实践方法论

1. 一个优秀的机器学习实践者需要知道：
    - 存在哪些算法以及这些算法为何有效的原理
    - 如何针对具体应用挑选一个合适的算法
    - 如何监控算法，并根据实验反馈改进机器学习系统
2. 实际开发过程中，实践者需要决定：是否收集更多的数据、是否需要增加/降低模型容量、是否需要添加/删除正则化项、是否需要改进模型的优化算法、是否需要改进模型的近似推断....这些都需要大量的时间
3. 在实际应用中，正确使用一个普通算法通常要比草率的使用一个不清楚的算法要效果更好
4. 正确应用一个算法需要掌握一些相当简单的方法论：
    - 确定目标：使用什么样的误差度量（是准确率还是召回率？），并为此误差度量确定目标（我们期望达到什么样的效果：如召回率大于95%？）
        - 这些目标和误差度量取决于该应用为了解决什么问题
    - 尽快建立一个端到端的工作流程，包括计算出合适的性能度量（如：计算出召回率）
    - 搭建系统，并确定性能瓶颈。
        - 检查哪个部分的性能差于预期
        - 检查差于预期的原因：是否过拟合、欠拟合、还是数据或者软件缺陷造成的
    - 根据具体观察反复进行增量式改动：如收集新数据、调整超参数、改进算法

一、性能度量

1. 确定目标，即使用什么误差度量是第一步
    - 因为误差度量将指导接下来的所有工作
    - 同时我们也能够了解大概能得到什么级别的目标性能
2. 对大多数应用而言，不可能实现绝对零误差
    - 即使有无限的训练数据，并且恢复了真正的概率分布，但是由于输入特征可能无法包含输出变量的完整信息、或者系统本质上是个随机系统，则仍然产生了误差
    - 当然实际上我们也不可能有无限的训练数据
3. 通常我们需要收集更多的数据。但是我们需要在收集更多数据的成本，与进一步减少误差的价值之间权衡
4. 通常我们需要对错误率定一个 baseline 从而判断预测得好坏
    - 对于学术界，我们可以将先前公布的基准结果作为 baseline
    - 在工业界，我们从安全的、能带来性价比、或者能吸引用户的角度来确定错误率的 baseline
5. 性能度量：表明从哪个角度来度量算法的性能
    - 常见的有：精度`precision`，召回率`recall`以及`PR curve`，`ROC`曲线，`F-score`等
    - 还有一种：覆盖率。它是机器学习系统能够产生响应的样本占所有样本的比例
        - 一个系统可以拒绝处理任何样本，从而达到 100% 的精度。但是覆盖率为 0%
    - 也可以从专业角度考量：点击率、用户满意度调查等等
    - 最重要的是：确定使用哪个性能度量

二、默认的基准模型

1. 根据问题的复杂性，项目开始时可能无需使用深度学习
    - 如果只需要正确选择几个线性权重就能解决问题，那么项目开始可以使用一个简单的统计模型，如逻辑回归
2. 如果问题属于 “AI-完全”类型的，如对象识别、语音识别等，那么项目开始于一个合适的深度学习模型，则效果会比较好
3. 根据数据结构选择一类合适的模型：
    - 如果是以固定大小的向量作为输入的有监督学习，那么可以使用全连接的前馈网络
    - 如果输入已知的拓扑结构（如图像），则可以使用卷积网络
    - 如果输入或者输出是一个序列，则使用门控循环网络（LSTM 或者 GRU）
4. 刚开始时可以使用某些分段线性单元：如 `ReLU` 或者其扩展
5. 可以选择具有衰减学习率以及动量的 `SGD` 作为优化算法
    - 常见的衰减方法有：
        - 衰减到固定最低学习率的线性衰减
        - 指数衰减
        - 每次发生验证错误停滞时将学习率降低 2-10 倍的衰减策略
    - 另一种优化选择是 `Adam` 算法
6. `batch normalization` 对优化性能有着显著的影响，特别是对于卷积网络和具有 `sigmoid` 非线性函数的网络而言
    - 最初的基准中，可以忽略 `batch normalization`
    - 当优化似乎出现问题时，应立即使用 `batch normalization`
7. 除非训练集包含数千万或者更多的样本，否则项目一开始就应该包含一些温和的正则化
    - 建议采用早停策略
    - 建议采用 `dropout` 策略，它也兼容很多模型以及许多正则化项
    - `batch normalization` 有时可以降低泛化误差，此时可以省略 `dropout` 策略。因为用于 `normalize` 的统计量估计本身就存在噪音
8. 如果我们的任务和另一个被广泛研究的任务相似，那么通过复制之前研究中已知的性能良好的模型和算法，可能会得到很好的效果
    - 你可以从该任务中复制一个训练好的模型。如从 `ImageNet` 上训练好的卷积网络的特征来解决其他计算机视觉任务
9. 对于是否使用无监督学习，和特定领域有关
    - 对于某些领域，如自然语言处理，能大大受益于无监督学习技术
    - 在其他领域，目前无监督学习并没有带来好处
    - 如果你所处理的应用，无监督学习是非常重要的，那么将其包含在第一个端到端的基准中；否则只有在解决无监督问题时，才第一次尝试使用无监督学习

三、决定是否收集更多数据

1. 建立第一个端到端的系统后，就可以度量算法的性能并决定如何改进算法
    - 很多新手忍不住尝试很多不同的算法来改进
    - 实际上，收集更多的数据往往比改进学习算法要有用的多
2. 决定是否需要收集更多数据的标准：
    - 首先：确定训练集上的性能可否接受。如果模型在训练集上的性能就很差，那么没必要收集更多的数据。

        - 此时可以尝试增加更多的网络层
        - 或者每层增加更多的隐单元从而增加模型的规模
        - 也可以尝试调整学习率等超参数来改进算法
        - 如果这些都不行，则说明问题可能源自训练数据的质量：数据包含太多噪声，或者数据未能包含预测输出所需要的正确输入。此时我们需要重新开始收集干净的数据，或者收集特征更丰富的数据
    - 如果模型在训练集上的性能可以接受，那么我们开始度量测试集上的性能。
    - 如果测试集上的性能也可以接受，则任务完成
    - 如果测试集上的性能比训练集的要差得多，则收集更多的数据时最有效的解决方案之一
    - 此时主要考虑三个要素：
        - 收集更多数据的代价和可行性
        - 其他方法降低测试误差的代价和可行性
        - 增加数据数量能否显著提升测试集性能
3. 如果增加数据数量代价太大，那么一个替代的方案是降低模型规模，或者改进正则化（调整超参数、或者加入正则化策略）
    - 如果调整正则化参数之后，训练集性能和测试集性能之间的差距还是无法接受，则只能收集更多的数据
4. 当决定增加数据数量时，还需要确定收集多少数据。可以绘制曲线来显式训练集规模和泛化误差之间的关系
    - 根据曲线的延伸，可以预测还需要多少训练数据来到达一定的性能
    - 通常加入小比例的样本不会对泛化误差产生显著的影响。因此建议在对数尺度上考虑训练集的大小，以及增加的数据数量
5. 如果收集更多的数据是不可行的（成本太高无法实现，或者无法改进泛化误差），那么改进泛化误差的唯一方法是：改进学习算法本身
    - 这是属于研究领域，并不是对实践者的建议

四、选择超参数

1. 大部分深度学习算法都有许多超参数来控制不同方面的算法表现
    - 有的超参数会影响算法运行的时间和存储成本
    - 有的超参数会影响学习到的模型质量，以及在新输入上推断正确结果的能力
2. 有两种选择超参数的方法：
    - 手动选择。手动选择超参数需要了解超参数做了些什么，以及机器学习模型如何才能取得良好的泛化
    - 自动选择。自动选择超参数算法不需要你了解超参数做了什么以及机器学习模型如何才能取得零号的泛化，但是它往往需要更高的计算成本

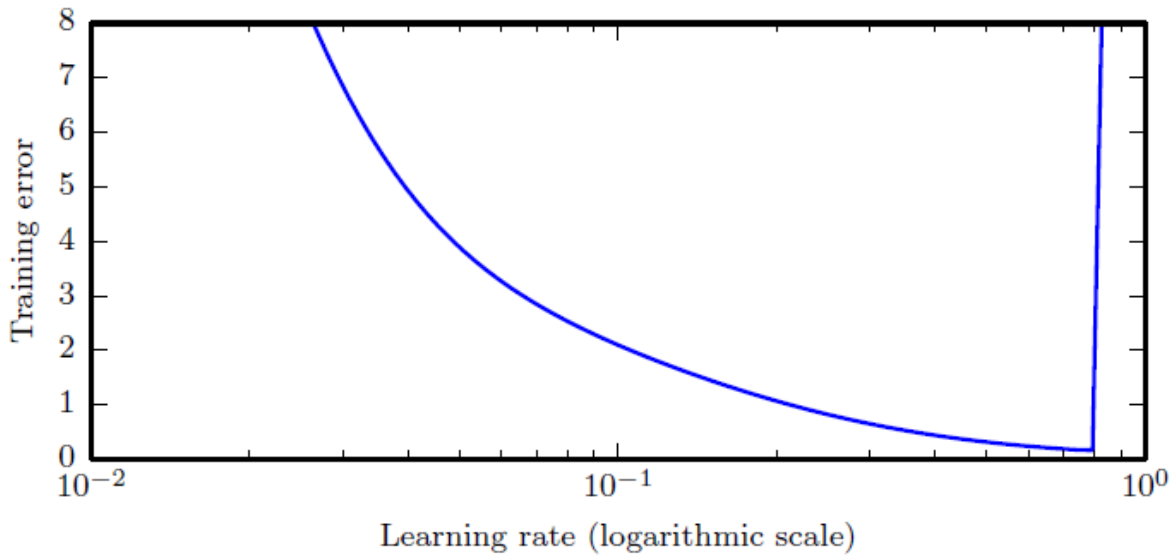
4.1 手动调整超参数

1. 手动设置超参数：

   我们必须了解超参数、训练误差、泛化误差、计算资源（内存和运行时间） 之间的关系。这要求我们切实了解一个学习算法有效容量的基本概念
2. 手动搜索超参数的任务是：在给定运行时间和内存预算范围的条件下，最小化泛化误差
    - 我们不讨论超参数对于运行时间和内存的影响，因为它们高度依赖于平台
3. 手动搜索超参数的主要目标是：调整模型的有效容量以匹配任务的复杂性
    - 模型的有效容量受限于三个因素：
        - 模型的表示容量。更多的网络层、每层更多的隐单元的模型具有更大的容量（能表达更复杂的函数）

        - 学习算法成功最小化训练模型代价函数的能力
        - 训练过程正则化模型的程度
    - 模型的表示容量并不是越高越好。如果无法找到合适的代价函数来最小化训练代价、或者正则化项排除了某些合适的函数，那么即使模型的表达能力再强，也无法学习出合适的函数。
4. 如果以超参数为自变量，泛化误差为因变量。那么会在的曲线通常会表现为 U 形
    - 在某个极端情况下，超参数对应着低容量（并不是超参数越小，模型容量越低；也可能是相反的情况）。此时泛化误差由于训练误差较大而很高。这就是欠拟合
    - 在另一个极端情况下，超参数对应着高容量，此时泛化误差也很大。这就是过拟合
        - 过拟合中，泛化误差较大的原因是：虽然此时训练误差较小，但是训练误差和测试误差之间的差距较大。
    - 最优的模型容量位于曲线中间的某个位置
5. 对于某些超参数，当超参数值太大时，会发生过拟合。如中间层隐单元的数量，数量越大，模型容量越高，也更容易发生过拟合。

   对于某些超参数，当超参数值太小时，也会发生过拟合。如 $L2$ 正则化的权重系数，系数为0，表示没有正则化，此时很容易过拟合。
6. 并不是每个超参数都对应着完整的 U 形曲线
    - 很多超参数是离散的，如中间层隐单元的数量，或者 `maxout` 单元中线性片段的数目
    - 有些超参数甚至是二值的。如是否决定对输入特征进行标准化这个布尔值的超参数
    - 有些超参数可能有最小值或者最大值
7. 学习率可能是最重要的超参数
    - 如果你只有时间来调整一个超参数，那么就调整学习率
    - 相比其他超参数，学习率以一种更复杂的方式控制模型的有效容量：
        - 当学习率大小适当时，模型的有效容量最高
        - 当学习率过大时，梯度下降可能会不经意地增加而非减少训练误差。在理想的二次情况下，如果学习率是最佳值的两倍时，会发生这种情况
        - 当学习率太小时，训练不仅会很慢，还有可能永久停留在一个很高的训练误差。对于这种情况我们知之甚少（但是我们可以知道这种情况不会发生在一个凸损失函数中）
    - 学习率关于训练误差具有 U 形曲线。泛化误差也是类似的 U 形曲线，但是正则化作用在学习率过大或者过小处比较复杂



8. 调整学习率以外的其他参数时，需要同时监测训练误差和测试误差，从而判断模型是否过拟合或者欠拟合，然后适当调整其容量
    - 如果训练集错误率大于目标错误率（这个根据任务背景人工指定），那么只能增加模型容量以改进模型。但是这增加了模型的计算代价
    - 如果测试集错误率大于目标错误率，则有两个方法：
        - 如果训练误差较小（这说明模型容量较大），则表明测试误差取决于训练误差与测试误差之间的差距。要减少这个差距，我们可以改变正则化超参数，以减少有效的模型容量
        - 如果训练误差较大（者说明模型容量较小），那么也只能增加模型容量以改进模型
9. 通常最佳性能来自于正则化很好的大规模模型，如使用`Dropout`的神经网络
10. 大部分超参数可以推论出是否增加或者减少模型容量，部分示例如下：

超参数	容量何时增加	原因	注意事项
隐单元数量	增加	增加隐单元数量会增加模型的表示能力	几乎模型每个操作所需要的时间和内存代价都会随隐单元数量的增加而增加
学习率	调至最优	不正确的学习率，不管是太高还是太低都会由于优化失败而导致低的有效容量的模型	
卷积核宽度	增加	增加卷积核宽度会增加模型的参数数量	较宽的卷积核导致较窄的输出尺寸，除非使用隐式零填充来减少此影响，否则会降低模型容量。较宽的卷积核需要更多的内存来存储参数，并增加运行时间
隐式零填充	增加	在卷积之前隐式添加零能保持较大尺寸的表示	大多数操作的时间和内存代价会增加
权重衰减系数	降低	降低权重衰减系数使得模型参数可以自由地变大	
dropout比率	降低	较少地丢弃单元可能更多的让单元彼此“协力”来适应训练集	

11. 手动调整超参数时不要忘记最终目标：提升测试集性能
    - 加入正则化只是实现这个目标的一种方法
    - 如果训练误差很低，也可以通过收集更多的训练数据来减少泛化误差。如果训练误差太大，则收集更多的训练数据就没有意义。
    - 实践中的一种暴力方法是：不断提高模型容量和训练集的大小。这种方法增加了计算代价，只有在拥有充足的计算资源时才可行

4.2 自动超参数优化算法

1. 理想的学习算法应该是只需要输入一个数据集，然后就可以输出学习的函数而不需要人工干预调整超参数
    - 一些流行的算法如逻辑回归、支持向量机，其流行的部分原因是：这类算法只需要调整一到两个超参数，而且性能也不错
    - 某些情况下，神经网络只需要调整少量的超参数就能达到不错的性能；但是大多数情况下需要调整更多的超参数
2. 原则上可以开发出封装了学习算法的超参数优化算法，并自动选择其超参数
    - 超参数优化算法往往有自己的超参数（如每个超参数的取值范围），这将问题变得更复杂
    - 我们可以人工选择参数优化算法这一级的超参数，因为这一级的超参数通常更容易选择

4.3 网格搜索

1. 当只有三个或者更少的超参数时，常见的超参数搜索方法是：网格搜索
    - 对于每个超参数，选择一个较小的有限值集合去搜索
    - 然后这些超参数笛卡尔乘积得到多组超参数
    - 网格搜索使用每一组超参数训练模型，挑选验证集误差最小的超参数作为最好的超参数
2. 如何确定搜索集合的范围？
    - 如果超参数是数值，则搜索集合的最小、最大元素可以基于先前相似实验的经验保守地挑选出来
    - 如果超参数是离散的，则直接使用离散值
3. 通常网格搜索会在对数尺度下挑选合适的值
4. 通常重复进行网格搜索时，效果会更好。假设在集合 `{-1,0,1}` 上网格搜索超参数 α
    - 如果找到的最佳值是 1，那么说明我们低估了 α 的取值范围。此时重新在 `{1,2,3}` 上搜索
    - 如果找到的最佳值是 0，那么我们可以细化搜索范围以改进估计。此时重新在 `{-0.1,0,0.1}` 上搜索
5. 网格搜索的一个明显问题时：计算代价随着超参数数量呈指数级增长。
    - 如果有 m 个超参数，每个最多取 n 个值，那么所需的试验数将是 $O(n^m)$ 。虽然我们可以并行试验，但是指数级增长的计算代价仍然不可行

4.4 随机搜索

1. 随机搜索是一种可以替代网格搜索的方法，它编程简单、使用方便、能更快收敛到超参数的良好取值：
    - 首先为每个超参数定义一个边缘分布，如伯努利分布（对应着二元超参数）或者对数尺度上的均匀分布（对应着正实值超参数）
    - 然后假设超参数之间相互独立，从各分布中抽样出一组超参数。
    - 使用这组超参数训练模型
    - 经过多次抽样 -> 训练过程，挑选验证集误差最小的超参数作为最好的超参数
2. 随机搜索的优点：
    - 不需要离散化超参数的值，也不需要限定超参数的取值范围。这允许我们在一个更大的集合上进行搜索
    - 当某些超参数对于性能没有显著影响时，随机搜索相比于网格搜索指数级地高效，它能更快的减小验证集误差
3. 与网格搜索一样，我们通常会基于前一次运行结果来重复运行下一个版本的随机搜索
4. 随机搜索比网格搜索更快的找到良好超参数的原因是：没有浪费的实验
    - 在网格搜索中，两次实验之间只会改变一个超参数 （假设为 β ）的值，而其他超参数的值保持不变。如果这个超参数 β 的值对于验证集误差没有明显区别，那么网格搜索相当于进行了两个重复的实验

- 在随机搜索中，两次实验之间，所有的超参数值都不会相等（因为每个超参数的值都是从它们的分布函数中随机采样而来）。因此不大可能会出现两个重复的实验
- 如果 β 超参数与泛化误差无关，那么不同的 β 值：
  - 在网格搜索中，不同 β 值、相同的其他超参数值，会导致大量的重复实验
  - 在随机搜索中，其他超参数值每次也都不同，因此不大可能出现两个重复的实验（除非所有的超参数都与泛化误差无关）

4.5 基于模型的超参数优化

1. 超参数搜索问题可以转化为一个优化问题：
   - 决策变量是超参数
   - 优化的代价是超参数训练出来的模型在验证集上的误差
   - 在简化的设定下，可以计算验证集上可导的误差函数关于超参数的梯度，然后基于这个梯度进行更新
2. 实际上，大多数问题中，超参数的梯度是不可用的
   - 一方面可能是因为因为高额的计算代价和存储成本。因为你需要计算非常多的超参数才能获得一系列的超参数的导数（这就要求你运行非常多轮次的算法）
   - 另一方面可能是因为验证集误差在超参数上本质不可导（如超参数时离散值的情况）
3. 为了弥补超参数梯度的缺失，我们可以使用贝叶斯回归模型来估计每个超参数的验证集误差的期望和该期望的不确定性（贝叶斯回归模型不需要使用梯度）
   - 目前无法明确的确定：贝叶斯超参数优化是否有效。实际上，它有时表现得像一个人类专家，但是有时候又发生灾难的失误。目前该方法还不够成熟或可靠
4. 大部分超参数优化算法比随机搜索更复杂，并且具有一个共同的缺点：在获取任何有效的超参数信息之前，你必须完整运行整个训练过程
   - 这种做法是相当低效的。实际上人类手动搜索之前，可以很早就判断某些超参数组合是否是完全病态的

五、调试策略

1. 当一个机器学习系统效果不好时，很难判断效果不好的原因是算法本身，还是算法实现错误。由于各种原因，机器学习系统很难调试
   - 大多数情况下，我们不能提前知道算法的行为
     - 实际上，使用机器学习的出发点是：机器学习会发现一些我们无法发现的有用的行为
     - 如果我们在一个新的分类任务上训练一个神经网络，它达到了 5% 的测试误差。我们无法直接知道：这是期望的结果，还是次优的结果？
   - 大部分机器学习模型有多个自适应的部分
     - 如果某个部分失效了，那么其他部分仍然可以自适应这种状况，并获得大致可接受的性能
     - 比如在多层神经网络中，假设我们在梯度下降中对偏置更新时犯了一个错误 $b=b-\alpha$（错误原因：没有使用梯度。其中 α 为学习率）。这个错误导致偏置在学习过程中不断减小。但是只是检查模型输出的话，这个错误并不是显而易见的。因为权重 $\mathbf{W}$ 可以自适应的补偿偏置 b 的错误
2. 大部分神经网络的调试策略都是解决上述两个难点的一个或者两个
   - 我们可以设计一种足够简单的情况，能够提前得到正确结果，判断模型预测是否与之相符
   - 我们可以设计一个测试（类似于 `UnitTest`），独立检查神经网络实现的各个部分
3. 一些重要的调试方法如下：
   - 可视化计算中模型的行为：直接观察机器学习模型运行机器任务，有助于确定其达到的量化性能数据是否看上去合理

        - 仅仅评估模型性能可能是最具破坏性的错误之一，因为它会使你在系统出问题时误以为系统运行良好
    - 可视化那些最严重的错误：大多数模型能够输出运行任务时的某种置信度（如基于`softmax`函数输出层的分类器给每个类分配了一个概率）。如果某个样本被错误的分类，而且其分类置信度很低，那么可以检查这些样本的问题
    - 根据训练和测试误差检测软件：通常我们很难确定底层软件是否正确实现，而测试和训练误差提供了一些线索
        - 如果训练误差较低，而测试误差较高，那么很可能是由于算法过拟合。也有可能是软件错误，导致测试误差未能正确地度量
        - 如果训练误差和测试误差都很高，那么很难确定是由于软件错误，还是由于算法模型的欠拟合。此时需要进一步测试，如下所述
    - 监控激活函数值和梯度的直方图：可视化激活函数值和梯度的统计量往往是有用的。
        - 隐单元的激活函数值告诉我们：该单元是否饱和，或者它们饱和的频率如何（如多久关闭一次，是否永远关闭，以及采用双曲正切单元时饱和的程度）
        - 梯度的统计量告诉我们：梯度是否快速的增长或者快速的消失
        - 参数的梯度的量级和参数的量级也有意义：我们希望参数在一个小批量更新中变化的幅度是参数量值的 1% 这样的级别（而不是 50% 或者 0.0001%，这两者要么导致参数移动太快，要么太慢）
        - 如果数据是稀疏的（如自然语言），则有些参数可能是很少更新的，检测时需要注意
4. 许多深度学习算法为每一步的结果产生了某种保证。其中包括：
    - 目标函数值确保在算法的迭代步中不会增加
    - 某些变量的导数在算法的每一步中都是 0
    - 所有变量的梯度在收敛时，会变为 0

    > 由于计算机存储浮点数的舍入误差，这些条件可能会有误差

    如果违反了这些保证，那么一定是软件错误

5.1 梯度检查

1. 比较反向传播导数和数值导数：如果我们正在使用一个软件框架或者库，那么必须定义`bprop`方法，常见的错误原因是未能正确的实现梯度表达。验证该错误的一个方法是：比较实现的自动求导和通过有限差分计算的导数

    $$\frac{df(x)}{dx}=\frac{f(x+h)-f(x)}{h}$$

    或者采用中心差分法：

    $$\frac{df(x)}{dx}=\frac{f(x+h)-f(x-h)}{2h}$$

    其中 ϵ 必须足够大，从而确保不会由于有限精度问题产生舍入误差
    - 如果梯度是雅克比矩阵（输出为向量，输入也是向量），则我们可以多次使用有限差分法来评估所有的偏导数
    - 通常推荐使用中心差分法，因为根据泰勒展开，有：

    $$\frac{f(x+h)-f(x)}{h}=f'(x)+O(h)$$
    $$\frac{f(x+h)-f(x-h)}{2h}=f'(x)+O(h^2)$$

使用中心差分法的误差更小。

2. 理论上进行梯度检查很简单，就是把解析的梯度和数值计算的梯度进行比较。但是实际操作过程中，这个过程复杂且容易出错。
3. 使用中心化公式来计算数值梯度。

   通常我们计算数值梯度时，采用：

$$\frac{df(x)}{dx} = \frac{f(x+h) - f(x)}{h}$$

   其中 h 为一个很小的数字，在实践过程中近似为 10^{-5}。但是实践中证明，使用中心化公式效果更好：

$$\frac{df(x)}{dx} = \frac{f(x+h) - f(x-h)}{2h}$$

   - 缺点：在检查梯度的每个维度时，需要计算两次损失函数
   - 优点：梯度的数值求解结果会准确的多。中心化公式的近似误差为 $O(h^2)$，而前一个公式的近似误差为 $O(h)$
4. 使用相对误差来比较。对于数值梯度 f'_n 和解析梯度 f'_a，通常都是比较其相对误差而不是绝对误差来判断数值计算得到的梯度是否正确：

$$s = \frac{|f'_a - f'_n|}{\max(|f'_a|, |f'_n|)}$$

   - 如果相对误差 $s > 10^{-2}$，则通常意味着求解梯度可能出错
   - 如果相对误差 $10^{-4} > s$，则对于有不可导点的目标函数时OK的，但是对于使用了`tanh/softmax`的目标函数，还是有点高
   - 如果相对误差 $10^{-7} > s$，则求解梯度结果正确
   - 因为网络越深，相对误差就越高。因此对于一个 10 层网络的输入数据做梯度检查，那么 10^{-2} 的相对误差可能就OK了，因为误差一直在累积
5. 使用双精度类型。当使用单精度浮点数来进行梯度检查时，即使梯度计算过程正确，相对误差也会很高（如 10^{-2} ）
6. 梯度检查时，一个导致不准确的原因是：不可导点。
   - 不可导点是目标函数不可导的部分，由于 relu 等函数的引入导致。
   - 解决方法是：使用更少的数据点。
     - 因为数据点越少，不可导点就越少，所以在计算有限差值近似时，横跨不可导点的几率就越小。
     - 另外如果你的梯度检查对 2-3 个数据点都有效，那么基本上对于整个 batch 数据进行梯度检查也没有问题。所以用少量的数据点，能让梯度检查更迅速有效
7. 谨慎设置步长 h
   - 并不是 h 越小越好。因为当 h 特别小时，可能会遇到数值精度问题。如果梯度检查不通过，可以尝试将 h 调到 10^{-4} 或者 10^{-6}
8. 建议让网络预热一小段时间，等到损失函数开始下降之后在进行梯度检查
   - 第一次迭代就开始梯度检查的危险在于：此时可能处于不正常的边界情况，从而掩盖了梯度没有正确实现的事实。因为梯度检查是在参数空间中的一个特定的、单独的点上进行的。即使在该点上检查成功，也不能保证梯度实现是正确的。
9. 不要让正则化吞没数据
   - 通常损失函数为：数据损失+正则化损失之和。某些情况下，损失函数的梯度主要来源于正则化部分，此时就会掩盖掉数据损失梯度的不正确实现

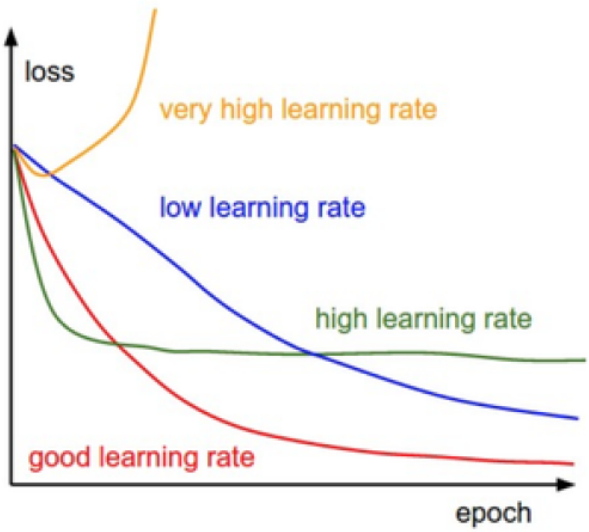
- 建议先关掉正则化对数据损失做单独检查，然后对正则化做单独检查

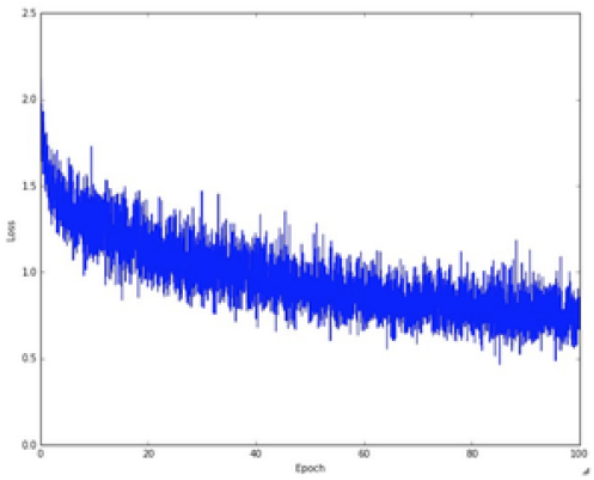
10. 关闭 dropout 和数据 augmentation
    - 梯度检查时，关闭网络中任何不确定的效果的操作，如随机 dropout，随机数据扩展。不然它们会在计算数值梯度时导致巨大误差
11. 检查少量的维度。由于梯度可以有上百万的参数，此时只能检查其中一些维度然后假设其他维度是正确的
    - 确保在所有不同的参数中都抽取一部分来梯度检查。比如 $\mathbf{W}_1, \mathbf{W}_2, \cdots$ 这些权重矩阵中，每个矩阵抽取一部分来检查。
    - 不要将所有参数拼接成一个巨大的向量，然后从这个向量中随机抽取一些维度来检查
12. 只有在调试过程中进行梯度检验。不要在训练过程中执行梯度检验，因为非常耗计算资源。

5.2 学习过程检查

1. 拟合极小的数据集：

   当训练集上有很大的误差时，我们需要确定问题是欠拟合，还是软件错误。尝试对一个小数据子集进行训练，然后确保能达到 0 的损失值。
   - 通常即使是极小模型也能保证很好的拟合一个足够小的数据集。如只有一个样本的分类数据，它可以通过正确设置输出层的偏置来拟合
   - 如果分类器不能正确标定一个单样本组成的训练集、自编码器无法成功再现一个单独的样本、生成模型无法一致的生成一个单独的样本，那么很有可能是由于软件错误
   - 这种测试可以推广到少量样本的小数据集上，不一定是只有一个样本的数据集
   - 进行这个训练时，最好让正则化强度为0，不然它会阻止得到0的损失
   - 如果不能通过这个检验，那么训练过程有问题。此时进行整个数据集的训练是没有意义的；如果能通过这个检验，那么也不保证训练过程没问题
2. 在训练神经网络时，应该跟踪多个重要数值
   - 这些数值输出的图表是观察训练进程的一个窗口，是直观理解不同超参数设置效果的工具
   - 通常 x 轴都是以周期 epoch 为单位，它衡量了训练中每个样本数据都被观察过次数的期望
     - 一个周期表示每个样本数据都被观察过了一次
     - 不要使用迭代次数。因为迭代次数与数据的 batch size 有关。而 batch size 可以任意设置
3. 第一个要跟踪的数值就是损失函数





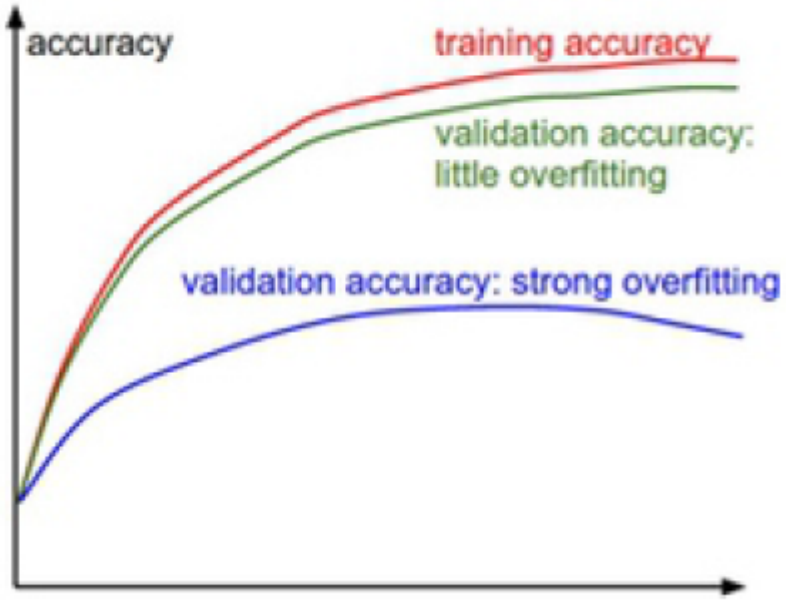
从左图中可见：

- 过低的学习率（蓝色曲线）导致算法的损失函数下降近似线性的
- 稍高一些的学习率（红色曲线） 会导致算法的损失函数看起来呈几何指数下降
- 更高一些的学习率（绿色曲线）会让损失函数下降的更快，但是它使得损失函数停留在一个较高的水平位
- 异常高的学习率（黄色曲线）会让损失函数上升

从右图可见：

- 损失函数值曲线看起来比较合理，但是 batch size 可能有点小。因为损失值的噪音很大
- 损失值的震荡程度和 batch size 有关。当 batch size=1 时，震荡会相对较大；而 batch size 为整个数据集大小时，震荡最小。因为每个梯度更新都是单调的优化损失函数

4. 在训练分类器时，第二个要跟踪的就是验证集和训练集的准确率。从该图标可以获知模型过拟合的程度



- 训练集准确率和验证集准确率之间的空隙指明了模型过拟合的程
- 蓝色的验证集曲线相对于训练集，验证集的准确率低了很多。这说明模型有很强的过拟合。此时应该增大正则化强度或者收集更多的数据
- 绿色的验证集曲线和训练集曲线如影随形，说明你的模型容量还不够大，发生了欠拟合。此时应该通过增加参数数量让模型更大一些。

5. 另一个应该跟踪的数值是：权重更新的相对值

假设权重为 $\mathbf{W}$，更新的增量为 $\Delta \mathbf{W} = -\eta * \nabla_{\mathbf{W}} L$。 η 为学习率。那么权重更新的相对值为：

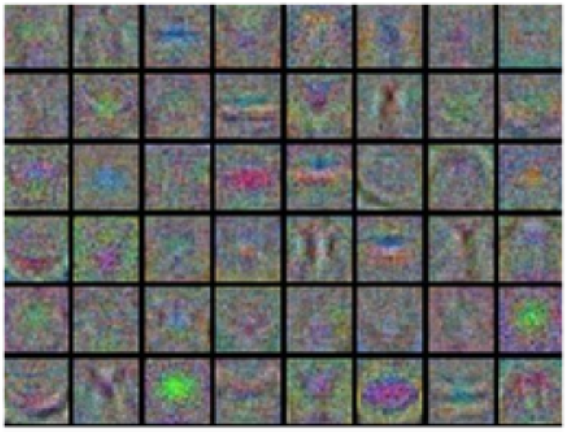
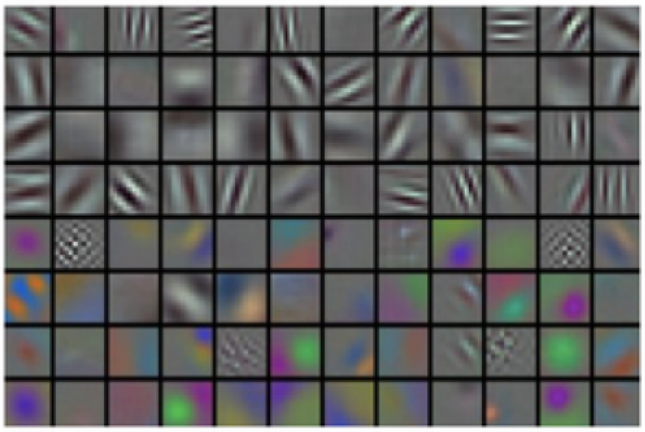
$$s = \frac{||\Delta \mathbf{W}||_F}{||\mathbf{W}||_F}$$

- 这里的梯度是一个更新的梯度。如果多个 batch ，则每个 batch 更新一次就计算一次
- 这个比例 s 应该在 10^{-3} 左右。如果更低，说明学习率可能太小；如果更高，说明学习率可能太高

6. 一个不正确的初始化可能让学习过程变慢，甚至停止。这个问题可以比较简单的诊断出来：

- 输出网络中所有层的激活数据和梯度分布的柱状图
- 如果看到任何奇怪的分布，都不是好兆头。如；对于使用 tanh 的神经元，我们应该看到激活数据的值在整个 [-1,1] 区间都有分布。如果看到神经元的输出都是0，或者都在饱和部分，那么就有问题

7. 如果数据是图像像素数据，那么把第一层特征可视化会有帮助：

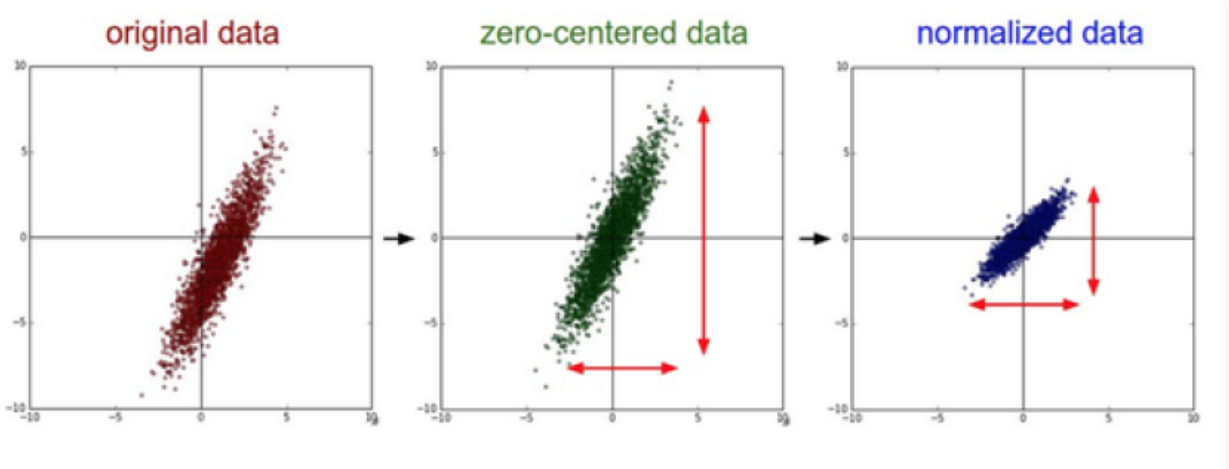
- 左图的特征充满了噪音，这暗示网络可能出现问题：网络没有收敛 、学习率设置不当、正则化惩罚的权重过低
- 右图特征不错，平滑、干净、种类繁多。说明训练过程良好

六、示例：数字识别系统

1. 项目开始于性能度量的选择以及这些度量的期望。
   - 总的原则是：度量的选择要符合项目的业务目标
   - 因为地图只有高准确率时才有价值，所以该系统要求达到 98% 的准确率（达到人类水平）
   - 为了达到这个级别的准确率，系统牺牲了覆盖率。因此在保持准确率 98% 的前提下，覆盖率成了项目的主要性能度量
2. 选择基准系统：对于视觉任务而言，基准系统是带有 `ReLU` 单元的卷积网络

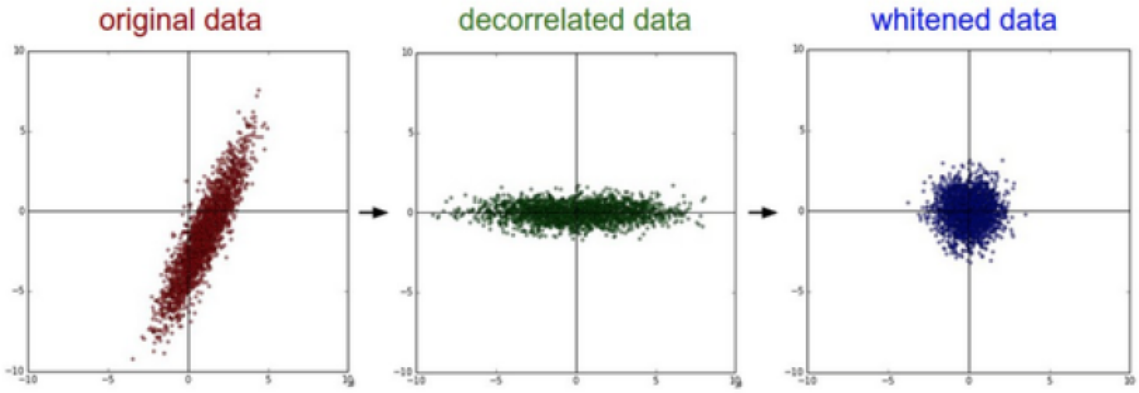
七、数据预处理

1. 常见数据预处理方式：（红色的线指出各维度的数值范围）



   - 均值减法：对数据中每个独立特征减去均值，集合上理解为：在每个维度上，都将数据云的中心迁移到原点
   - 归一化：将所有维度都做归一化：
     - 第一种做法：先对数据做零中心化处理，然后每个维度都除以标准差
     - 第二种做法：每个维度都做归一化，使得每个维度最大和最小值都是 1 和 -1
   - PCA 降维：取得数据的主成分，可以对数据去除相关性

- 白化 `whitening`：先对数据进行旋转（旋转的矩阵就是 SVD 分解中的 U矩阵），然后对每个维度除以特征值（为防止分母为0，通常加一个很小的值作为平滑系数）来对数值范围进行归一化
  - 如果数据服从多变量的高斯分布，则白化之后，数据的分布是一个均值为零，且协方差相等的矩阵
  - 该变换的缺点是：可能会放大数据中的噪声。因为它将所有维度都拉伸到相同的维度，这包括了那些大多数是噪声的维度。这个问题可以采用更强的平滑系数来解决



- 实际在神经网络中，并不会采用 PCA 和白化。

2. 任何预处理策略都只能在训练集的数据上进行，然后再应用到验证集或测试集上。
   - 如数据均值：首先分成训练集、验证集、测试集，从训练集中求数据的均值。然后训练集、验证集、测试集中的数据减去这个均值。 而不是减去测试集均值或者验证集均值
3. 激活函数：当前推荐使用 ReLU 激活函数
4. Batch Normalization：让数据在通过激活函数之前，添加一个 batch normalization

八、变量初始化

8.1 权重初始化

1. 权重一定不能全零初始化。因为这会导致神经元在前向传播中计算出同样的输出，然后在反向传播中计算出同样的梯度，从而进行同样的权重更新。这就产生了大量对称性神经元。
2. 通常采用小随机数初始化，通过这样来打破对称性。至于使用高斯分布还是均匀分布，对结果影响很小。
   - 之所以用小的随机数，是因为：如果网络中存在 `tanh` 或者 `sigmoid` 激活函数，或者网络的输出层为 `sigmoid` 等单元，则它们的变量值必须很小。

     如果使用较大的随机数，则很有可能这些单元会饱和，使得梯度趋近于零。这意味着基于梯度下降的算法推进的很慢，从而学习很慢。
   - 如果网络中不存在 `sigmoid/tanh` 等激活函数，网络的输出层也不是 `sigmoid` 等单元，则可以使用较大的随机数初始化。
3. 通常使用 $\frac{1}{\sqrt{n}}$ 来校准权重初始化标准差。随着输入数据的增长，随机初始化的神经元的输出数据的分布中的方差也在增大。

   假设权重 $\vec{\mathbf{w}}$ 和输入 $\vec{\mathbf{x}}$ 之间的内积为 $s=\sum_i^n w_i x_i$（不考虑非线性激活函数）。我们检查 s 的方差

$$Var(s) = Var(\sum_i^n w_i x_i) = \sum_i^n Var(w_i x_i)$$

$$= \sum_i^n [E(w_i)]^2 Var(x_i) + E[(x_i)]^2 Var(w_i) + Var(x_i)Var(w_i)$$

$$= \sum_i^n Var(x_i)Var(w_i) = (nVar(w))Var(x)$$

其中假设输入和权重的平均值都是0，因此有 $E[x_i] = E[w_i] = 0$。同时假设所有的 w_i 服从同样的分布 $f_w(w)$，假设所有的 x_i 服从同样的分布 $f_x(x)$

- 要使得输出 s 和输入 x 具有同样的方差，则必须保证每个权重 w 的方差是 $\frac{1}{n}$ 。则如果权重通过一个标准高斯分布初始化，则需要将标准差除以 $\sqrt{n}$
- 在`Glorot`的论文中，他推荐初始化公式为： $Var(w) = \frac{2}{n_{in}+n_{out}}$。其中 n_{in}, n_{out} 为输入和输出的数量。

8.2 偏置初始化

1. 通常将偏置初始化为0。这是因为随机小数值权重已经打破了对称性

九、结构设计

1. 对于任何给定的问题，很难提前去预测到底需要多深的神经网络。

   一个常规的做法是：首先尝试使用逻辑回归（可以视为没有隐层的神经网络）。然后尝试单隐层的神经网络、双隐层的神经网络....

   这种做法将隐层的数量看做是一个超参数。通过交叉验证来确定该超参数的值。