知乎\_深度学习调参

作者：Miracle  
链接：<https://www.zhihu.com/question/29641737/answer/243982984>

# 网易工程师

现在不管是图像分类还是目标检测还是语意分割，模型方面基本上都做得非常好了，因此真正需要调的参数其实并不多了，或者说调很多参数实际带来的提升都非常小了。不过有几个参数还是非常重要的，根据我的经验，按重要程度来说吧，希望能给需要的人提供帮助：

**1、学习率**

**初始学习率很重要，学习率的变化策略也很重要**，你真正跑过数据就会发现其中的奥秘，适当时候可微调，一般在0.1到0.00001范围，根据具体情况定。

**2、batch size**

我们用的随机梯度下降是建立在batch基础上的，合适的batch size对你模型的优化是比较重要的，这个参数倒不需要微调，在一个大致数量即可，常取2的n次方，太大的batch size会受GPU显存的限制，所以不能无限增大。

**3、其他参数**

比如**L1，L2正则化**的参数，也就是很多深度学习框架里面的wd参数，一般默认是0.0001，调整正则化的参数可以根据模型表现来，过拟合的时候可以适当加大系数，非过拟合的时候可不调这个参数，毕竟跑一次模型得花不少时间。**epoch**，应该要和模型表现搭配，如果你的模型已经过拟合了，你就没必要继续跑了；相反，如果你的epoch太小，你epoch跑完了，模型的loss还在下降，模型还在优化，那么你这个epoch就太小了，应该增加。至于**卷积核个数，模型层数**等个人认为看情况而定，现在的优秀模型都是很多前沿的研究者经过理论分析和无数次的实验得到的，一般不会说你调一些卷积核个数或其他一些层结构就会有明显提升，当然有时间你可以试试

# Andrew Ng

的个人经验和偏好是：

第一梯队： learning rate α

第二梯队： hidden units

mini-batch size

momentum β

第三梯队： number of layers

learning rate decay

other optimizer hyperparameters

# 知乎推荐

作者：Sunwalker  
链接：https://www.zhihu.com/question/29641737/answer/276342696

本文主要针对深度神经网络训练和调试过程中，从工程的角度对一些常见优化和调试方法进行探讨。

目前谷歌开源的Tensorflow是一个目前比较常用流行的的深度学习框架，他采用了一种计算图模型的思想，对搭建好的结构可以用TensorBoard进行展示。除此以外Keras 可以理解为是一个在 TensorFlow 顶部的layer，它可以让模型实现起来变得更加简单。

在用深度学习算法实现一个新应用的时候，比较烦的一点有着很多的超参数和较多优化方法可选择，所以需要尝试多次不同的可能性来找到在该应用中的最匹配结果。这也是值得讨论探讨的地方。

在训练一个深度学习模型的过程中，超参数选择什么最优，这是一个基于实验和经验的过程。需要不停的尝试，直到找到合适的参数值。

**1、一些术语的区分和定义**

**超参数**：如学习率α、adam方法的β1和β2参数、网络层数、每层隐藏单元数、学习率衰减值、mini-batch大小等需要人工选取的参数。

**参数**：这里所指的参数另外一种说法就是权重，需要被训练的参数。比如：W\*X+b，这里W、b就是参数。

**2、模型调试的技巧：**

**2.1 使用随机参数调参**

Bengio在[http://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf](https://link.zhihu.com/?target=http%3A//www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf)文章中指出，Random Search比Gird  
Search更有效。

网格搜索主要适合与超参数的数量较少时的情况。但是在深度学习等超参数较多的情况中，应该选用随机实验参数来效率更高。

**2.2、由粗到细：**

遵循从粗糙到精细的搜索过程。比如两个参数的情况，首先对整个由两个随机参数组成的多个点划定的区域进行粗略搜索，然后不断缩小搜索区域，再在这个区域中不断精细调节参数。

**2.3、超参数选取尺度的考虑：**

每一种超参数考虑的尺度是不一样的，比如隐藏层数、隐藏层单元数肯定是以线性尺度进行搜索。而如果是学习率α，就需要取对数了，因为学习率有可能从0.00001到0.1之间，因此选用对数搜索尺度更加合理。

**2.4、学习率的选择：**

参数更新时的学习率选择，需要根据尝试不同的学习率来确定。可以画出损失函数随迭代次数的收敛图，选取学习率能使得代价函数收敛到最低的位置

**2.5、泛化的问题：**

不需要一上来就考虑泛化的问题。先要保证模型的复杂度能够充分拟合数据，代价函数能够快速的下降，在训练集上达到一个非常高的准确率，再考虑泛化的问题。

**3、** **激活函数的选择：**

神经网络目前效果随着网络的深度不断增加，效果也不断提升，一个重要原因就是通过非线性的变换。如果没有非线性变换的激活函数，只是做多层的线性变换，实际上效果是跟单个函数是一样的。

下面对各种激活函数进行介绍：

**Sigmoid函数**：

Sigmoid函数是以前神经网络中常用的一种激活函数。目前一般不用它，原因如下：

1、该函数输出当X稍大或稍小的时候要么非常接近0，要么非常接近1。会导致梯度弥散（反向传播算法中出现梯度趋近0的问题）。这会导致训练效率变的很差。

2、构造比较复杂，所以比较消耗系统资源。

3、sigmoid(X)均是大于0的数，输出全部是正值，输出的中心值趋近于0.5而不是0，这不利于下一层的学习。

**Tanh函数:**

表现形式如下

Tanh函数是sigmoid函数的改进版，它解决了sigmoid(X)的值均为正数的缺点，使得输出的中心值趋近于0，而不是0.5这让下一层的学习更方便一点。缺点是还不能解决梯度弥散的问题。

**ReLU函数：**

**表现形式如下**

最近ReLu函数非常流行。ReLu函数公式为max(0,x)，它相比其它激活函数非常简洁。

[http://www.cs.toronto.edu/~fritz/absps/imagenet.pdf](https://link.zhihu.com/?target=http%3A//www.cs.toronto.edu/%7Efritz/absps/imagenet.pdf)

在这篇关于卷积神经网络的中指出使用ReLu函数，而不使用sigmoid函数或者tanh函数可以使你的网络收敛的更快。

该文章指出，和tanh函数相比，通过ReLu函数训练卷积神经网在他们的实验中达到了之前六倍的收敛速度。

**Leaky Relu:**

该函数保留了Relu的特性，并优化了ReLu函数，但并不一定任何时候比Relu好，目前有争议。

**4、欠拟合和过度拟合的情况：**

初始训练完模型之后，首先要知道算法的偏差高不高。如果偏差高，出现欠拟合，甚至无法拟合训练集，就需要增加神经网络层数和大小，需要让网络变得更复杂来充分拟合现有的数据。

如果方差高，出现过度拟合，最好的办法是寻找更多的数据，或者使用正则化的方法。

下面介绍深度学习中正则化的一些方法：

**2.1、增加惩罚项：**

我们知道，在传统的回归、分类模型中，为了防止过度拟合，会在代价函数（cost function）加入惩罚项来防止过度拟合，其中有L1（Lasso回归），L2（岭回归）等方法。同样的在深度学习的由softmax产生的代价函数也可以用L1、L2等惩罚项的方法来防止过度拟合。

**2.2、droupout 方法：**

Droupout方法

**原理：**

Droupout方法是Srivastava et al发表的关于防止过度拟合的方法。

论文链接：[http://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf](https://link.zhihu.com/?target=http%3A//www.cs.toronto.edu/%7Ersalakhu/papers/srivastava14a.pdf)

该方法就是随机让一些神经元失活，随机保留部分神经元。类似于从总体神经元中随机抽样。

这样每次在训练模型都会不一样，这类似于一种集成模型的思想。

对于RNN网路如何放置droupout参见这篇文章：[https://arxiv.org/abs/1409.2329](https://link.zhihu.com/?target=https%3A//arxiv.org/abs/1409.2329)

**使用技巧：**

随机失活率一般设置为0.5。当然每一层的神经网络设置的失活率都可以设置成不一样的，一般方法是对越大越复杂的隐藏层，意味着过度拟合的可能性越大，所以失活率设置越高，保留率要设置得越低。相反对于简单的网络层，保留率设置越高，或者不用droupout。需要注意的是，除非算法过拟合，不然最好不需要用droupout等方法。

使用droupout有个比较烦的点就是，损失函数不再被明确定义，因为每一次迭代所用的网络都不同。所以失去一种通过绘制（迭代次数-损失函数值）曲线图来调试的工具。在这种情况下，通常是先关闭droupout，并设定keep.prop值为1，运行代码确保损失函数单调递减，然后再打开droupout功能。

**5、数据预处理：**

对输入数据进行归一化处理，能够显著加速梯度下降的收敛过程。至于为什么有效，原因见下图：

可以看到如果不归一化，输入数据每个维度的尺度范围的差异会对梯度下降的迭代过程造成影响。

**6、权重初始化：**

目前有很多关于如何做权重参数初始化的研究。因为深度学习权重初始化很重要，如果有问题就不会有好结果。这是一个非常重要的问题。

如果权值初始化为0的话，用梯度下降算法，那会完全失效。因为如果权重初始化为0，每个神经元将会输出同样的结果，方向传播时就会计算出同样的梯度，最后会得到完全相同的参数更新，所以算法失效。

如果用很小的随机数值初始化。比如用高斯分布乘以一个很小的常数进行初始化：W = 0.01 \* np.random.randn()。对于层数较少的神经网络效果很好，但是随着层数的增加，对于初始化更为敏感。

实验表明随着隐含层隐藏层的增加，前面的层还是服从高斯分布的，但是越到后面的隐藏层隐含层输出值的分布图会趋近于0。

Glorot的文章[http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf](https://link.zhihu.com/?target=http%3A//proceedings.mlr.press/v9/glorot10a/glorot10a.pdf)这篇论文中，作者推荐的初始化方式是W=2/(n1+n2)，其中n1是前一层的单元个数，n2是后一层的单元个数。Tensorflow的tf.contrib.layers.xavier\_initializer()函数即使用的该方法。

最近Kai minghe发表的文章[https://arxiv-web3.library.cornell.edu/abs/1502.01852](https://link.zhihu.com/?target=https%3A//arxiv-web3.library.cornell.edu/abs/1502.01852)也做出了很多关于这方面的最新研究。这篇论文中作者推荐的初始化方式是：W = np.random.randn(n) \* sqrt(2.0/n)

**7、批量归一化（Batch Normalization）**

最近提出了一种批量归一化的方法。其实现方式是在连接层和激活函数之间加入一个BatchNorm层，确保每个进入激活函数的分布都成高斯分布。

原理详见[https://arxiv.org/abs/1502.03167](https://link.zhihu.com/?target=https%3A//arxiv.org/abs/1502.03167)。

该方法一定程度上减轻了对权重初始化的依赖性和麻烦。实验表明该方法有时候还有着防过拟合的作用。

**8、各种优化算法：**

传统梯度下降运算在大规模神经网络学习过程中会很慢，为了加快速度，目前有了很多优化的算法。如**Momentum、Nesterov Momentum、Adagrad、RMSprop、Adam**等。

**8.1、Mini-batch（SGD算法）：**

在做梯度更新的时候如果用全量话的数据会造成迭代的速度慢，选取Mini-batch可以加快学习速度。Min-batch的原理就是在梯度更新时不是每次对所有的数据进行求和运算，而是从总数据中随机抽取小批量的数据进行更新。

使用技巧：

如果是小批量数据，可以使用全量数据，不需要使用SGD算法，因为SGD算法更新会造成收敛的波动，因为。但如果是大批量的数据，就需要考虑使用SGD算。batch size的经验取值范围在64~512之间。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。

**8.2、Momentum下降法：**

动量更新是非常有意思的一种更新思想。它有意思的一点是引入了物理学的中的概念。momentum即动量，动量更新方法是依据物理学的势能与动能之间能量转换原理提出来的。梯度下降在更新权值时，采用如下公式:

w = w -  
learning\_rate \* dw

Momentum更新改采用如下公式：

v = mu \* v -  
learning\_rate \* dw （v可以看成速度）

w += v （通过速度v改变w，而不直接是dw）

梯度不直接对w值进行更新，而是先更新速度值，再通过速度改变位移。dw可以看成加速度（即梯度），v可以看成速度，w是位移（即权重的值）。mu是摩擦系数，一般设定为0.9。

该算法可以看成一个小球在一个封闭的物理凹形空间中，势能和动能进行转换，能量遵循守恒定律，加入摩擦系数会丢失能量，随着能量不断消耗最终会掉入全局最小值。

**8.3、Adam优化算法：**

adam是最近比较火的一种优化算法。Adam优化算法将Momentum和RMSprop两种优化算法结合到了一起。

见论文：[https://arxiv.org/pdf/1412.6980.pdf](https://link.zhihu.com/?target=https%3A//arxiv.org/pdf/1412.6980.pdf)

该文所展示的实验表明运用adam算法对比其他优化算法会使得代价函数收敛的更快。

论文中推荐的参数值**eps=1e-8, beta1=0.9, beta2=0.999**

**8.4、学习率衰减(退火算法)：**

在学习率选择上，我们经常陷入如何在模型的学习速度和收敛效果的权衡问题。比如在训练初期我们希望学习率偏大，这样训练模型的速度才能快，但是随着迭代的不断进行，我们希望学习率小一些，这样才能达到一个良好的收敛效果。

如果不设置学习率（α）衰减值，学习率为一个固定的值，会发现在做梯度下降算法时，虽然开始损失函数下降很快，但是并不能很好的收敛，会在最优值附近震荡。

# 炼丹实验室调参心得1

新开了一个专栏，为什么叫炼丹实验室呢，因为以后会在这个专栏里分享一些关于深度学习相关的实战心得，而深度学习很多人称它为玄学，犹如炼丹一般。不过即使是炼丹也是可以摸索出一些经验规律的，希望和各位炼丹术士一起多多交流。

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。  
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](http://link.zhihu.com/?target=http%3A//jmlr.org/proceedings/papers/v9/glorot10a/glorot10a.pdf)  
He初始化论文：[https://arxiv.org/abs/1502.01852](http://link.zhihu.com/?target=https%3A//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](http://link.zhihu.com/?target=https%3A//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](http://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](http://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](http://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.

# 炼丹实验室调参心得2

之前曾经写过一篇文章，讲了一些深度学习训练的技巧，其中包含了部分调参心得：[深度学习训练心得](https://zhuanlan.zhihu.com/p/20767428)。不过由于一般深度学习实验，相比普通机器学习任务，时间较长，因此调参技巧就显得尤为重要。同时个人实践中，又有一些新的调参心得，因此这里单独写一篇文章，谈一下自己对深度学习调参的理解，大家如果有其他技巧，也欢迎多多交流。

好的实验环境是成功的一半

由于深度学习实验超参众多，代码风格良好的实验环境，可以让你的人工或者自动调参更加省力，有以下几点可能需要注意：

* 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](http://link.zhihu.com/?target=http%3A//www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf)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](http://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](http://link.zhihu.com/?target=https%3A//github.com/jaberg/hyperopt), 比较简单。
  + [fmfn/BayesianOptimization](http://link.zhihu.com/?target=https%3A//github.com/fmfn/BayesianOptimization)， 比较复杂，支持并行调参。

总结

* 合理性检查，确定模型，数据和其他地方没有问题。
* 训练时跟踪损失函数值，训练集和验证集准确率。
* 使用Random Search来搜索最优超参数，分阶段从粗（较大超参数范围训练较少周期）到细（较小超参数范围训练较长周期）进行搜索。

参考资料

这里列了一些参数资料，大家有时间，可以进一步阅读。   
[Practical recommendations for gradient-based training of deep architectures by Yoshua Bengio (2012)](http://link.zhihu.com/?target=https%3A//arxiv.org/abs/1206.5533)  
[Efficient BackProp, by Yann LeCun, Léon Bottou, Genevieve Orr and Klaus-Robert Müller](http://link.zhihu.com/?target=http%3A//yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf)  
[Neural Networks: Tricks of the Trade, edited by Grégoire Montavon, Geneviève Orr, and Klaus-Robert Müller.](http://link.zhihu.com/?target=http%3A//www.springer.com/computer/theoretical%2Bcomputer%2Bscience/book/978-3-642-35288-1)