第十五章 正则化

15.1 什么是正则化？

15.2 正则化原理？

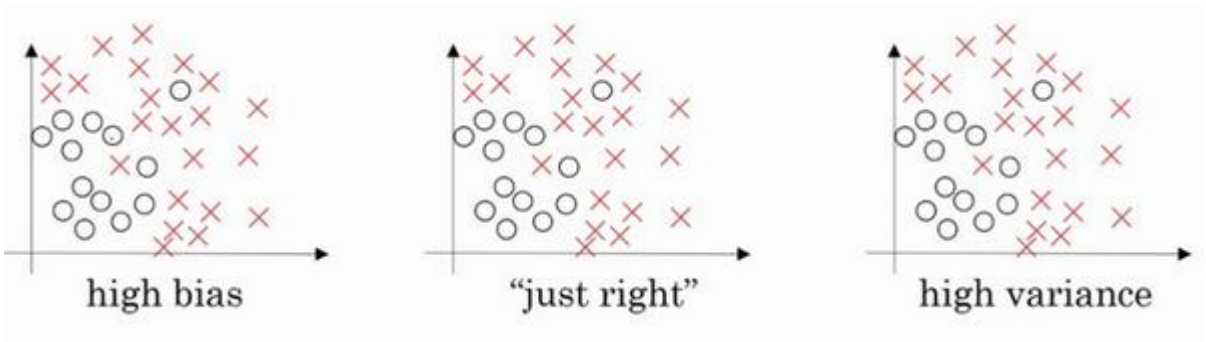
15.3 为什么要正则化？

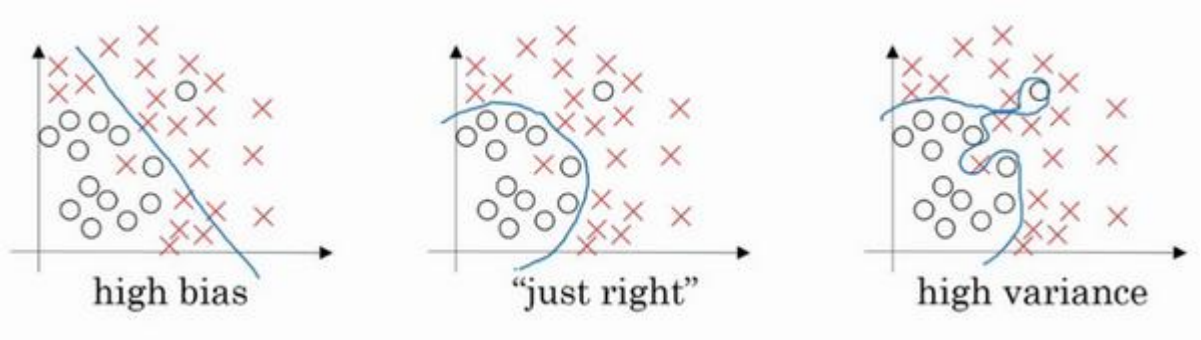
除了正则化和随机失活（dropout）正则化，还有几种方法可以减少神经网络中的过拟合:

深度学习可能存在过拟合问题——高方差，有两个解决方法，一个是正则化，另一个是准备更多的数据，这是非常可靠的方法，但你可能无法时时刻刻准备足够多的训练数据或者获取更多数据的成本很高，但正则化通常有助于避免过拟合或减少你的网络误差。

如果你怀疑神经网络过度拟合了数据，即存在高方差问题，那么最先想到的方法可能是正则化，另一个解决高方差的方法就是准备更多数据，这也是非常可靠的办法，但你可能无法时时准备足够多的训练数据，或者，获取更多数据的成本很高，但正则化有助于避免过度拟合，或者减少网络误差，下面我们就来讲讲正则化的作用原理。

15.4 为什么正则化有利于预防过拟合？





左图是高偏差，右图是高方差，中间是 Just Right，这几张图我们在前面课程中看到过。

15.5 为什么正则化可以减少方差？

15.6 L2 正则化的理解？

15.7 理解 dropout 正则化

Dropout 可以随机删除网络中的神经单元，他为什么可以通过正则化发挥如此大的作用呢？

直观上理解：不要依赖于任何一个特征，因为该单元的输入可能随时被清除，因此该单元通过这种方式传播下去，并为单元的四个输入增加一点权重，通过传播所有权重，dropout 将产生收缩权重的平方范数的效果，和之前讲的 L2 正则化类似；实施 dropout 的结果实它会压缩权重，并完成一些预防过拟合的外层正则化；L2 对不同权重的衰减是不同的，它取决于激活函数倍增的大小。

15.8 有哪些 dropout 正则化方法？

15.8 如何实施 dropout 正则化

如何实施 dropout 呢？方法有几种，接下来我要讲的是最常用的方法，即 inverted dropout（反向随机失活），出于完整性考虑，我们用一个三层（）网络来举例说明。编码中会有很多涉及到 3 的地方。我只举例说明如何在某一层中实施 dropout。

15.9 Python 实现 dropout 正则化

15.10 L2 正则化和 dropout 有什么不同？

dropout 的功能类似于正则化，与正则化不同的是应用方式不同会带来一点点小变化，甚至更适用于不同的输入范围。

第二个直观认识是，我们从单个神经元入手，如图，这个单元的工作就是输入并生成一些有意义的输出。通过 dropout，该单元的输入几乎被消除，有时这两个单元会被删除，有时会删除其它单元，就是说，我用紫色圈起来的这个单元，它不能依靠任何特征，因为特征都有可能被随机清除，或者说该单元的输入也都可能被随机清除。我不愿意把所有赌注都放在一个节点上，不愿意给任何一个输入加上太多权重，因为它可能会被删除，因此该单元将通过这种方式积极地传播开，并为单元的四个输入增加一点权重，通过传播所有权重，dropout 将产生收缩权重的平方范数的效果，和我们之前讲过的正则化类似，实施 dropout 的结果是它会压缩权重，并完成一些预防过拟合的外层正则化。

事实证明，dropout 被正式地作为一种正则化的替代形式，对不同权重的衰减是不同的，它取决于倍增的激活函数的大小。

总结一下，dropout 的功能类似于正则化，与正则化不同的是，被应用的方式不同，dropout 也会有所不同，甚至更适用于不同的输入范围。

15.11 dropout 有什么缺点？

dropout 一大缺点就是代价函数 J 不再被明确定义，每次迭代，都会随机移除一些节点，如果再三检查梯度下降的性能，实际上是很难进行复查的。定义明确的代价函数 J 每次迭代后都会下降，因为我们所优化的代价函数 J 实际上并没有明确定义，或者说在某种程度上很难计算，所以我们失去了调试工具来绘制这样的图片。我通常会关闭 dropout 函数，将 keep-prob 的值设为 1，运行代码，确保 J 函数单调递减。然后打开 dropout 函数，希望在 dropout 过程中，代码并未引入 bug。我觉得你也可以尝试其它方法，虽然我们并没有关于这些方法性能的数据统计，但你可以把它们与 dropout 方法一起使用。

SCUTan

15.12 其他正则化方法？

一.数据扩增

假设你正在拟合猫咪图片分类器，如果你想通过扩增训练数据来解决过拟合，但扩增数据代价高，而且有时候我们无法扩增数据，但我们可以通过添加这类图片来增加训练集。例如，水平翻转图片，并把它添加到训练集。所以现在训练集中有原图，还有翻转后的这张图片，所以通过水平翻转图片，训练集则可以增大一倍，因为训练集有冗余，这虽然不如我们额外收集一组新图片那么好，但这样做节省了获取更多猫咪图片的花费。

除了水平翻转图片，你也可以随意裁剪图片，这张图是把原图旋转并随意放大后裁剪的，仍能辨别出图片中的猫咪。

通过随意翻转和裁剪图片，我们可以增大数据集，额外生成假训练数据。和全新的，独立的猫咪图片数据相比，这些额外的假的数据无法包含像全新数据那么多的信息，但我们这么做基本没有花费，代价几乎为零，除了一些对抗性代价。以这种方式扩增算法数据，进而正则化数据集，减少过拟合比较廉价。

像这样人工合成数据的话，我们要通过算法验证，图片中的猫经过水平翻转之后依然是猫。大家注意，我并没有垂直翻转，因为我们不想上下颠倒图片，也可以随机选取放大后的部分图片，猫可能还在上面。

对于光学字符识别，我们还可以通过添加数字，随意旋转或扭曲数字来扩增数据，把这些数字添加到训练集，它们仍然是数字。为了方便说明，我对字符做了强变形处理，所以数字 4 看起来是波形的，其实不用对数字 4 做这么夸张的扭曲，只要轻微的变形就好，我做成这样是为了让大家看的更清楚。实际操作的时候，我们通常对字符做更轻微的变形处理。因为这几个 4 看起来有点扭曲。所以，数据扩增可作为正则化方法使用，实际功能上也与正则化相似。

二.early stopping

还有另外一种常用的方法叫作 early stopping，运行梯度下降时，我们可以绘制训练误差，或只绘制代价函数 J 的优化过程，在训练集上用 0-1 记录分类误差次数。呈单调下降趋势，如图。

因为在训练过程中，我们希望训练误差，代价函数 J 都在下降，通过 early stopping，我们不但可以绘制上面这些内容，还可以绘制验证集误差，它可以是验证集上的分类误差，或验证集上的代价函数，逻辑损失和对数损失等，你会发现，验证集误差通常会先呈下降趋势，然后在某个节点处开始上升，early stopping 的作用是，你会说，神经网络已经在这个迭代过程中表现得很好了，我们在此停止训练吧，得到验证集误差，它是怎么发挥作用的？

在机器学习中，超级参数激增，选出可行的算法也变得越来越复杂。我发现，如果我们用一组工具优化代价函数，机器学习就会变得更简单，在重点优化代价函数时，你只需要留意和，的值越小越好，你只需要想办法减小这个值，其它的不用关注。然后，预防过拟合还有其他任

务，换句话说就是减少方差，这一步我们用另外一套工具来实现，这个原理有时被称为“正交化”。思路就是在一个时间做一个任务，后面课上我会具体介绍正交化，如果你还不了解这个概念，不用担心。

但对我来说 early stopping 的主要缺点就是你不能独立地处理这两个问题，因为提早停止梯度下降，也就是停止了优化代价函数，因为现在你不再尝试降低代价函数 J，所以代价函数的值可能不够小，同时你又希望不出现过拟合，你没有采取不同的方式来解决这两个问题，而是用一种方法同时解决两个问题，这样做的结果是我要考虑的东西变得更复杂。

如果不用 early stopping，另一种方法就是正则化，训练神经网络的时间就可能很长。我发现，这导致超级参数搜索空间更容易分解，也更容易搜索，但是缺点在于，你必须尝试很多正则化参数的值，这也导致搜索大量值的计算代价太高。

Early stopping 的优点是，只运行一次梯度下降，你可以找出的较小值，中间值和较大值，而无需尝试正则化超级参数的很多值。

如果你还不能完全理解这个概念，没关系，下节课我们会详细讲解正交化，这样会更好理解。

虽然正则化有缺点，可还是有很多人愿意用它。吴恩达老师个人更倾向于使用正则化，尝试许多不同的值，假设你可以负担大量计算的代价。而使用 early stopping 也能得到相似结果，还不用尝试这么多值。

这节课我们讲了如何使用数据扩增，以及如何使用 early stopping 降低神经网络中的方差或预防过拟合。

SCU-lan