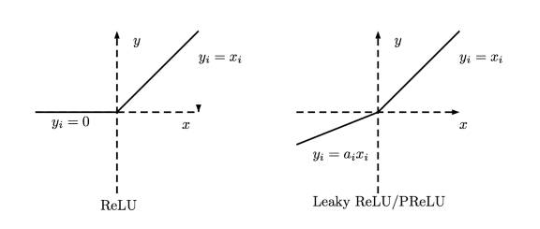
缺点:

1.> ReLU的输出不是zero-centered

2.> 它的局限性在于它只能在神经网络模型的隐藏层中使用。因此，对于输出层，我们应该使用Softmax函数来处理分类问题从而计算类的概率。而对于回归问题，它只要简单地使用线性函数就可以了。

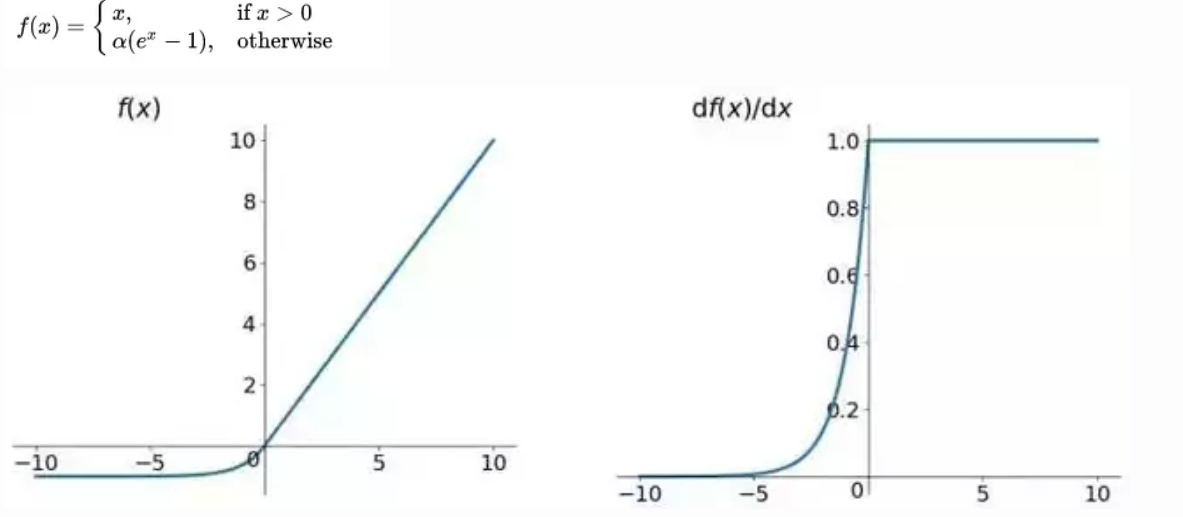
3.> ReLu函数的另一个问题是，一些梯度在训练过程中可能很脆弱，甚至可能会死亡(Dead ReLU Problem)。指的是某些神经元可能永远不会被激活，导致相应的参数永远不能被更新。有两个主要原因可能导致这种情况产生: (1) 非常不幸的参数初始化，这种情况比较少见 (2) learning rate太高导致在训练过程中参数更新太大，不幸使网络进入这种状态。解决方法是可以采用Xavier初始化方法，以及避免将learning rate设置太大或使用adagrad等自动调节learning rate的算法。

2.4 Leaky ReLU函数



人们为了解决Dead ReLU Problem，提出了将ReLU的前半段设为而非0,它引入了一个小斜坡从而保持更新值具有活力。另外一种直观的想法是基于参数的方法，即Parametric ReLU:，其中可由back propagation学出来。理论上来讲，Leaky ReLU有ReLU的所有优点，外加不会有Dead ReLU问题，但是在实际操作当中，并没有完全证明Leaky ReLU总是好于ReLU。

2.5 ELU (Exponential Linear Units) 函数



ELU也是为解决ReLU存在的问题而提出，显然，ELU有ReLU的基本所有优点，以及：

1.> 不会有Deal ReLU问题

2.> 输出的均值接近0，zero-centered

缺点：

它的一个小问题在于计算量稍大。类似于Leaky ReLU，理论上虽然好于ReLU，但在实际使用中目前并没有好的证据ELU总是优于ReLU。

2.6 Maxout函数。

我们还有另一个变体，它形成于ReLu函数和Leaky ReLu函数的结合，我们称之为Maxout函数。

总结：建议使用ReLU函数，但是要注意初始化和learning rate的设置，避免出现我们的模型在训练过程中遇到死亡神经元；可以尝试使用Leaky ReLU或ELU函数；不建议使用tanh，尤其是sigmoid函数，因为梯度消失问题（vanishing Gradient Problem）是一个很严重的问题，会在训练一个神经网络模型中导致更多问题。

二、指数衰减法

1. 学习率的作用

1.1 优化算法会反向修改函数中设置为Variable的变量值，使得误差逐步缩小。而这里的学习效率就是Variable更新变化的幅度。

1.2 如果幅度过大，参数就很可能跨过最优值，最后在最优值的两侧来回浮动。

1.3 如果幅度太小，又会大大的增加学习时间。

1.4 比较理想的做法是，在学习初期，将这个值设的大一些，当逐渐靠近最优解的时候，逐渐缩小学习效率使得获得的值更加靠近最优值。

1.5 学习率控制参数更新的速度，决定了参数每次更新的幅度。如果幅度过大，那么可能导致参数在极优值的两侧来回移动，无论进行多少轮迭代，参数不会收敛到一个极小值。相反，当学习率过小时，虽然能保证收敛性，但是这会大大降低优化速度。

2.tensorFlow实现指数衰减法.

通过下面这个函数，可以先使用较大的学习率来快速得到一个较优的解，然后随着迭代的继续逐步减少学习率，使得模型在训练后期更加稳定exponential\_decay函数会指数级地减少学习率。

tf.train.exponential\_decay(learning\_rate, global\_step, decay\_steps, decay\_rate, staircase=True)

**参数**：

learning\_rate - 初始学习率

global\_step - 用于衰减计算的全局步骤。 一定不为负数。喂入一次 BACTH\_SIZE 计为一次 global\_step

decay\_steps - 衰减速度，一定不能为负数，每间隔decay\_steps次更新一次learning\_rate值

decay\_rate - 衰减系数，衰减速率，其具体意义参看函数计算方程(对应α^t中的α)。

staircase - 若 ‘ True ’ ，则学习率衰减呈 ‘ 离散间隔 ’ （discrete intervals），具体地讲，`global\_step / decay\_steps`是整数除法，衰减学习率（ the decayed learning rate ）遵循阶梯函数；若为‘False’(默认为False) ，则更新学习率的值是一个连续的过程，每步都会更新学习率。

**返回值**：

与初始学习率 ‘ learning\_rate ’ 相同的标量 ’ Tensor ‘ 。

**优点**：

训练伊始可以使用较大学习率，以快速得到比较优的解。

后期通过逐步衰减后的学习率进行迭代训练，以使模型在训练后期更加稳定。

2.1 计算公式

Decayed\_learning\_rate = \

learing\_rate \* decay\_rate ^ (global\_step / decay\_steps)

2.2 staircase为true解释

当staircase为true时，decay\_steps通常代表了完整的使用一遍训练数据所需要的迭代轮数。这个迭代轮数也就是总训练样本数除以每一个batch中的训练样本数。这种设置的常用场景是每完整地过完一遍训练数据，学习率就减少一次。这可以使得训练数据集中的所有数据对模型训练有相等的作用。当使用连续的指数衰减学习率时，不同的训练数据有不同的学习率，而当学习率减少时，对应的训练数据对模型训练结果的影响也就小了。

2.3 test

Mnist训练中的指数衰减法运用。

tf.train.exponential\_decay(0.8, global\_step, mnist.train.num\_examples / BATCH\_SIZE, 0.99)

其中mnist.train.num\_examples / BATCH\_SIZE=55000/100=550，当staircase为False时，每一轮都会根据2.1中的公式更新学习率。当staircase为True时，每550轮会根据2.1中的公式更新学习率。一般来说初始学习率、衰减系数和衰减速度都是根据经验设置的。而且损失函数下降的速度和迭代结束之后总损失的大小没有必然的联系。也就是说并不能通过前几轮损失函数下降的速度来比较不同神经网络的效果。

三、损失函数

损失函数（loss）:用来表示预测值（y）与已知答案（y\_）的差距。在训练神经网络时，通过不断改变神经网络中所有参数，使损失函数不断减小，从而训练出更高准确率的神经网络模型。

1.交叉熵

1.1 如何判断一个输出向量和期望的向量有多接近呢?交叉熵(cross entropy)是常用的评判方法之一。交叉熵刻画了两个概率分布之间的距离，它是分类问题中使用比较广的一种损失函数。

1.2 交叉熵是一个信息论中的概念，它原本是用来估算平均编码长度的。给定两个概率分布p和q，通过q来表示p的交叉熵为:H(p,q)= -Σp(x)log q(x)。交叉熵刻画的是两个概率分布之间的距离，然而神经网络的输出却不一定是一个概率分布。概率分布刻画了不同事件发生的概率，当事件总数有限的情况下，概率分布函数p(X = x)满足: p(X = x)ε[0,1] 且Σp(X = x) =1。也就是说，任意事件发生的概率都在0到1之间，且总有某一个事件发生(概率的和为1)。

1.3 如何将神经网络前向传播得到的结果也变成概率分布呢？softmax回归就是一个非常常用的方法。Softmax回归本身可以作为一个学习算法来优化分类结果，但在TensorFlow中，softmax归回的参数被去掉了，它只是一层额外的处理层，将神经网络的输出变成一个概率分布。假设原始的神经网络输出为y1,y2,…,yn,那么经过softmax回归处理之后的输出为:



从以上公式中可以看出，原始神经网络的输出被用作置信度来生成新的输出，而新的输出满足概率分布的所有要求。这个新的输出可以理解为经过神经网络的推导，一个样例为不同类别的概率分别是多大。这样就把神经网络的输出也变成了一个概率分布，从而可以通过交叉熵来计算预测的概率分布和真实答案的概率分布之间的距离了。

1.4 从交叉熵的公式中可以看到交叉熵函数不是对称的（H(p,q) ≠ H(q,p)）,它刻画的是通过概率分布q来表达概率分布p的困难程度。因为正确答案是希望得到的结果，所以当交叉熵作为神经网络的损失函数时，p代表的是正确答案，q代表的是预测值，交叉熵刻画的是两个概率分布的距离，也就是说交叉熵值越小，两个概率分布越接近。

1.5 TensorFlow表示

cross\_entropy = - tf.reduce\_mean(y\_ \* tf.log(tf.clip\_by\_value(y, 1e-10, 1.0)) + (1 - y) \* tf.log(tf.clip\_by\_value(1 - y, 1e-10, 1.0)))

因为交叉熵一般会与softmax回归一起使用，所以TensorFlow对这两个功能进行了统一封装，并提供了tf.nn.softmax\_cross\_entropy\_with\_logiths(labels=y\_, logits=y),其中y代表了原始神经网络的输出结果，而y\_给出了标准答案。这样通过一个命令就可以得到使用了softmax回归之后的交叉熵。

在只有一个正确答案的分类问题中，TensorFlow提供了tf.nn.sparse\_softmax\_cross\_entropy\_with\_logits函数来进一步加速计算过程。

2 均方误差

与分类问题不同，回归问题解决的是对具体数值的预测。比如房价预测、销量预测等都是回归问题。这些问题需要预测的不是一个事先定义好的类别，而是一个任意实数。解决回归问题的神经网络一般只有一个输出节点，这个节点的输出值就是预测值。对于回归问题，最常用的损失函数是均方误差（MSE, mean squared error）。定义如下：

MSE(y,yI)=

其中yi为一个batch中第i个数据的正确答案，而yiI为神经网络给出的预测值。以下代码用TensorFlow实现均方误差损失函数：

mse = tf.reduce\_mean(tf.square(y\_ - y))

其中y代表了神经网络的输出答案，y\_代表了标准答案。

四、优化器

1. 概念

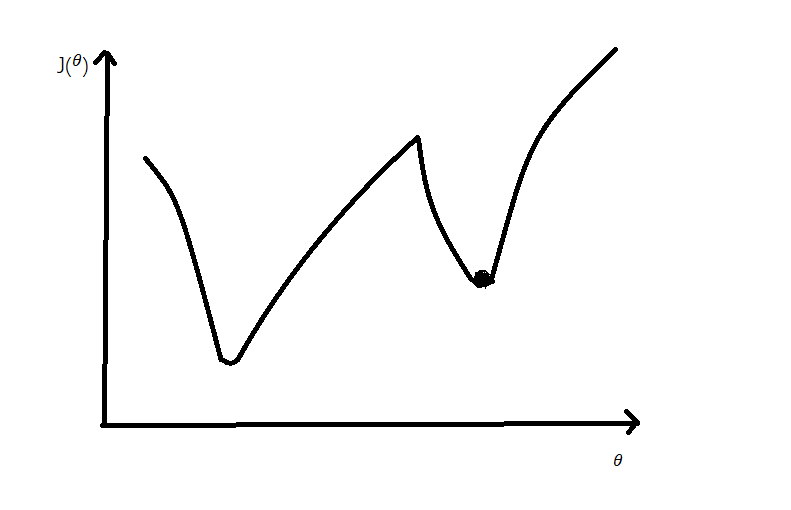
1.1 反向传播算法(backpropagation)和梯度下降算法(gradient decent)调整神经网络中参数的取值。梯度下降算法主要用于优化单个参数的取值，而反向传播算法给出了一个高效的方式在所有参数上使用梯度下降算法，从而使神经网络模型在训练数据上的损失函数尽可能小。

1.2 参数的梯度可以通过求偏导的方式计算，对于参数θ，其梯度为。有了梯度，还需要定义一个η(learning rate)来定义每次参数更新的幅度。从直观上理解，可以认为学习率定义的就是每次参数移动的幅度。通过参数的梯度和学习率，参数更新的公式为:。

下面给出了一个具体的例子来说明梯度下降算法是如何工作的。假设要通过梯度下降算法来优化参数x，使得损失函数J(x)=x2的值尽量小。梯度下降算法的第一步需要随机产生一个参数x的初始值，然后在通过梯度和学习率来更新参数x的取值。在这个样例中，参数x的梯度为▽==2x，那么使用梯度下降算法每次对参数x的更新公式为xn+1=xn-η▽。假设参数的初始值为5，学习率为0.3，那么这个优化过程可以总结为如下表：

|  |  |  |  |
| --- | --- | --- | --- |
| 轮数 | 当前轮参数值 | 梯度 \* 学习率 | 更新后参数值 |
| 1 | 5 | 2\*5\*0.3=3 | 5-3=2 |
| 2 | 2 | 2\*2\*0.3=1.2 | 2-1.2=0.8 |
| 3 | 0.8 | 2\*0.8\*0.3=0.48 | 0.8-0.48=0.32 |
| 4 | 0.32 | 2\*0.32\*0.3=0.192 | 0.32-0.192=0.128 |
| 5 | 0.128 | 2\*0.128\*0.3=0.0768 | 0.128-0.0768=0.0512 |

从上表可以看出，经过5次迭代之后，参数x的值变成了0.0512，这个和参数最优值0已经比较接近了。神经网络的优化过程可以分为两个阶段，**第一个阶段先通过前向传播算法(权重\*输入值)计算得到预测值，并将预测值和真实值作对比得出两者之间的差距。然后在第二个阶段通过反向通过反向传播算法计算损失函数对每一个参数的梯度，再根据梯度和学习率使用梯度下降算法更新每一个参数。只有当损失函数为凸函数时，梯度下降算法才能保证达到全局最优解。**如果为其他函数，比如下图时：



梯度下降算法并不能保证被优化的函数达到全局最优解。图中给出的函数就有可能只能得到局部最优解而不是全局最优解。在小黑点处，损失函数的偏导为0，于是参数就不会再进一步更新。在这个样例中，如果参数x的初始值落在右侧深色的区间中，那么通过梯度下降得到的结果都会落在小黑点代表的局部最优解。只有当x的初始值落在左侧浅色的区间时梯度下降才能给出全局最优答案。由此可见在训练神经网络时，参数的初始值会很大程度影响最后得到的结果。

1.3 **除了不一定能达到全局最优，梯度下降算法的另外一个问题就是计算时间太长**。因为要在全部训练数据上最小化损失，所以损失函数J(θ)是在所有训练数据上的损失和。这样在每一轮迭代中都需要计算在全部训练数据上的损失函数。在海量训练数据下，要计算所有训练的损失函数是非常损耗时间的。为了加速训练过程，可以使用随机梯度下降的算法(stochastic gradient descent)。这个算法优化的不是在全部训练数据上的损失函数，而是在每一轮迭代中，随机优化某一条训练数据上的损失函数。这样每一轮参数更新的速度就大大加快了。因为随机梯度下降算法每次优化的只是某一条数据上的损失函数，所以它的问题也非常明显：在某一条数据上损失函数更小并不代表在全部数据上损失函数更小，于是使用随机梯度下降优化得到的神经网络甚至可能无法达到局部最优。

1.4 为了综合梯度下降算法和随机梯度下降算法的优缺点，在实际应用中一般采用这两个算法的折中—--每次计算一小部分训练数据的损失函数。这一小部分数据被称之为一个batch。通过矩阵运算，每次在一个batch上优化神经网络的参数并不会比单个数据慢太多。另一方面，每次使用一个batch可以大大减少收敛所需要的迭代轮数，同时可以使收敛到的结果更加接近梯度下降的效果。

2 一些优化算法

AdadeltaOptimizer：实现Adadelta算法的优化器

AdagradDAOptimizer：用于稀疏线性模型的Adagrad Dual Averaging算法。

AdagradOptimizer：实现Adagrad算法的优化器。

**AdamOptimizer**：实现Adam算法的优化器。**(比较常用的高级优化器)**

Adam(Adaptive Moment Estimation)本质上是带有动量项的RMSprop，它利用梯度的一阶矩估计和二阶矩估计动态调整每个参数的学习率。Adam的优点主要在于经过偏置校正后，每一次迭代学习率都有个确定范围，使得参数比较平稳。

**特点：**

结合了Adagrad善于处理稀疏梯度和RMSprop善于处理非平稳目标的优点

对内存需求较小

为不同的参数计算不同的自适应学习率

也适用于大多非凸优化

适用于大数据集和高维空间

FtrlOptimizer：实现FTRL算法的优化器

**GradientDescentOptimizer**：优化器，实现梯度下降算法。

**MomentumOptimizer**：实现Momentum算法的优化器。**(比较常用的高级优化器)**

momentum是模拟物理里动量的概念，积累之前的动量来替代真正的梯度

**特点：**

下降初期时，使用上一次参数更新，下降方向一致，乘上较大的μμ能够进行很好的加速

下降中后期时，在局部最小值来回震荡的时候，gradient→0gradient→0，μμ使得更新幅度增大，跳出陷阱

在梯度改变方向的时候，μμ能够减少更新

总而言之，momentum项能够在相关方向加速SGD，抑制振荡，从而加快收敛

ProximalAdagradOptimizer：优化器，实现Proximal Adagrad算法。

ProximalGradientDescentOptimizer：优化器，实现近端梯度下降算法

**RMSPropOptimizer**：实现RMSProp算法的优化器。(高级优化器)

SyncReplicasOptimizer：要同步的类，聚合渐变并将它们传递给优化器。

具体可以参考下面链接：

<https://blog.csdn.net/u012759136/article/details/52302426>

<https://tensorflow.google.cn/api_docs/python/tf/train>

五、滑动平均模型

1 作用

在采用随机梯度下降算法训练神经网络时，使用滑动平均模型在很多应用中都可以在一定程度提高最终模型在测试数据上的表现。使模型在**测试数据上更健壮(robust)。**

2 TensorFlow实现

通过tf.ExponentialMovingAverage(decay, num\_updates=None)来实现滑动平均模型。Decay为衰减率，用于控制模型更新的速度。ExponentialMovingAverage对每一个变量会维护一个影子变量(shadow variable)，这个影子变量的初始值就是相应变量的初始值，而每次运行变量更新时，影子变量的值会更新为:

shadow\_variable = decay \* shadow\_variable + (1-decay) \* variable

其中shadow\_variable为影子变量，variable为待更新的变量，decay为衰减率。可以看出，decay决定了模型更新的速度，decay越大模型越趋于稳定。一般decay设为0.99。为了使得模型在训练前期可以更新得更快，ExponentialMovingAverage还提供了num\_updates参数来动态设置decay的大小。如果在ExponentialMovingAverage初始化时提供了num\_updates参数，那么每次使用的衰减率将是：

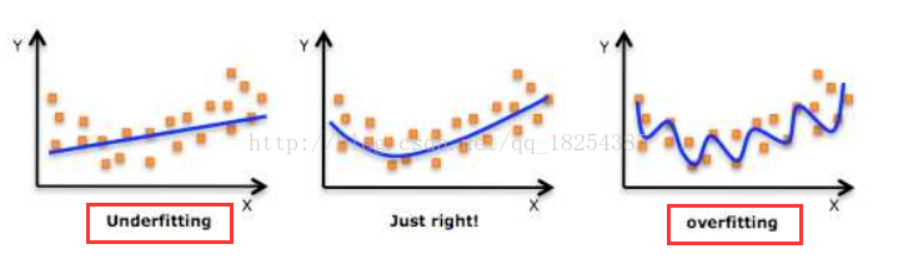
min{ decay, }



六、过拟合

1 作用

所谓过拟合，指的是当一个模型过为复杂之后，它可以很好地“记忆”每一个训练数据中随机噪音的部分而忘记了要去“学习”训练数据中通用的趋势。对于未知数据可能无法做出可靠的判断。如下第三个图就是过拟合



2.过拟合和欠拟合比较

2.1 欠拟合

根本原因是特征维度过少，模型过于简单，导致拟合的函数无法满足训练集，

误差较大。

解决方法：增加特征维度，增加训练数据；

2.2 过拟合：根本原因是特征维度过多，模型假设过于复杂，参数过多，训练数据过少，噪声过多，导致拟合的函数完美的预测训练集，但对新数据的测试集预测结果差。 过度的拟合了训练数据，而没有考虑到泛化能力。

解决方法：（1）减少特征维度；（2）正则化，降低参数值。

3.过拟合原因

3.1. 训练集的数量级和模型的复杂度不匹配。训练集的数量级要小于模型的复杂度；

3.2. 训练集和测试集特征分布不一致；

3.3. 样本里的噪音数据干扰过大，大到模型过分记住了噪音特征，反而忽略了真实的输入输出间的关系；

3.4. 权值学习迭代次数足够多(Overtraining)，拟合了训练数据中的噪声和训练样例中没有代表性的特征。

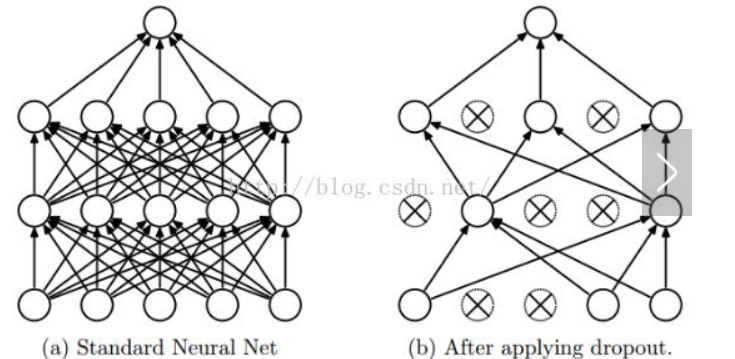
3.5 数据太少+模型太复杂

4. 解决方法

4.1获取更多数据 ：从数据源头获取更多数据；数据增强（Data Augmentation）。训练集越多，过拟合的概率越小。在计算机视觉领域中，增广的方式是对图像旋转，缩放，剪切，添加噪声等。

4.2 使用合适的模型(simpler model structure)：减少网络的层数、神经元个数等均可以限制网络的拟合能力；调小模型复杂度，使其适合自己训练集的数量级（缩小宽度和减小深度）

4.3 dropout：深度学习中最常用的正则化技术是dropout，随机的丢掉一些神经元(让神经元以一定的概率不工作)；dropout一般只在全连接层而不是卷积层或者池化层使用



如上图所示，左边a图是没用用dropout方法的标准神经网络，右边b图是在训练过程中使用了dropout方法的神经网络，即在训练时候以一定的概率p来跳过一定的神经元。

tf.nn.dropout(fcl, 0.5)

4.4 正则化，在训练的时候限制权值变大。参数太多，会导致我们的模型复杂度上升，容易过拟合，也就是我们的训练误差会很小。正则化是指通过引入额外新信息来解决机器学习中过拟合问题的一种方法。这种额外信息通常的形式是模型复杂性带来的惩罚度。正则化可以保持模型简单，另外，规则项的使用还可以约束我们的模型的特性。依据是奥克姆剃刀原理。

正则化的思想就是在损失函数中加入刻画模型复杂程度的指标。假设用于刻画模型在训练数据上表现的损失函数为J(θ)，那么在优化时不是直接优化J(θ)，而是优化J(θ)+ λR(w)。其中R(w)刻画的是模型的复杂程度，而λ表示模型复杂损失在总损失中的比例。注意这里θ表示的是一个神经网络中所有的参数，它包括边上的权重W和偏置项b。

4.4.1 L1正则化，计算公式为：

R(w)=‖w‖1 = Σ|wi|

tf.contrib.layers.l1\_regularizer(lambda)(w)

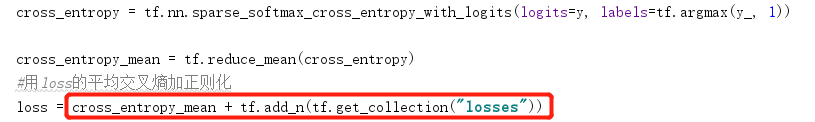
lambda(0.001)参数表示了正则化项的权重，也就是J(θ)+ λR(w)中的λ，w为需要计算正则化损失的参数(权重和偏置)。

4.4.2 L2正则化，计算公式为：

R(w)=‖w‖22 = Σ|wi2|

tf.contrib.layers.l2\_regularizer(0.001)(weights)

无论哪种正则化公式，基本思想都是希望通过限制权重的大小，使得模型不能任意拟合训练数据中的随机噪音。两个的区别是，L1正则化会让参数变的更加稀疏(所谓稀疏是指会有更多的参数变为0，这样可以达到类似特征选取的功能。)，而L2不会。L1正则化的计算公式不可导，而L2正则化公式可导。**正则化只在训练数据集上使用，测试数据集上不需要正则化,** **只用全连接层的权重需要加入正则化，在卷积层不需要加入。**在深度学习中，L1会趋向于产生少量的特征，而其他的特征都是0增加网络稀疏性；而L2会选择更多的特征，这些特征都会接近于0，防止过拟合。神经网络需要每一层的神经元尽可能的提取出有意义的特征，而这些特征不能是无源之水，因此L2正则用的多一些。



4.5 限制训练时间；通过评估测试；

4.6 增加噪声 Noise： 输入时+权重上（高斯初始化） ；

4.7 数据清洗(data ckeaning/Pruning)：将错误的label 纠正或者删除错误的数据。

4.8 结合多种模型(ensemble)： Bagging用不同的模型拟合不同部分的训练集；Boosting只使用简单的神经网络；集成学习算法也可以有效的减轻过拟合。Bagging通过平均多个模型的结果，来降低模型的方差。Boosting不仅能够减小偏差，还能减小方差

4.9 提前停止（early stopping）：就是让模型在训练的差不多的时候就停下来，比如继续训练带来提升不大或者连续几轮训练都不带来提升的时候，这样可以避免只是改进了训练集的指标但降低了测试集的指标。

对模型进行训练的过程即是对模型的参数进行学习更新的过程，这个参数学习的过程往往会用到一些迭代方法，如梯度下降（Gradient descent）学习算法。Early stopping便是一种迭代次数截断的方法来防止过拟合的方法，即在模型对训练数据集迭代收敛之前停止迭代来防止过拟合。

Early stopping方法的具体做法是，在每一个Epoch结束时（一个Epoch集为对所有的训练数据的一轮遍历）计算validation data的accuracy，当accuracy不再提高时，就停止训练。这种做法很符合直观感受，因为accurary都不再提高了，在继续训练也是无益的，只会提高训练的时间。那么该做法的一个重点便是怎样才认为validation accurary不再提高了呢？并不是说validation accuracy一降下来便认为不再提高了，因为可能经过这个Epoch后，accuracy降低了，但是随后的Epoch又让accuracy又上去了，所以不能根据一两次的连续降低就判断不再提高。一般的做法是，在训练的过程中，记录到目前为止最好的validation accuracy，当连续10次Epoch（或者更多次）没达到最佳accuracy时，则可以认为accuracy不再提高了。此时便可以停止迭代了（Early Stopping）。这种策略也称为“No-improvement-in-n”，n即Epoch的次数，可以根据实际情况取，如10、20、30……

4.10 批量正则化（BN）：就是将卷积神经网络的每层之间加上将神经元的权重调成标准正态分布的正则化层，这样可以让每一层的训练都从相似的起点出发，而对权重进行拉伸，等价于对特征进行拉伸，在输入层等价于数据增强。注意正则化层是不需要训练。

七、BN(Batch Normalization)

<https://blog.csdn.net/m0_37699976/article/details/81584101>

<https://blog.csdn.net/donkey_1993/article/details/81871132>

1.优点

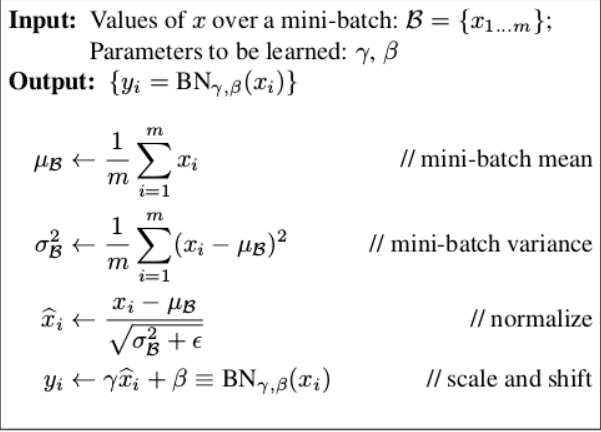
1.1 加快训练速度，这样我们就可以使用较大的学习率来训练网络。BN层可以加速网络收敛速度。

1.2 提高网络的泛化能力。

1.3 BN层本质上是一个归一化网络层，可以替代局部响应归一化层（LRN层）。

1.4 可以打乱样本训练顺序（这样就不可能出现同一张照片被多次选择用来训练）论文中提到可以提高1%的精度。

2.计算流程



2.1.计算样本均值。

2.2.计算样本方差。

2.3.样本数据标准化处理。

2.4.进行平移和缩放处理。引入了γ和β两个参数。来训练γ和β两个参数。引入了这个可学习重构参数γ、β，让我们的网络可以学习恢复出原始网络所要学习的特征分布。

3. 为什么BN层一般用在线性层和卷积层后面，而不是放在非线性单元后

原文中是这样解释的，因为非线性单元的输出分布形状会在训练过程中变化，归一化无法消除他的方差偏移，相反的，全连接和卷积层的输出一般是一个对称,非稀疏的一个分布，更加类似高斯分布，对他们进行归一化会产生更加稳定的分布。其实想想也是的，像relu这样的激活函数，如果你输入的数据是一个高斯分布，经过他变换出来的数据能是一个什么形状？小于0的被抑制了，也就是分布小于0的部分直接变成0了，这样不是很高斯了。

4.TensorFlow实现

tf.nn.batch\_normalization(x, mean, variance, offset, scale, variance\_epsilon, name=None)

各个参数的说明如下：

x：代表任意维度的输入张量。

mean:代表样本的均值。

variance：代表样本的方差。

offset：代表偏移，即相加一个转化值，也是公式中的beta。

scale：代表缩放，即乘以一个转化值，也是公式中的gamma。

variance\_epsilon：是为了避免分母为0的情况下，给分母加上的一个极小值，默认即可。

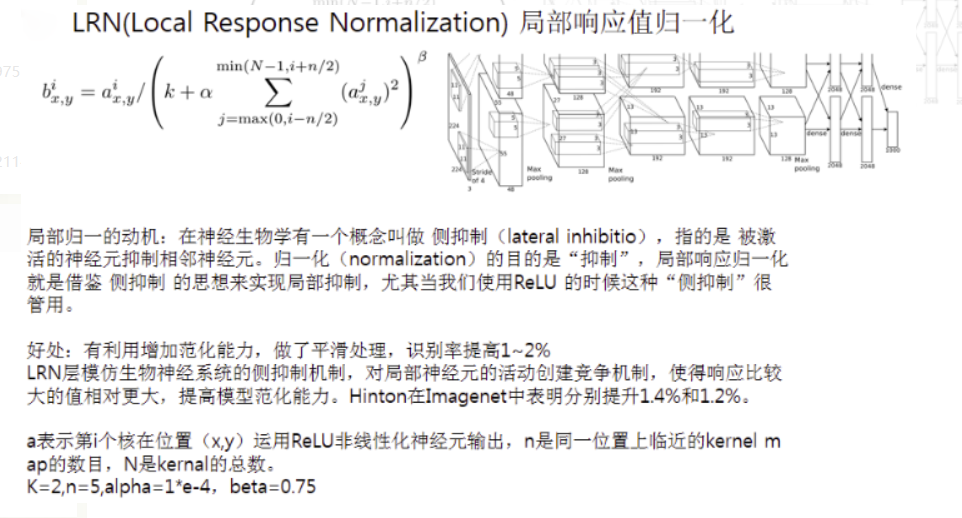
name：名称。

要想使用这个整数，必须由另一个函数配合使用，tf.nn.moments，由它来计算均值和方差，然后就可以使用BN了。

八、LRN(Local Response Normalization局部归一化)

1.作用

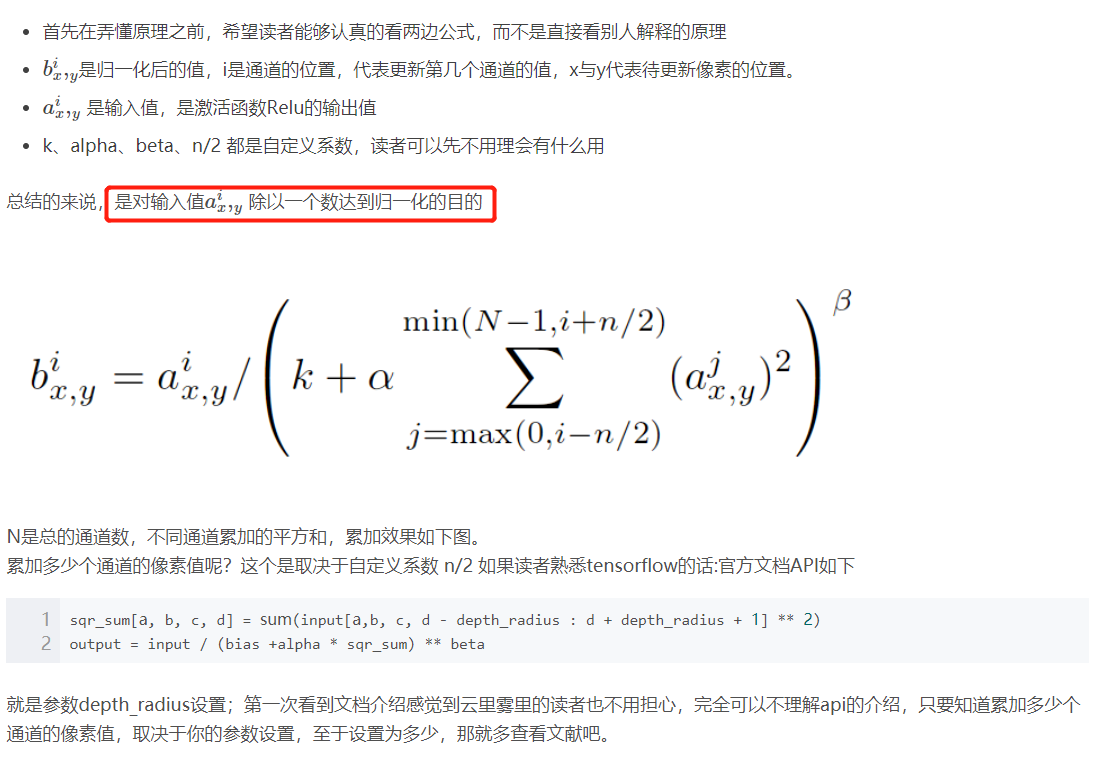
对局部神经元的活动创建竞争机制，使得其中响应比较大的值变得相对更大，并抑制其他反馈较小的神经元，增强了模型的泛化能力。AlexNet模型中首次使用。



2.为什么输入数据需要归一化（Normalized Data）

归一化后有什么好处呢？原因在于神经网络学习过程本质就是为了学习数据分布，一旦训练数据与测试数据的分布不同，那么网络的泛化能力也大大降低；另外一方面，一旦每批训练数据的分布各不相同(batch 梯度下降)，那么网络就要在每次迭代都去学习适应不同的分布，这样将会大大降低网络的训练速度，这也正是为什么我们需要对数据都要做一个归一化预处理的原因。

对于深度网络的训练是一个复杂的过程，只要网络的前面几层发生微小的改变，那么后面几层就会被累积放大下去。一旦网络某一层的输入数据的分布发生改变，那么这一层网络就需要去适应学习这个新的数据分布，所以如果训练过程中，训练数据的分布一直在发生变化，那么将会影响网络的训练速度。



3.TensorFlow实现

tf.nn.lrn (input, depth\_radius=5, bias=1, alpha=1, beta=0.5, name=None)