提高泛化能力的思路

目 录

[**背景** 2](#_Toc10465331)

[**前言：提升算法性能的思路** 2](#_Toc10465332)

[**1.从数据上提升性能** 3](#_Toc10465333)

[**1）收集更多的数据** 3](#_Toc10465334)

[**2）产生更多的数据** 3](#_Toc10465335)

[**3）对数据做缩放** 4](#_Toc10465336)

[**4）对数据做变换** 4](#_Toc10465337)

[**5）特征选择** 5](#_Toc10465338)

[**6）问题重构** 5](#_Toc10465339)

[**2. 从算法上提升性能** 6](#_Toc10465340)

[**1）算法的筛选** 6](#_Toc10465341)

[**2）从文献中学习** 6](#_Toc10465342)

[**3）重采样的方法** 7](#_Toc10465343)

[**3. 从算法调优上提升性能** 8](#_Toc10465344)

[**1）可诊断性** 8](#_Toc10465345)

[**2）权重的初始化** 9](#_Toc10465346)

[**3）学习率** 9](#_Toc10465347)

[**4）激活函数** 9](#_Toc10465348)

[**5）网络拓扑结构** 10](#_Toc10465349)

[**6）batch和epoch** 10](#_Toc10465350)

[**7）正则项** 11](#_Toc10465351)

[**8）优化方法和损失函数** 11](#_Toc10465352)

[**9）Early Stopping** 11](#_Toc10465353)

[**4. 用融合方法提升效果** 13](#_Toc10465354)

[**1）模型融合** 13](#_Toc10465355)

[**2）视角融合** 13](#_Toc10465356)

[**3）stacking** 13](#_Toc10465357)

[**总结** 14](#_Toc10465358)

[**补充资料** 14](#_Toc10465359)

**背景**

如何提升[深度学习](http://lib.csdn.net/base/deeplearning)模型的效果，换一种问法如何提高模型的准确率，或者反过来问如果网络模型效果不好，该怎么办？

本文中列举一些能够提升性能的方法，这些想法不仅可以用于深度学习，事实上可以用在任何[机器学习](http://lib.csdn.net/base/machinelearning)的[算法](http://lib.csdn.net/base/datastructure)上。

**前言：提升算法性能的思路**

 提升算法性能的思路划分为四块，列表如下，当然这个列表并不完整，却是很好的出发点，往往只需要尝试一个想法就能得到提升：

* **从数据上提升性能**
* **从算法上提升性能**
* **从算法调优上提升性能**
* **从模型融合上提升性能**

性能提升的力度按列表的顺序从上到下依次递减。

例如，新的建模方法或者更多的数据带来的效果提升往往好于调出最优的参数，但这并不是绝对的，大多数情况下如此。

下文中列举了一些教程以及一些[经典神经网络问题](ftp://ftp.sas.com/pub/neural/FAQ.html)。 其中有部分只是针对人工神经网络，但大多数都是通用性的，可以将它们与其它技术结合起来使用。

**一、从数据上提升性能**

调整训练数据或是问题的抽象定义方法可能会带来巨大的效果改善，甚至是最显著的改善。

**1.1、收集更多的数据**

模型的质量往往取决于训练数据的质量，需要确保使用的数据是针对问题最有效的数据，训练时一般希望数据尽可能多。深度学习和其它现代的非线性机器学习模型在[大数据](http://lib.csdn.net/base/hadoop)集上的效果更好，尤其是深度学习， 下面的图片：



当然并非总是数据越多效果越好，但是多数情况下如此。如果可以选择，建议选择要更多的数据。参考资料：[数据集压倒算法](https://www.edge.org/response-detail/26587)

**1.2、产生更多的数据**

上文提到，深度学习算法往往在数据量大的时候效果更好，如果由于某些原因得不到更多的数据，那么可以制造一些数据。

* 如果数据是数值型的向量，那么随机生成已有向量的变形向量。
* 如果数据是图像，用已有的图像随机生成相似图像。
* 如果数据是文本，则可以通过工具随机生成。

这类做法通常被称为数据扩展或是数据生成。

想要达到上述效果可以使用生成模型，也可以用一些简单的小技巧。 举个例子，若是用图像数据，简单地随机选择和平移已有的图像，就能较大提升模型的泛化能力。

有时候是往数据中增加噪声，这相当于是一种规则方法，避免过拟合训练数据。

参考资料：

* [深度学习中的图像数据扩充](http://machinelearningmastery.com/image-augmentation-deep-learning-keras/)
* [训练含有噪声的数据](ftp://ftp.sas.com/pub/neural/FAQ3.html#A_jitter)

**1.3、对数据做缩放**

这种方法简单有效。   
使用神经网络模型的一条经验法宝：将数据缩放到激活函数的阈值范围。

如果使用sigmoid激活函数，将数据缩放到0~1之间。如果选用tanh激活函数，将值域控制在-1~1之间。

输入、输出数据都经过同样的变换。比如，如果在输出层有一个sigmoid函数将输出值转换为二值数据，则将输出的y归一化为二进制。如果选用的是softmax函数，对y进行归一化还是有效的。

建议将训练数据扩展生成多个不同的版本：

* 归一化到0 ~ 1
* 归一化到-1 ~ 1
* 标准化

然后在每个数据集上[测试](http://lib.csdn.net/base/softwaretest)模型的性能，选用最好的一组生成数据。

如果更换了激活函数，最好重复做一次这个步骤。在模型中不适合计算大的数值。

此外，还有许多其它方法来压缩模型中的数据，比如对权重和激活值做归一化，后文中将会介绍这些技巧。

参考资料：

* [我需要对输入数据（列向量）做标准化吗?](ftp://ftp.sas.com/pub/neural/FAQ2.html#A_std)
* [如何用Scikit-Learn准备机器学习的输入数据](http://machinelearningmastery.com/prepare-data-machine-learning-python-scikit-learn/)

**1.4、对数据做变换**

与1.3的方法相关，但是需要更多的工作量。

首先必须真正了解所用到的数据，将数据可视化，然后挑出异常值，估测每一列数据的分布：

* 这一列数据是不是倾斜的高斯分布，若是如此，尝试用Box-Cox方法纠正倾斜
* 这一列数据是不是指数分布，若是如此，则进行对数变换
* 这一列数据是不是存在某些特性，但是难以直观地发现，尝试一下对数据平方或者开方
* 是否可以将特征离散化，以便更好地强调一些特征

根据经验和直觉，尝试以下几种方法：

* 是否可以用投影的方法对数据预处理，比如PCA？
* 是否可以将多个属性合并为单个值？
* 是否可以发掘某个新的属性，用布尔值表示？
* 是否可以在时间尺度或是其它维度上有些新发现？

神经网络有特征学习的功能，它们能够完成这些事情，不过若是可以将问题的结构更好地呈现出来，网络模型学习的速度就会更快。在训练集上尝试各种变换方法，看看哪些方法有些，而哪些不起作用。

参考资料：

* [如何定义你的机器学习问题](http://machinelearningmastery.com/how-to-define-your-machine-learning-problem/)
* [特征挖掘工程，如何构造特征以及如何提升](http://machinelearningmastery.com/discover-feature-engineering-how-to-engineer-features-and-how-to-get-good-at-it/)
* [如何用Scikit-Learn准备机器学习的输入数据](http://machinelearningmastery.com/prepare-data-machine-learning-python-scikit-learn/)

**1.5、特征选择**

神经网络受不相关数据的影响很小。

神经网络会对不相关数据赋予一个趋近于0的权重，几乎忽略此类数据特征对预测值的贡献，因此考虑是否可以移除训练数据的某些属性？ 有许多的特征选择方法和特征重要性方法来鉴别哪些特征可以保留，哪些特征需要移除。

据此可以在相同的神经网络模型上选择尝试多个方法，看看它们的效果分别如何：

* 也许用更少的特征也能得到同样的、甚至更好的效果。
* 也许所有的特征选择方法都选择抛弃同一部分特征属性，那么就该好好审视这些无用的特征。
* 也许选出的这部分特征带来了新的启发，构建出更多的新特征。

参考资料：

* [特征选择入门介绍](http://machinelearningmastery.com/an-introduction-to-feature-selection/)
* [基于Python的机器学习中的特征选择问题](http://machinelearningmastery.com/feature-selection-machine-learning-python/)

**1.6、问题重构**

上文所收集到的观测数据是否是描述问题的唯一途径？是否有其它的途径能更清晰地将问题的结构暴露出来？这种反问可以强迫我们拓宽思路。当然这很难做好，尤其是在已经投入大量的时间、精力、金钱在现有的方法上的前提下。

以下列举了4种不同的方式：

* 也许可以将时间元素融入到一个窗口之中
* 也许分类问题可以转化为回归问题，反之亦然
* 也许可以把二值类型的输出转化为softmax的输出
* 也许可以对子问题建模

深入思考问题是一个好习惯，最好在选择工具下手之前先完成上述步骤，以减少无效的精力投入。另外，也不必抛弃前期的大量工作，详情可以参见后文内容。

参考资料：

* [如何定义机器学习问题](http://machinelearningmastery.com/how-to-define-your-machine-learning-problem/)

**二、从算法上提升性能**

机器学习总是与算法相关。所有的理论和数学知识都在描述从数据中学习决策过程的不同方法（这里仅讨论预测模型）。选用深度学习来求解是不是最合适的技术呢？在下文中针对算法的选择进行阐述，后续内容将具体介绍如何据此提升深度学习的效果。

**2.1、算法的筛选**

在比较多种算法之前一般不可能知道哪种算法对实际应用效果最好。如果事先就已经知道，可能也就不需要机器学习了。

有哪些证据可以证明现在已经采用的方法是最佳选择呢？当在所有可能出现的问题上进行效果评测时，没有哪一项单独的算法效果会好于其它算法。所有的算法都是平等的。

当前的热门算法也许并不是最合适的。建议先收集证据，先假设有其它的合适算法。筛选一些常用的算法，挑出其中适用的几个。

* 尝试一些线性算法，比如逻辑回归和线性判别分析
* 尝试一些树模型，比如CART、随机森林和梯度提升
* 尝试SVM和kNN等算法
* 尝试其它的神经网络模型，比如LVQ、MLP、CNN、LSTM等等

采纳效果较好的几种方法，然后精细调解参数和数据来进一步提升效果。将所选用的深度学习方法与上述这些方法比较，看看是否有所提升？也许可以放弃深度学习模型转而选择更简单模型，训练的速度也会更快，而且模型易于理解。

参考资料：

* [一种数据驱动的机器学习方法](http://machinelearningmastery.com/a-data-driven-approach-to-machine-learning/)
* [面对机器学习问题为何需要筛选算法](http://machinelearningmastery.com/why-you-should-be-spot-checking-algorithms-on-your-machine-learning-problems/)
* [用scikit-learn筛选机器学习的分类算法](http://machinelearningmastery.com/spot-check-classification-machine-learning-algorithms-python-scikit-learn/)

**2.2、从文献中学习**

从文献中获取思路是一条捷径，其它人是否已经做过类似的问题，他们使用的是什么方法。阅读论文、书籍、问答网站、教程以及Google提供的一切信息。记下所有的思路，然后沿着这些方向继续探索。这并不是重复研究，这是帮助发现新的思路。

**优先选择已经发表的论文，**已经有许许多多的聪明人写下了很多有意思的事情。

参考资料：

* [如何研究一种机器学习算法](http://machinelearningmastery.com/how-to-research-a-machine-learning-algorithm/)
* [Google学术](http://scholar.google.com/)

**2.3、重采样的方法**

首先必须明白自己模型的效果如何，估计的模型效果是否可靠，深度学习模型的训练速度很慢。这就意味着我们不能用标准的黄金法则来评判模型的效果，比如k折交叉验证。

* 也许只是简单地把数据分为训练集和测试集。如果是这样，就需要保证切分后的数据分布保持不变。单变量统计和数据可视化是不错的方法。
* 也许可以扩展硬件来提升效果。举个例子，如果有一个集群或是AWS的账号，可以并行训练n个模型，然后选用它们的均值和方差来获取更稳定的效果。
* 也许可以选择一部分数据做交叉验证（对于early stopping非常有效）。
* 也许可以完全独立地保留一部分数据用于模型的验证。
* 另一方面，也可以让数据集变得更小，采用更强的重采样方法。
* 也许会看到在采样后的数据集上训练得到的模型效果与在全体数据集上训练得到的效果有很强的相关性。那么，你就可以用小数据集进行模型的选择，然后把最终选定的方法应用于全体数据集上。
* 也许可以任意限制数据集的规模，采样一部分数据，用它们完成所有的训练任务。

**总之，必须对模型效果的预测有充分的把握。**   
参考资料：

* [用Keras评估深度学习模型的效果](http://machinelearningmastery.com/evaluate-performance-deep-learning-models-keras/)
* [用重采样的方法评估机器学习算法的效果](http://machinelearningmastery.com/evaluate-performance-machine-learning-algorithms-python-using-resampling/)

**三、从算法调优上提升性能**

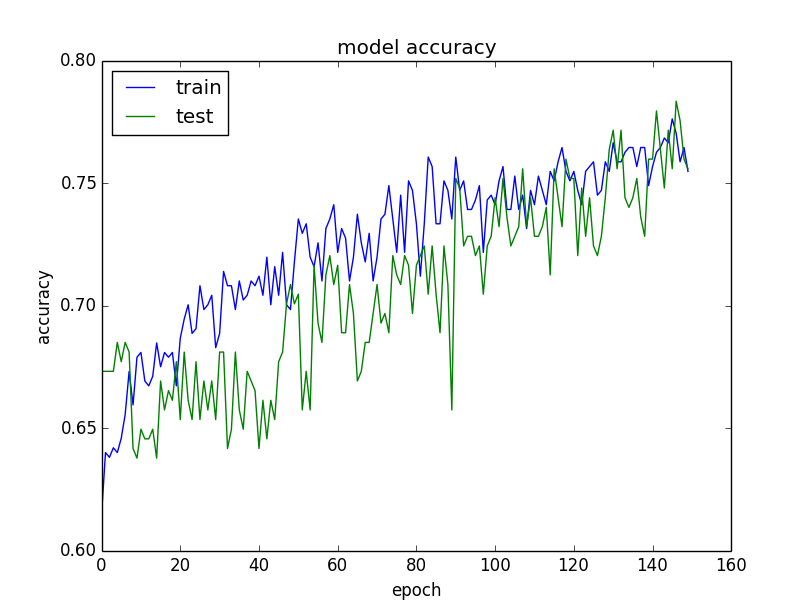
通过算法筛选往往能找出一到两个效果不错的算法。但想要达到这些算法的最佳状态需要耗费数日、数周甚至数月。可能需要指定参数来多次（3-10次甚至更多）训练模型，以得到预计效果最好的一组参数。对每个参数都要不断的尝试。

参考资料：

* [如何用Keras网格搜索深度学习模型的超参数](http://machinelearningmastery.com/grid-search-hyperparameters-deep-learning-models-python-keras/)

**2.1、模型可诊断性**

只有知道为什么模型的性能不再有提升了，才能基于此进行调整从而达到最好的效果。 是因为模型过拟合，还是欠拟合？模型总是处于这两种状态之间，只是程度不同罢了。一种快速查看模型性能的方法就是每一步计算模型在训练集和验证集上的表现，将结果绘制成图表。



在训练集和验证集上测试模型的准确率

* 如果训练集的效果好于验证集，说明可能存在过拟合的现象，试一试增加正则项
* 如果训练集和验证集的准确率都很低，说明可能存在欠拟合，你可以继续提升模型的能力，延长训练步骤。
* 如果训练集和验证集的曲线有一个焦点，可能需要用到early stopping的技巧了

经常绘制类似图表，深入研究并比较不同的方法，以提高模型的性能。 另一种有效的诊断方法是研究模型正确预测或是错误预测的样本。

在某些场景下，尝试以下思路：

* 准备更多的难预测的样本数据
* 从训练集中删去那些容易被学习的样本
* 有针对性地对不同类型的输入数据训练不同的模型

参考资料：

* [用Keras展现深度学习模型的训练过程](http://machinelearningmastery.com/display-deep-learning-model-training-history-in-keras/)
* [机器学习算法的过拟合和欠拟合](http://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/)

**2.2、权重的初始化**

经验：用小的随机数初始化权重。

一般情况下，根据经验操作就已经足够，但是这不一定是该网络模型的最佳选择，不同的激活函数也可以有不同的应对策略。保持模型结构不变，尝试不同的初始化策略。权重值就是模型需要训练的参数。不同的权重值都能取得不错的效果，但最终期望得到更好的效果。

* 尝试所有的初始化方法，找出最好的一组初始化值
* 试一试用非监督式方法预学习，比如自动编码机
* 尝试用一组现有的模型权重参数，然后重新训练输入和输出层（迁移学习）

修改权重初始化值的方法与修改激活函数或者目标函数的效果相当。

参考资料：

* [深度网络模型的初始化](http://deepdish.io/2015/02/24/network-initialization/)

**2.3、学习率**

调节学习率也能带来效果提升。

以下列举了一些思路：

* 尝试非常大、非常小的学习率
* 根据参考文献，在常规值附近用网格化搜索
* 尝试使用逐步减小的学习率
* 尝试每隔固定训练步骤衰减的学习率
* 尝试增加一个向量值，然后用网格搜索

大的网络模型需要更多的训练步骤，反之亦然。如果存在很多的神经节点和网络层，需要加大学习率。学习率与训练步骤、batch大小和优化方法都有耦合关系。

参考资料：

* [使用Keras对深度学习模型进行学习率调节](http://machinelearningmastery.com/using-learning-rate-schedules-deep-learning-models-python-keras/)
* [反向传播算法该选用什么学习率？](ftp://ftp.sas.com/pub/neural/FAQ2.html#A_learn_rate)

**2.4、激活函数**

可以考虑选用ReLU激活函数，因为效果更好。在ReLU之前流行sigmoid和tanh，然后是输出层的softmax、线性和sigmoid函数。这三种函数都可以进行尝试，需要把输入数据归一化到它们的值域范围，同时根据输出内容的形式选择转移函数。

例如，将二值分类的sigmoid函数改为回归问题的线性函数，然后对输出值进行再处理。同时，需要根据实际结果调整合适的损失函数。

参考资料：

* [为何使用激活函数？](ftp://ftp.sas.com/pub/neural/FAQ2.html#A_act)

**2.5、网络拓扑结构**

调整网络的拓扑结构也会对模型性能提升有帮助。需要设计多少个节点，需要几层网络，必须找到一组合理的参数配置：

* 尝试加一层有许多节点的隐藏层（拓宽）
* 尝试一个深层的神经网络，每层节点较少（纵深）
* 尝试将上面两种组合
* 尝试模仿近期发表的含有相似问题的论文
* 尝试拓扑模式和书本上的经典技巧（参照参考资料链接）

越大的网络模型有越强的表达能力，更多层的结构提供了抽象特征的更多结构化组合的可能，后期的网络模型需要更多的训练过程，需要不断地调节训练步长和学习率。

参考资料：

* [我的网络模型该设计几层呢？](ftp://ftp.sas.com/pub/neural/FAQ3.html#A_hl)
* [我的网络模型该设计几个节点呢？](ftp://ftp.sas.com/pub/neural/FAQ3.html#A_hu)

**2.6、batch和epoch**

batch的大小决定了梯度值，以及权重更新的频率。一个epoch指的是训练集的所有样本都参与了一轮训练，以batch为序。尝试过不同的batch大小和epoch的次数。在前文中，我们已经讨论了学习率、网络大小和epoch次数的关系。深度学习模型常用小的batch和大的epoch以及反复多次的训练。

尝试以下的思路：

* 尝试将batch大小设置为全体训练集的大小（batch learning）
* 尝试将batch大小设置为1（online learning）
* 用网格搜索尝试不同大小的mini-batch（8，16，32，…）
* 尝试再训练几轮epoch，然后继续训练很多轮epoch

尝试设置一个近似于无限大的epoch次数，然后输出一些中间结果，在此基础上寻找效果最好的模型。有些模型结构对batch的大小很敏感。一般多层感知器对batch的大小很不敏感，而LSTM和CNN则非常敏感。

参考资料：

* [什么是批量学习、增量学习和在线学习？](ftp://ftp.sas.com/pub/neural/FAQ2.html#A_styles)
* [直觉上，mini-batch的大小如何影响（随机）梯度下降的效果？](https://www.quora.com/Intuitively-how-does-mini-batch-size-affect-the-performance-of-stochastic-gradient-descent)

**2.7、正则项**

正则化是克服训练数据过拟合的有效方法。

比较热门的正则化方法是dropout， Dropout方法在训练过程中随机地略过一些神经节点，强制让同一层的其它节点接管，这是简单却有效的方法：

* 权重衰减来惩罚大的权重值
* 激活限制来惩罚大的激活函数值

尝试用各种惩罚措施和惩罚项进行实验。

参考资料：

* [使用Keras对深度学习模型做dropout正则化](http://machinelearningmastery.com/dropout-regularization-deep-learning-models-keras/)
* [什么是权值衰减？](ftp://ftp.sas.com/pub/neural/FAQ3.html#A_decay)

**2.8、优化方法和损失函数**

以往主要的求解方法是随机梯度下降，这是默认的方法。先用它得到一个结果，然后调节不同的学习率、动量值进行优化。

目前许多更高级的优化方法都用到更多的参数，结构更复杂，收敛速度更快。为了更好的优化模型的性能，需要深入钻研每个参数，然后用网格搜索法测试不同的取值。这个过程比较艰辛，需要花很多时间，若时间充裕值得去尝试。

更新/更流行的方法收敛速度更快，能够快速了解某个网络拓扑的潜力，例如：

* [ADAM](https://arxiv.org/abs/1412.6980)
* RMSprop

可以探索其它的优化算法，例如更传统的算法（Levenberg-Marquardt）和比较新的算法（基因算法）。这些方法能给SGD创造好的开端，便于后续调优。待优化的损失函数则与需要解决的问题更相关。

另外也有一些常用的方法（比如回归问题常用MSE和MAE），这可能也与输入数据的尺度以及所使用的激活函数相关。

参考资料：

* [梯度下降优化算法概览](http://sebastianruder.com/optimizing-gradient-descent/)
* [什么是共轭梯度和Levenberg-Marquardt？](http://blog.csdn.net/starzhou/article/details/52754436)
* [深度学习的优化方法，2011](http://ai.stanford.edu/~ang/papers/icml11-OptimizationForDeepLearning.pdf)

**2.9、提早结束训练**

可以在模型性能开始下降的时候停止训练，这样会节省大量时间，同时也许因此就能使用更精细的重采样方法来评价模型了。

提早结束训练也是防止数据过拟合的一种正则化方法，需要你在每轮训练结束后观察模型在训练集和验证集上的效果。一旦模型在验证集上的效果下降了，则可以停止训练。可以设置检查点，保存当时的状态，然后模型可以继续学习。

参考资料：

* [如何在Keras给深度学习模型设置check-point](http://machinelearningmastery.com/check-point-deep-learning-models-keras/)
* [什么是early stopping？](ftp://ftp.sas.com/pub/neural/FAQ3.html#A_stop)

**四、用融合方法提升性能**

可以将多个模型的预测结果融合，这是继模型调优之后另一个大的提升手段。往往将几个效果还可以的模型的预测结果融合，取得的效果要比多个精细调优的模型分别预测的效果好。

**2.1、模型融合**

如果训练了多个深度学习模型，每一个的效果都不错，则将它们的预测结果取均值。模型的差异越大，效果越好。

举个例子，可以使用差异很大的网络拓扑和技巧，如果每个模型都独立且有效，那么集成后的结果效果更稳定，相反的，也可以反过来做实验。

每次训练网络模型时，都以不同的方式初始化，最后的权重也收敛到不同的值。多次重复这个过程生成多个网络模型，然后集成这些模型的预测结果。它们的预测结果会高度相关，但对于比较难预测的样本也许会有一点提升。

参考资料：

* [用scikit-learn集成机器学习算法](http://machinelearningmastery.com/ensemble-machine-learning-algorithms-python-scikit-learn/)
* [如何提升机器学习的效果](http://machinelearningmastery.com/how-to-improve-machine-learning-results/)

**2.2、视角融合**

视角融合的目的还是得到有用的模型，但是与模型融合的方式不同（如不相关的预测结果）。可以根据模型融合中提到的方法，对训练数据采取完全不同的缩放和变换技巧。所选用的变化方式和问题的刻画角度差异越大，效果提升的可能性也越大。

**2.3、stacking**

可以尝试将各个模型的预测结果相融合，这种方法简称为stacking。

用简单的线性回归的方式学习各个模型预测值的权重，把各个模型预测结果取均值作为baseline，用带权重的融合进行尝试。

参考资料：

* [Stacked Generalization (Stacking)](http://machine-learning.martinsewell.com/ensembles/stacking/)

**附：补充资料**

* [神经网络常见问答](ftp://ftp.sas.com/pub/neural/FAQ.html)
* [如何用网格搜索法求解深度学习模型的超参数](http://machinelearningmastery.com/grid-search-hyperparameters-deep-learning-models-python-keras/)
* [深度神经网络必知的技巧](http://lamda.nju.edu.cn/weixs/project/CNNTricks/CNNTricks.html)
* [如何提升深度神经网络的验证准确率？](http://stackoverflow.com/questions/37020754/how-to-increase-validation-accuracy-with-deep-neural-net)