1. 不管什么模型，先在一个较小的训练集上train和test，看看它能不能过拟合。如果不能过拟合，可能是学习率太大，或者代码写错了。先调小学习率试一下，如果还不行就去检查代码，先看dataloader输出的数据对不对，再看模型每一步的size是否符合自己期待。
2. 看train/eval的loss曲线，正常的情况应该是train loss呈log状一直下降最后趋于稳定，eval loss开始时一直下降到某一个epoch之后开始趋于稳定或开始上升，这时候可以用early stopping保存eval loss最低的那个模型。如果loss曲线非常不正常，很有可能是数据处理出了问题，比如label对应错了，回去检查代码。
3. 不要一开始就用大数据集，先在一个大概2w训练集，2k测试集的小数据集上调参。
4. 尽量不要自己从头搭架子（新手和半新手）。找一个已经明确没有bug能跑通的其它任务的架子，在它的基础上修改。否则debug过程非常艰难，因为有时候是版本迭代产生的问题，修改起来很麻烦。
5. 优化器优先用adam，学习率设1e-3或1e-4，再试Radam（LiyuanLucasLiu/RAdam）。不推荐sgdm，因为很慢。
6. lrscheduler用torch.optim.lr\_scheduler.CosineAnnealingLR，T\_max设32或64，几个任务上试效果都不错。（用这个lr\_scheduler加上adam系的optimizer基本就不用怎么调学习率了）
7. 有一些任务（尤其是有RNN的）要做梯度裁剪，torch.nn.utils.clip\_grad\_norm。
8. 参数初始化，lstm的h用orthogonal，其它用he或xavier。
9. 激活函数用relu一般就够了，也可以试试leaky relu。
10. batchnorm和dropout可以试，放的位置很重要。优先尝试放在最后输出层之前，以及embedding层之后。RNN可以试layer\_norm。有些任务上加了这些层可能会有负作用。
11. metric learning中先试标label的分类方法。然后可以用triplet loss，margin这个参数的设置很重要。
12. batchsize设置小一点通常会有一些提升，某些任务batchsize设成1有奇效。
13. embedding层的embedsize可以小一些（64 or 128），之后LSTM或CNN的hiddensize要稍微大一些（256 or 512）。（ALBERT论文里面大概也是这个意思）
14. 模型方面，可以先用2或3层LSTM试一下，通常效果都不错。
15. weight decay可以试一下，我一般用1e-4。
16. 有CNN的地方就用shortcut。CNN层数加到某一个值之后对结果影响就不大了，这个值作为参数可以调一下。
17. GRU和LSTM在大部分任务上效果差不多。
18. 看论文时候不要全信，能复现的尽量复现一下，许多论文都会做低baseline，但实际使用时很多baseline效果很不错。
19. 对于大多数任务，数据比模型重要。面对新任务时先分析数据，再根据数据设计模型，并决定各个参数。例如nlp有些任务中的padding长度，通常需要达到数据集的90%以上，可用pandas的describe函数进行分析。

想到其它的继续加。

觉得对你有用的话就点个赞吧~ 有具体问题的同学可以在评论区留言（或私信我），有能力的我都会回答。也可以翻翻我对其它问题的回答，可能也有一点帮助。

**作者：萧瑟**

**https://www.zhihu.com/question/41631631/answer/94816420**

训练技巧对深度学习来说是非常重要的，作为一门实验性质很强的科学，同样的网络结构使用不同的训练方法训练，结果可能会有很大的差异。这里我总结了近一年来的炼丹心得，分享给大家，也欢迎大家补充指正。

**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。

下面的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

He初始化论文：https://arxiv.org/abs/1502.01852

* 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

**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
* 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, 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
* 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.

**作者：Towserhttps://www.zhihu.com/question/41631631/answer/862075836**

总结一下我遇到的小朋友常犯的错：

1、一上来就自己动手写模型。建议首先用成熟的开源项目及其默认配置（例如 Gluon 对经典模型的各种复现、各个著名模型作者自己放出来的代码仓库）在自己的数据集上跑一遍，在等程序运行结束的时间里仔细研究一下代码里的各种细节，最后再自己写或者改代码。

2、训 RNN 不加 gradient clipping，导致训练一段时间以后 loss 突然变成 Nan。

3、tying input & output embedding（就是词向量层和输出 softmax 前的矩阵共享参数，在语言模型或机器翻译中常用）时学习率需要设置得非常小，不然容易 Nan。

4、在数据集很大的情况下，一上来就跑全量数据。建议先用 1/100、1/10 的数据跑一跑，对模型性能和训练时间有个底，外推一下全量数据到底需要跑多久。在没有足够的信心前不做大规模实验。

5、只喜欢漂亮的模型结构，瞧不起调参数的论文/实验报告，看论文时经常不看超参数设置等细节。举个例子，现在还有相当多的人不知道 BERT 的激活函数是 GELU 而不是 transformer 原论文中的 ReLU（虽然我觉得这一点很无厘头）。在自己没有太多资源实验的情况下，实验报告类文章简直是业界良心好不好！

NLP 领域主要推荐以下几篇：

Regularizing and Optimizing LSTM Language Models（LSTM 的训练技巧）

Massive Exploration of Neural Machine Translation Architectures（NMT 里各个超参的影响）

Training Tips for the Transformer Model（训练 Transformer 时会发生的各种现象）

RoBERTa: A Robustly Optimized BERT Pretraining Approach（BERT 预训练技巧，虽然跟大部分人没啥关系）

CV 我不算太熟，不过也可以勉强推荐几篇：

Training ImageNet in 1 Hour（大批量训练技巧）

Bag of Tricks for Image Classification with Convolutional Neural Networks（各种训练技巧集大成）

Bag of Freebies for Training Object Detection Neural Networks（同上）

EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks（当前对参数利用最有效的 CNN，类似地还有一篇 EfficientDet）

6、初始学习率：

有时受 batch size、sequence length 各种因素的影响，loss 很大（比如说好几万），对于这种数字人是没有数感的，建议首先计算一下 per token loss（如果是多任务，可以每个任务单独算；类似地，某些 CV 任务可以计算 per pixel loss），心里有点感觉。脱离损失函数的形式谈学习率没有意义（例如单是对 batch size 求和或者取平均这个差异就会使梯度差成百上千倍）。

在确定初始学习率的时候，从一个很小的值（例如 1e-7）开始，然后每一步指数增大学习率（例如扩大1.05 倍）进行训练。训练几百步应该能观察到损失函数随训练步数呈对勾形，选择损失下降最快那一段的学习率即可。

7、Adam 可以解决一堆奇奇怪怪的问题（有时 loss 降不下去，换 Adam 瞬间就好了），也可以带来一堆奇奇怪怪的问题（比如单词词频差异很大，当前 batch 没有的单词的词向量也被更新；再比如Adam和L2正则结合产生的复杂效果）。用的时候要胆大心细，万一遇到问题找各种魔改 Adam（比如 MaskedAdam, AdamW 啥的）抢救。

8、subword 总是会很稳定地涨点，只管用就对了。

9、GPU 上报错时尽量放在 CPU 上重跑，错误信息更友好。例如 GPU 报 "ERROR:tensorflow:Model diverged with loss = NaN" 其实很有可能是输入 ID 超出了 softmax 词表的范围。

**10、要有耐心**！

这一条放在最后，是因为很多人不把它当一回事儿。可能是觉得这一条不需要写代码所以不重要？我见过太多人因为这条浪费时间了，所以专门强调一下。

有些指标是有滞后性的，需要等训练一段时间才开始动（例如 BN 训练和推断行为不一致，可能训练的准确率已经很高了，但测试准确率很低，这是因为 BN 的统计量还不准，再等等就好了）。很多人训练几步看没什么效果就把程序停掉开始 debug 了，但其实代码毫无问题。如此反复好几天甚至一两周都在原地踏步，其实需要做的仅仅是让程序自个儿安安静静地跑上几个小时或者一天……

**作者：京东白条https://www.zhihu.com/question/41631631/answer/776852832**

相信很多刚开始接触深度学习朋友，会感觉深度学习调参就像玄学一般，有时候参数调的好，模型会快速收敛，参数没调好，可能迭代几次loss值就直接变成Nan了。

记得刚开始研究深度学习时，做过两个小例子。一个是用tensorflow构建了一个十分简单的只有一个输入层和一个softmax输出层的Mnist手写识别网络，第一次我对权重矩阵W和偏置b采用的是正态分布初始化，一共迭代了20个epoch，当迭代完第一个epoch时，预测的准确度只有10%左右（和随机猜一样，Mnist是一个十分类问题），当迭代完二十个epoch，精度也仅仅达到了60%的样子。然后我仅仅是将权重矩阵W初始化方法改成了全为0的初始化，其他的参数均保持不变，结果在训练完第一个epoch后预测精度就达到了85%以上，最终20个epoch后精度达到92%。另一个例子是回归问题的预测，当时采用的SGD优化器，一开始学习率设定的0.1，模型可以正常训练，只是训练速度有些慢，我试着将学习率调整到0.3，希望可以加速训练速度，结果没迭代几轮loss就变成Nan了。于是从那时起我就深刻的感受到参数调节在深度学习模型训练中的重要意义。

其实上述问题产生的原因也很好理解，对于参数初始化，因为我们学习的本来就是权重W与偏置b，如果初始化足够好，直接就初始化到最优解，那都不用进行训练了。良好的初始化，可以让参数更接近最优解，这可以大大提高收敛速度，也可以防止落入局部极小。对于学习率，学习率如果取太大，会使模型训练非常震荡，可以想象我们最小化一个二次抛物线，选取一个很大的学习率，那么迭代点会一直在抛物线的两边震荡，收敛不到最小值，甚至还有螺旋上升迭代点的可能。

下面对深度学习调参技巧谈些心得，虽说不能让你通过以下阅读成为一个调参高手，但最起码可以提供一些调参的思路。

**1.** **激活函数选择：**

常用的激活函数有relu、leaky-relu、sigmoid、tanh等。对于输出层，多分类任务选用softmax输出，二分类任务选用sigmoid输出，回归任务选用线性输出。而对于中间隐层，则优先选择relu激活函数（relu激活函数可以有效的解决sigmoid和tanh出现的梯度弥散问题，多次实验表明它会比其他激活函数以更快的速度收敛）。另外，构建序列神经网络（RNN）时要优先选用tanh激活函数。

**2、学习率设定：**

一般学习率从0.1或0.01开始尝试。学习率设置太大会导致训练十分不稳定，甚至出现Nan，设置太小会导致损失下降太慢。学习率一般要随着训练进行衰减。衰减系数设0.1，0.3，0.5均可，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自动进行衰减。

**3、防止过拟合：**

一般常用的防止过拟合方法有使用L1正则项、L2正则项、dropout、提前终止、数据集扩充等。如果模型在训练集上表现比较好但在测试集上表现欠佳可以选择增大L1或L2正则的惩罚力度（L2正则经验上首选1.0，超过10很少见），或增大dropout的随机失活概率（经验首选0.5）；或者当随着训练的持续在测试集上不增反降时，使用提前终止训练的方法。当然最有效的还是增大训练集的规模，实在难以获得新数据也可以使用数据集增强的方法，比如CV任务可以对数据集进行裁剪、翻转、平移等方法进行数据集增强，这种方法往往都会提高最后模型的测试精度。

**4、优化器选择：**

如果数据是稀疏的，就用自适应方法，即 Adagrad, Adadelta, RMSprop, Adam。整体来讲，Adam 是最好的选择。SGD 虽然能达到极小值，但是比其它算法用的时间长，而且可能会被困在鞍点。如果需要更快的收敛，或者是训练更深更复杂的神经网络，需要用一种自适应的算法。

**5、残差块与BN层：**

如果你希望训练一个更深更复杂的网络，那么残差块绝对是一个重要的组件，它可以让你的网络训练的更深。

BN层具有加速训练速度，有效防止梯度消失与梯度爆炸，具有防止过拟合的效果，所以构建网络时最好要加上这个组件。

**6.自动调参方法：**

（1）Grid Search：网格搜索，在所有候选的参数选择中，通过循环遍历，尝试每一种可能性，表现最好的参数就是最终的结果。其原理就像是在数组里找最大值。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。

（2）Random Search：经验上，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。另外Random Search往往会和由粗到细的调参策略结合使用，即在效果比较好的参数附近进行更加精细的搜索。

（3）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，贝叶斯调参比Grid Search迭代次数少，  速度快；而且其针对非凸问题依然稳健。

**7.参数随机初始化与数据预处理：**

参数初始化很重要，它决定了模型的训练速度与是否可以躲开局部极小。relu激活函数初始化推荐使用He normal，tanh初始化推荐使用Glorot normal，其中Glorot normal也称作Xavier normal初始化；数据预处理方法一般也就采用数据归一化即可。