Neural network

   
将各个参数的设置部分集中在一起。如果参数的设置分布在代码的各个地方，那么修改的过程想必会非常痛苦。

 可以输出模型的损失函数值以及训练集和验证集上的准确率。

 可以考虑设计一个子程序，可以根据给定的参数，启动训练并监控和周期性保存评估结果。再由一个主程序，分配参数以及并行启动一系列子程序。

**画图**

画图是一个很好的习惯，一般是训练数据遍历一轮以后，就输出一下训练集和验证集准确率。同时画到一张图上。这样训练一段时间以后，如果模型一直没有收敛，那么就可以停止训练，尝试其他参数了，以节省时间。  
如果训练到最后，训练集，测试集准确率都很低，那么说明模型有可能欠拟合。那么后续调节参数方向，就是增强模型的拟合能力。例如增加网络层数，增加节点数，减少dropout值，减少L2正则值等等。  
如果训练集准确率较高，测试集准确率比较低，那么模型有可能过拟合，这个时候就需要向提高模型泛化能力的方向，调节参数。

# 从粗到细分阶段调参

实践中，一般先进行初步范围搜索，然后根据好结果出现的地方，再缩小范围进行更精细的搜索。

1. 建议先参考相关论文，以论文中给出的参数作为初始参数。至少论文中的参数，是个不差的结果。
2. 如果找不到参考，那么只能自己尝试了。可以先从比较重要，对实验结果影响比较大的参数开始，同时固定其他参数，得到一个差不多的结果以后，在这个结果的基础上，再调其他参数。例如学习率一般就比正则值，dropout值重要的话，学习率设置的不合适，不仅结果可能变差，模型甚至会无法收敛。
3. 如果实在找不到一组参数，可以让模型收敛。那么就需要检查，是不是其他地方出了问题，例如模型实现，数据等等。可以参考我写的[深度学习网络调试技巧](https://zhuanlan.zhihu.com/p/20792837)

**提高速度**

调参只是为了寻找合适的参数，而不是产出最终模型。一般在小数据集上合适的参数，在大数据集上效果也不会太差。因此可以尝试对数据进行精简，以提高速度，在有限的时间内可以尝试更多参数。

* 对训练数据进行采样。例如原来100W条数据，先采样成1W，进行实验看看。
* 减少训练类别。例如手写数字识别任务，原来是10个类别，那么我们可以先在2个类别上训练，看看结果如何。

# 超参数范围

建议优先在对数尺度上进行超参数搜索。比较典型的是学习率和正则化项，我们可以从诸如0.001 0.01 0.1 1 10，以10为阶数进行尝试。因为他们对训练的影响是相乘的效果。不过有些参数，还是建议在原始尺度上进行搜索，例如dropout值: 0.3 0.5 0.7)。

**经验参数**

这里给出一些参数的经验值，避免大家调参的时候，毫无头绪。

* learning rate: 1 0.1 0.01 0.001, 一般从1开始尝试。很少见learning rate大于10的。学习率一般要随着训练进行衰减。衰减系数一般是0.5。 衰减时机，可以是验证集准确率不再上升时，或固定训练多少个周期以后。  
  不过更建议使用自适应梯度的办法，例如adam,adadelta,rmsprop等，这些一般使用相关论文提供的默认值即可，可以避免再费劲调节学习率。对RNN来说，有个经验，如果RNN要处理的序列比较长，或者RNN层数比较多，那么learning rate一般小一些比较好，否则有可能出现结果不收敛，甚至Nan等问题。
* 网络层数： 先从1层开始。
* 每层结点数： 16 32 128，超过1000的情况比较少见。超过1W的从来没有见过。
* batch size: 128上下开始。batch size值增加，的确能提高训练速度。但是有可能收敛结果变差。如果显存大小允许，可以考虑从一个比较大的值开始尝试。因为batch size太大，一般不会对结果有太大的影响，而batch size太小的话，结果有可能很差。
* clip c(梯度裁剪): 限制最大梯度,其实是value = sqrt(w1^2+w2^2….),如果value超过了阈值，就算一个衰减系系数,让value的值等于阈值: 5,10,15
* dropout： 0.5
* L2正则：1.0，超过10的很少见。
* 词向量embedding大小：128，256
* 正负样本比例： 这个是非常忽视，但是在很多分类问题上，又非常重要的参数。很多人往往习惯使用训练数据中默认的正负类别比例，当训练数据非常不平衡的时候，模型很有可能会偏向数目较大的类别，从而影响最终训练结果。除了尝试训练数据默认的正负类别比例之外，建议对数目较小的样本做过采样，例如进行复制。提高他们的比例，看看效果如何，这个对多分类问题同样适用。  
  在使用mini-batch方法进行训练的时候，尽量让一个batch内，各类别的比例平衡，这个在图像识别等多分类任务上非常重要。

**自动调参**

人工一直盯着实验，毕竟太累。自动调参当前也有不少研究。下面介绍几种比较实用的办法：

* Gird Search. 这个是最常见的。具体说，就是每种参数确定好几个要尝试的值，然后像一个网格一样，把所有参数值的组合遍历一下。优点是实现简单暴力，如果能全部遍历的话，结果比较可靠。缺点是太费时间了，特别像神经网络，一般尝试不了太多的参数组合。
* Random Search。Bengio在[Random Search for Hyper-Parameter Optimization](https://link.zhihu.com/?target=http%3A//www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf" \t "_blank)中指出，Random Search比Gird Search更有效。实际操作的时候，一般也是先用Gird Search的方法，得到所有候选参数，然后每次从中随机选择进行训练。
* Bayesian Optimization. 贝叶斯优化，考虑到了不同参数对应的实验结果值，因此更节省时间。和网络搜索相比简直就是老牛和跑车的区别。具体原理可以参考这个论文： [Practical Bayesian Optimization of Machine Learning Algorithms](https://link.zhihu.com/?target=http%3A//papers.nips.cc/paper/4522-practical-bayesian-optimization-of-machine-learning-algorithms.pdf) ，这里同时推荐两个实现了贝叶斯调参的Python库，可以上手即用：
  + [jaberg/hyperopt](https://link.zhihu.com/?target=https%3A//github.com/jaberg/hyperopt), 比较简单。
  + [fmfn/BayesianOptimization](https://link.zhihu.com/?target=https%3A//github.com/fmfn/BayesianOptimization)， 比较复杂，支持并行调参。

Initializer - glorot\_uniform\_initializer，也就是大家经常说的无脑使用xavier

Uniform

## 训练调参经验

介绍完BaseLine，接下来就来看看作者的优化方法。论文从加快模型训练，网络结构优化以及训练参数调优三个部分分别介绍如何提升模型的效果。

## 3.1 模型训练加速

关于模型训练加速，论文提到了2点，一是**使用更大的Batch Size**，二是**使用低精度(如FP16)进行训练**（也是我们常说的混合精度训练）。关于使用更大的Batch Size进行训练加速，作者指出一般只增加Batch Size的话，效果不会太理想，例如FaceBook这篇大名鼎鼎的论文有证明：

**Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour**

然后本文总结了几个重要要的调参方案，如下：

* **增大学习率**。因为更大的Batch Size意味着每个Batch数据计算得到的梯度更加贴近整个数据集，从数学上来说就是方差更小，因此当更新方向更加准确之后，迈的步子也可以更大，一般来说Batch Size变成原始几倍，学习率就增加几倍。
* **Warm up**。Warm up指的是用一个小的学习率先训练几个epoch，这是因为网络的参数是随机初始化的，假如一开始就采用较大的学习率容易出现数值不稳定，这也是为什么要使用Warm up。然后等到训练过程基本上稳定了就可以使用原始的初始学习率进行训练了。作者在使用Warm up的过程中使用线性增加的策略。举个例子假如Warm up阶段的初始学习率是0，warmup阶段共需要训练m个batch的数据（论文实现中m个batch共5个epoch），假设训练阶段的初始学习率是L，那么在第个batch的学习率就设置为。
* **每一个残差块后的最后一个BN层的参数初始化为0**。我们知道BN层的，参数是用来对标注化后的数据做线性变换的，公式表示为：，其中我们一般会把设为1，而这篇论文提出初始化为则更容易训练。
* **不对Bias参数做权重惩罚**。但是对权重还是要做的。。

接下来作者提到了使用低精度(16-Bit浮点型)来做训练加速，也即是我们常说的混合精度训练。但不是所有的NVIDIA GPU都支持FP16，我大概只知道V100和2080 Ti是支持混合精度训练的。

作者将上面的Tricks结合在一起进行训练，**下面的Table3展示了使用更大的Batch Size和16位浮点型进行训练的结果，可以看到这俩Tricks相比于BaseLine训练速度提升了许多，并且精度也更好了。**

**参数初始化。**

下面几种方式,随便选一个,结果基本都差不多。但是一定要做。否则可能会减慢收敛速度，影响收敛结果，甚至造成Nan等一系列问题。  
n\_in为网络的输入大小，n\_out为网络的输出大小，n为n\_in或(n\_in+n\_out)\*0.5  
Xavier初始法论文：[http://jmlr.org/proceedings/papers/v9/glorot10a/glorot10a.pdf](https://link.zhihu.com/?target=http%3A//jmlr.org/proceedings/papers/v9/glorot10a/glorot10a.pdf" \t "_blank)  
He初始化论文：[https://arxiv.org/abs/1502.01852](https://link.zhihu.com/?target=https%3A//arxiv.org/abs/1502.01852" \t "_blank)

* uniform均匀分布初始化：  
  w = np.random.uniform(low=-scale, high=scale, size=[n\_in,n\_out])
  + Xavier初始法，适用于普通激活函数(tanh,sigmoid)：scale = np.sqrt(3/n)
  + He初始化，适用于ReLU：scale = np.sqrt(6/n)
* normal高斯分布初始化：  
  w = np.random.randn(n\_in,n\_out) \* stdev # stdev为高斯分布的标准差，均值设为0
  + Xavier初始法，适用于普通激活函数 (tanh,sigmoid)：stdev = np.sqrt(n)
  + He初始化，适用于ReLU：stdev = np.sqrt(2/n)
* svd初始化：对RNN有比较好的效果。参考论文：[https://arxiv.org/abs/1312.6120](https://link.zhihu.com/?target=https%3A//arxiv.org/abs/1312.6120" \t "_blank)

**数据预处理方式**

* zero-center ,这个挺常用的.  
  X -= np.mean(X, axis = 0) # zero-center  
  X /= np.std(X, axis = 0) # normalize
* PCA whitening,这个用的比较少.

**训练技巧**

* 要做梯度归一化,即算出来的梯度除以minibatch size
* clip c(梯度裁剪): 限制最大梯度,其实是value = sqrt(w1^2+w2^2….),如果value超过了阈值,就算一个衰减系系数,让value的值等于阈值: 5,10,15
* dropout对小数据防止过拟合有很好的效果,值一般设为0.5,小数据上dropout+sgd在我的大部分实验中，效果提升都非常明显.因此可能的话，建议一定要尝试一下。 dropout的位置比较有讲究, 对于RNN,建议放到输入->RNN与RNN->输出的位置.关于RNN如何用dropout,可以参考这篇论文:[http://arxiv.org/abs/1409.2329](https://link.zhihu.com/?target=http%3A//arxiv.org/abs/1409.2329)
* adam,adadelta等,在小数据上,我这里实验的效果不如sgd, sgd收敛速度会慢一些，但是最终收敛后的结果，一般都比较好。如果使用sgd的话,可以选择从1.0或者0.1的学习率开始,隔一段时间,在验证集上检查一下,如果cost没有下降,就对学习率减半. 我看过很多论文都这么搞,我自己实验的结果也很好. 当然,也可以先用ada系列先跑,最后快收敛的时候,更换成sgd继续训练.同样也会有提升.据说adadelta一般在分类问题上效果比较好，adam在生成问题上效果比较好。
* 除了gate之类的地方,需要把输出限制成0-1之外,尽量不要用sigmoid,可以用tanh或者relu之类的激活函数.1. sigmoid函数在-4到4的区间里，才有较大的梯度。之外的区间，梯度接近0，很容易造成梯度消失问题。2. 输入0均值，sigmoid函数的输出不是0均值的。
* rnn的dim和embdding size,一般从128上下开始调整. batch size,一般从128左右开始调整.batch size合适最重要,并不是越大越好.
* word2vec初始化,在小数据上,不仅可以有效提高收敛速度,也可以可以提高结果.
* 尽量对数据做shuffle
* LSTM 的forget gate的bias,用1.0或者更大的值做初始化,可以取得更好的结果,来自这篇论文:[http://jmlr.org/proceedings/papers/v37/jozefowicz15.pdf](https://link.zhihu.com/?target=http%3A//jmlr.org/proceedings/papers/v37/jozefowicz15.pdf), 我这里实验设成1.0,可以提高收敛速度.实际使用中,不同的任务,可能需要尝试不同的值.
* Batch Normalization据说可以提升效果，不过我没有尝试过，建议作为最后提升模型的手段，参考论文：Accelerating Deep Network Training by Reducing Internal Covariate Shift
* 如果你的模型包含全连接层（MLP），并且输入和输出大小一样，可以考虑将MLP替换成Highway Network,我尝试对结果有一点提升，建议作为最后提升模型的手段，原理很简单，就是给输出加了一个gate来控制信息的流动，详细介绍请参考论文: [http://arxiv.org/abs/1505.00387](https://link.zhihu.com/?target=http%3A//arxiv.org/abs/1505.00387)
* 来自@张馨宇的技巧：一轮加正则，一轮不加正则，反复进行。

**Ensemble**

Ensemble是论文刷结果的终极核武器,深度学习中一般有以下几种方式

* 同样的参数,不同的初始化方式
* 不同的参数,通过cross-validation,选取最好的几组
* 同样的参数,模型训练的不同阶段，即不同迭代次数的模型。
* 不同的模型,进行线性融合. 例如RNN和传统模型.