# 2018.2.23

## 已解决问题

1.将csv文件构成图片数组进行读取。

## 待解决问题

1.解决方案

将train.csv文件中的数据按照7:3分成训练数据和测试数据，保证数据来自同一分布。

2.**评价指标**，由于不是单纯的分类问题，所以评价指标不应该是标签，**而是平均化后的误差所占原始数据的比值。**

3.**网络结构** 由于没有先例，所以对于网络结构使用**由简单到复杂的**原则，对于使用的trick也是由简单到复杂的原则，先构建一个简单的网络，再向其中不断增加层次和节点信息。使用**不同的节点数量**，**不同的网络层次，卷积核的数量和结构，卷积使用的函数。**

4.**结果的复用**，为了写论文的便捷而不用重复run程序，我们将程序每次运行的结果进行保存，保存到文件中，下次进行读取。

### Train.csv

35444条数据。

24500条作为训练数据 a\_train.csv

10500条作为测试数据 a\_test.csv

# 2018.2.24

## 已解决问题

### Train.csv

将train.csv成功分为训练数据和测试数据。其中：

24500条作为训练数据 a\_train.csv

10500条作为测试数据 a\_test.csv

将 tensorflow CIFAR-10模型转移到CNN\_Proto模型中。

## 未解决问题

### Loss function

Loss function 由于不是一个分类而是一个回归问题，使用交叉熵函数则得到的是一个基于概率分布判断彼此的距离，一般运用于分类问题，对于回归问题，一般还是使用均方差比较好。使用交叉熵一般和softmax搭配使用，这样如果我们预测的属性值之间本身不属于一个分布，那么怎么能很好的收敛呢？

使用均方误差（MSE），并且只预测一种输出的属性。

这是一种多任务学习的方式

<http://blog.csdn.net/u013555719/article/details/78459904>

使用均方误差（MSE）

### 模型评价指标函数，需要查看其它论文寻找思路。

预测多个值时其实是一种多任务的行为，我们是否有必要将预测的三个指标分成三个不同的网络，使用迁移学习的方式进行学习，即去除顶层设计后训练新的值。

**决定系数（训练集上，测试集上均可使用）**

**RPD（相对误差分析，对测试集上使用）**

**对于R2系数和RPD需要将最后三项分别取出后进行绘图。**

### 代码调试

**保存每一层weight的值，以及如何更好的输出weight值。**

# 2018.2.26

## 已解决问题

写出模型R2函数

**思路**

由于要将batch\_size中第二维度的三列分开提取出来，我们首先要确定model\_output的形状。

print(sess.run(tf.shape(model\_output))) #[128,3]

对于tensor使用矩阵拆分的操作。

将其拆分为3个[batch\_size]维度的向量

1. 使用tf.unstack()函数，出现too many values to unpack (expected 3)错误。
2. 对于查看数据而言

2.1 在定义函数及变量时使用print(sess.run(tf.shape(a)))只能看见tensor的信息，看不见定义数据后该tensor形状的变化

2.2 在声明模型时，但是没有执行时，我们可以使用print(sess.run(tf.shape(a)))看到shape的数据，不能使用print(sess.run(a))直接输出此变量的数值。

2.3 在init函数被激活，数据填充完毕(feed\_dict或输入线程激活后)才能使用print(sess.run(a))查看变量完整数值。

3.模型开发时，为确保不出现bug.先只在训练集上使用小batch\_size,少迭代次数进行操作。

## 未解决问题

R系数函数得到的值为nan,经过调试，发ss\_tot,和ss\_err得到值为0。

发现传进入的targets竟然是一样的值

# 2018.2.27

## 已解决问题

关于R2系数的问题，已经解决，

经过把所有值返回后发现，1：一个变量名打错了2：漏写一个函数名，直接返回了变量值。

### 性能指标

R系数3\*2 训练集和测试集上的三种属性

LOSS(MSE) 2 训练集和测试集

RMSE 3\*2 训练集和测试集上三种属性

RPD 3测试集上三种属性

6+2+6+3=17

## 未解决问题

Batch\_size中的所有数据，经过一段时间训练后，logtis会变得一样，没有区分性。

**经过：**

**RPD值用以表示对于数据的分化能力，用以衡量测试集上的预测能力，越大越好。**

但是，将RPD函数编写好后，发现其经过几次迭代以后变成0，原先以为是自己的程序有逻辑的bug,但是后来发现是，经过一段时间训练后，batch的所有元素输出全部变成一样的。现在我还不知道原因，但是这是一个很严重的问题。

现在有两种可能性：

1. 程序本身的预测性能不好
2. RPD对于测试集上一组数据只能进行一次

归根结底，还是对于tensorflow底层的数据feed的实现不懂，还需要进一步调试自己的网络。

**10次迭代后的RPD值**

****

****

****

**对于10次后的test\_out值**

**[[ 0.97685474 1.0584029 1.09349906]**

**[ 0.97685474 1.0584029 1.09349906]**

**[ 0.97685474 1.0584029 1.09349906]**

**[ 0.97685474 1.0584029 1.09349906]**

**[ 0.97685474 1.0584029 1.09349906]**

**[ 0.97685474 1.0584029 1.09349906]**

**[ 0.97685474 1.0584029 1.09349906]**

**[ 0.97685474 1.0584029 1.09349906]**

**[ 0.97685474 1.0584029 1.09349906]**

**[ 0.97685474 1.0584029 1.09349906]**

**[ 0.97685474 1.0584029 1.09349906]**

**[ 0.97685474 1.0584029 1.09349906]**

**[ 0.97685474 1.0584029 1.09349906]**

**[ 0.97685474 1.0584029 1.09349906]**

**[ 0.97685474 1.0584029 1.09349906]**

**[ 0.97685474 1.0584029 1.09349906]**

**[ 0.97685474 1.0584029 1.09349906]**

**[ 0.97685474 1.0584029 1.09349906]**

**[ 0.97685474 1.0584029 1.09349906]**

**[ 0.97685474 1.0584029 1.09349906]]**

# 2018.3.2

## 未解决的问题

R2指标为负数，RPD值几乎为0，对于数据没有较好的区分度，自己判定为是一种欠拟合的状态。参数需要细化，需要精确化。网络的结构也需要调整。几乎batch中的所有值在进入网络后都会成为相同的输出。

为了精确参数，我们现在需要自动保存最后一次迭代时的model\_output（确定不同的的输入会产生不同的输出）

每层最后一次迭代时的output

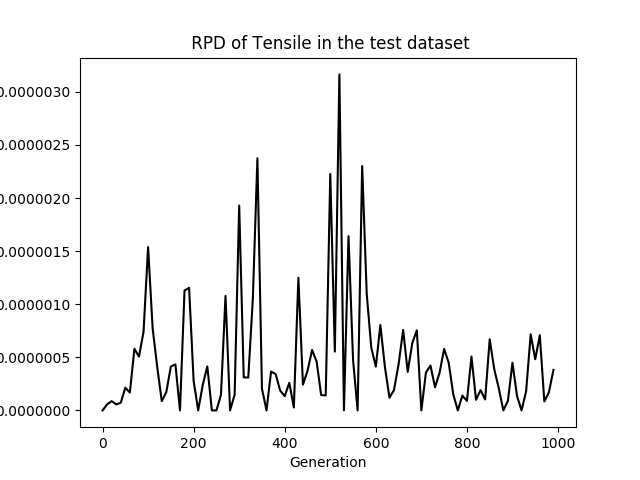
训练集上的最终loss值。（最终需要保存哪些参数还要进一步确认）

要保存在不同的网络结构时模型评价指标的图像

### 进展

为了进一步确认是自己的代码有逻辑问题还是拟合程度不够，我们按照网上罗列的trick对自己的网络进行调试，希望发现自己的问题。

1. 按照方法，我们对自己的程序进行可视化处理，而不使用肉眼进行观察。
2. 减小训练批次，增大训练迭代次数，减小测试批次。
3. 我们使用个10个数据训练1000次，然后随机从测试集中挑选10个数据进行测试，发现输出的model\_output不在是0，而且RPD值开始波动。证明我们的网络没有问题。自此神经网络CNN调试之路开启。



现在，自己很不知道方向，不知道如何前进，搜索一下网上的tricks,多看多思考多在自己的代码上测试。

### 学习率的调节

在proto1.1模型中，使用0.05的学习率会出现蓝色的loss情况（具体参见cs231n调参指南），学习率过低使模型不收敛。



如果是0.1的话学习率过大，高的学习率会使得曲线迅速下降，会在一个不好的结果下收敛。

应该把训练集的loss和测试集上的loss画在一张图片上。准确度也应该画在一张图片上。

# 2018.3.3

根据神经网络训练中的Tricks之高效BP（反向传播算法）中的内容。

## --对输入进行标准化Normalize

应该对自己的输入进行审查，观察每层的输入是不是全部都是正数，如果全部都是正数，肯定会带来收敛非常慢的情况。

然而，实际上，如果训练样本的输入变量的均值远离于0，都会让参数的更新倾向于一个特定的方向，从而降低了学习的速度。因此，将整个训练集每个样本的输入变量的均值偏移到0处是有好处的。而且，这种启发式的方法应该在网络的每一层都使用上，换句话说，我们希望每个节点的输出的均值都接近于0，因为这些输出实际上是下一层的输入**（也就是说我们应该每一层都做标准化的操作）**

还有一个加速收敛的方法是对样本进行缩放，让每一个特征维度都具有相同的协方差。

协方差取1是个不错的选择

**总结来说，对输入的变换如下：**

**A、 训练集的每个输入变量的均值要接近于0；**

**B、 对输入变量进行缩放，使他们的方差具有相同的值；**

**C、 输入变量最好是不相关的。**

## --参数初始化

它需要数据标准化、sigmoid的选择和参数初始化的选择这三者的协调

那参数就应该从一个均值为0，标准差为σw=m-1/2的分布（例如正态分布）中采样得到。

## --学习率

Tricks：

A、 给每个参数自己的学习率；

B、 学习率应该和该节点的输入个数的平方根成比例；

C、 低层参数的学习率应该比高层的大。

根据《Deep Learning for Vision Tricks of the Trade》中的内容

Pick learning rate by running on a subset of the data

通过在数据的子集上运行来选择学习速率

Decay learning rate by a factor of ~100 or more by the end of training

在训练结束时，衰减学习率是100或更多

# 2018.3.4

根据南京大学《Must Know Tips/Tricks in Deep Neural Networks》的描述

## 数据预处理

对于直接输进入网络的数据没有经过预处理

Tensorflow数据预处理

<https://github.com/tensorflow/transform>

## 正则化与Dropout

网络没有针对过拟合而使用的正则化和Dropout的操作。

## 参数初始化

全零初始化是一种十分错误的方法，更好的初始化方式应该是有一半正数一半负数的情况。因为如果网络中的每个神经元都计算出相同的输出，那么它们也将在反向传播期间计算相同的梯度并进行完全相同的参数更新。换句话说，如果它们的权重被初始化为相同，那么神经元之间就不存在不对称的来源。

## 当前建议

如前所述，通过校准神经元方差的先前初始化不考虑ReLU。He等人最近关于这个主题的论文。[[4]](http://arxiv.org/abs/1502.01852)专门为ReLU推导初始值，得出网络中神经元方差应该如下的结论2.0 / n的：

>>> w = np.random.randn（n）\* sqrt（2.0 / n）＃当前建议

这是目前在实践中使用的建议，正如[[4]中](http://arxiv.org/abs/1502.01852)所讨论的。

## 第四章：训练建议

现在一切准备就绪，我们开始训练神经网络吧。

### 卷积核和池化层的大小设置

训练期间，输入图像的大小优选为2的幂，诸如32（例如，*CIFAR-10*），*64,224*（例如，常用的*ImageNet*），384或512等。更重要的是，我们应该选用小的卷积核（3\*3）和小的步长（1）以及设置0填充（O padding）这不仅减少了参数的数量，而且提高了整个深度网络的准确率。同时，上面提到的一个特例，即3\*3步长为1的滤镜可以保留图像/特征图的空间大小。对于池化层，常用的池化大小是2\*2。

### 学习率

另外，正如Ilya Sutskever的博客[[2]](http://yyue.blogspot.sg/2015/01/a-brief-overview-of-deep-learning.html/)所描述的那样，他建议按照小批量大小（mini-batch-size）来划分梯度。因此，如果您更改小批量大小，则不应始终更改学习率（LR）。（即如果你更改batch\_size则不应该同时更改learning rate）为了获得合适的LR，利用验证集是一种有效的方法。通常，训练开始时LR的典型值是0.1。在实践中，如果你看到你停止在验证集上取得进展，将LR除以2（或5），并继续前进，这可能会给你一个惊喜。

### 对预先训练的模型进行微调。

如今，许多着名的研究小组即[*Caffe Model Zoo*](https://github.com/BVLC/caffe/wiki/Model-Zoo)和[*VGG Group*](http://www.vlfeat.org/matconvnet/pretrained/)发布了许多先进的深度网络。借助预先训练好的深层模型的精彩泛化能力，您可以直接将这些预先训练好的模型用于您自己的应用程序。为了进一步提高数据集的分类性能，一种非常简单而有效的方法是根据您自己的数据对预先训练好的模型进行微调。如下表所示，两个最重要的因素是新数据集的大小（小或大）以及它与原始数据集的相似程度。不同的情况下可以使用不同的微调策略。例如，一个好例子就是您的新数据集与用于培训预先训练的模型的数据非常相似。在这种情况下，如果您的数据非常少，则只需对从预训练模型顶层提取的特征训练一个线性分类器即可。如果您手边有相当多的数据，请用小的学习率对一些预训练模型的顶层进行微调。但是，如果您自己的数据集与预先训练的模型中使用的数据完全不同，但是具有足够的训练图像，则应该对数据进行大量图层调整，并且只需少量学习率即可提高性能。但是，如果您的数据集不仅包含很少的数据，而且与预先训练的模型中使用的数据非常不同，那么您将遇到麻烦。由于数据有限，因此只训练线性分类器似乎更好。由于数据集非常不同，因此从网络顶部训练分类器可能不是最好的，它包含更多的数据集特定功能。

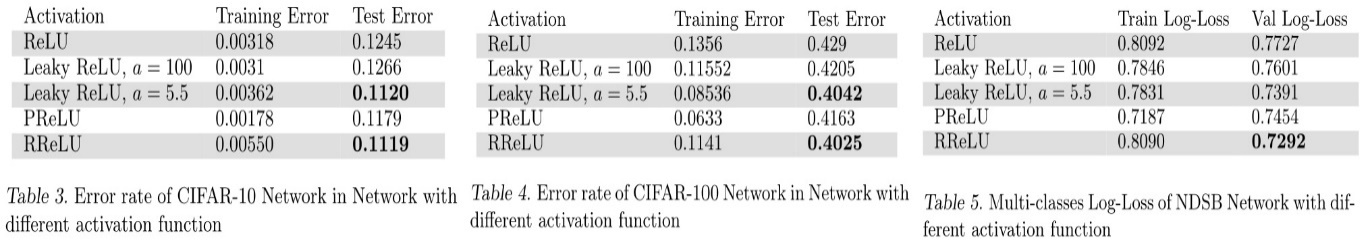
|  |
| --- |
| 表 |

在预先训练的模型上微调您的数据。微调的不同策略被用于不同的情况。对于数据集，Caltech-101与ImageNet相似，其中两个都是以对象为中心的图像数据集;而Place数据库与ImageNet不同，其中一个是以场景为中心的，另一个是以对象为中心的。

### 激活函数

也许激活函数也会带来一定的影响的，对于全正值的数据来说，是否需要使用变种ReLU做为神经网络的激活函数。

在[[5]中](http://arxiv.org/abs/1505.00853)，作者评估了*CIFAR-10*，*CIFAR-100*和*NDSB*数据集上具有不同激活函数的*两种现代* CNN架构的分类性能，如下表所示。*请注意，对于这两个网络来说，每个卷积层都有一个激活函数。这里的实际上表示的是ReLU公式中的1/*



从这些表格中，我们可以发现ReLU的性能不是所有三个数据集的最佳表现。对于Leaky ReLU，较大的斜率α将获得更好的准确率。PReLU很容易适用于小数据集（其训练错误最小，而测试错误不令人满意），但仍然优于ReLU。此外，RReLU明显优于NDSB上的其他激活功能，这表明RReLU可以克服过度拟合，因为该数据集的训练数据少于CIFAR-10 / CIFAR-100。***总之，在这三个数据集中，三种类型的ReLU变体都始终优于原始ReLU。PReLU和RRELU似乎是更好的选择。此外，何*等人*。也报道了类似的结论***[***[4]***](http://arxiv.org/abs/1502.01852)。

**[9]中斯坦福大学的很多学术报告可以供我参考学习，写出论文。**

[**http://cs231n.stanford.edu/reports/**](http://cs231n.stanford.edu/reports/)

# 2018.3.5

## 已完成任务

今天已经完成Tricks的全部搜集工作，在Tricks文件夹中已经收集了足够多的Tricks

## 未完成任务

1. 做一个excel表格，将自己的实验数据全部记录和填充进表格中。用以调试自己的神经网络。加上L2正则化和Dropout函数/以及将自己的实验数据保存起来，并将评价指标在训练集和测试集上的表现画在一张图片上。
2. 看看别人的实验数据和看论文，看看别人的实现。
3. 特别是Tensorflow例子上关于L2正则化/BN和Dropout的实现。

# 2018.3.6

## 已完成任务

《Tensorflow实践google深度学习框架》1,2,3章

# 2018.3.7

## 已完成任务

《Tensorflow实践google深度学习框架》4章

## 未完成任务

对程序进行BN处理（注意对测试数据集时的效果）

对程序进行L2

对程序进行滑动取平均

对程序进行Dropout

# 2018.3.8

## 已完成任务

1. 使用L2正则化对权值进行操作，减少测试集上的过拟合问题。
2. 使用滑动平均模型对所有可训练的变量进行操作。
3. **要学会使用《Tensorflow实战Google深度学习框架》中第五章各种图形的演变和解释。**

# 2018.3.9

## 已完成任务

1. 对变量名称管理的学习，通过定义不同的variable\_scope值可以定义相同的variable值。
2. 使用Saver类保存变量值进入日志文件。
3. 保存滑动平均值进入日志文件，并重命名滑动平均值。
4. **模型持久化**，从模型中提取出想要的变量的值的信息

## 未完成的任务

**我们应该将模型的搭建，训练过程，测试过程，分开进行，分为inference,train,test具有三个文件的框架结构**。

# 2018.3.11

1.《Tensorflow实战google深度学习框架》对数据集使用dropout《第六章》图像识别与卷积神经网络。

2.《Tensorflow实战google深度学习框架》输入图像数据进行标准化操作，或者BN算法？（如何实践BN算法）

3.adjusted = tf.image.per\_image\_whitening(image\_float)将代表一张图片的三维矩阵中的数字均值变为0，方差变为1。**（进行标准化操作）**

# 2018.3.12

## 已完成任务

1.**tf.train.string\_input\_producer**函数可以设置num\_epochs参数来限制加载初始文件列表的最大轮数。当所有文件都已经被使用设定的轮数后，可以继续尝试读取新的文件，输入队列会包OutOfRange的错误。在测试神经网络模型时，因为所有测试数据只需要使用一次，所以可以将num\_epochs参数设置为1.这样在计算完一轮之后程序将自动停止。

2**.min\_after\_dequeue**参数是tf.train\_shuffle\_batch函数特有的，min\_after\_dequeue参数限制了出队时队列中元素的最小个数来保证随机打乱顺序的作用，当出队函数被调用但是队列中元素不够时，出队操作将等待更多的元素入队才会完成。如果min\_after\_dequeue参数被设定，capacity也应该相应的调整来满足性能需求。

3.《Tensorflow实战google深度学习框架》一书第二版中加入了读取不同格式数据集的方式，可以根据这个写一篇类似的博文，方便以后使用。**Ch7 7.数据集基本使用方法**

**3.tensorboard可视化：内存，耗时，变量标准差**

**使用Tensorboard可以将变量的值以CSV文件的形式保存下来。**

## 未完成任务

1.先不要学习其他的，先把论文弄完，现在最要做的是看BN的tensorflow实现。

2.将自己的论文项目托管到云上

# 2018.3.16

## 已完成任务

1.使用tqdm函数库能够产生进度条的效果

## 未完成任务

1. tf.layers.batch\_normalization各参数的意义（要不然看程序看的我一脸懵逼）

2 有可能是tf.lrn局部标准化将一个batch中的数据全部标准化处理，而没有对测试数据做出相应的改进，而导致一个批次数据的输出完全相同。

# 2018.3.18

## 已完成任务

1. 看完Udacity上关于BN的教程https://github.com/udacity/deep-learning/tree/master/batch-norm

## 未完成任务

1. 整合代码
2. 应用代码
3. 看相关论文
4. 论文初稿
5. **明天要不要放松一下？23333333333**

# 2018.3.19

## 已完成任务

## 未完成任务

1. 今天坐在实验室都有点迷茫，比如我现在到底应该做什么，怎么做，这的确是一个很复杂而又相当尴尬的问题。
2. 准确来说，自己前半个月看的代码几乎都忘光了。**现在从头开始看看自己的代码和笔记。**