深度学习入门

基于Python的理论与实现

Stephen CUI

March 14, 2023

Chapter 1

与学习相关的技巧

本章将介绍神经网络的学习中的一些重要观点，主题涉及寻找最优权重参数的最优化方法、权重参数的初始值、超参数的设定方法等。此外，为了应对过拟合，本章还将介绍权值衰减、Dropout等正则化方法，并进行实现。

1.1 参数的更新

神经网络的学习的目的是找到使损失函数的值尽可能小的参数。这是寻找最优参数的问题，解决这个问题的过程称为**最优化**（optimization）。遗憾的是，神经网络的最优化问题非常难。这是因为参数空间非常复杂，无法轻易找到最优解（无法使用那种通过解数学式一下子就求得最小值的方法）。而且，在深度神经网络中，参数的数量非常庞大，导致最优化问题更加复杂。

使用参数的梯度，沿梯度方向更新参数，并重复这个步骤多次，从而逐渐靠近最优参数，这个过程称为**随机梯度下降法**（stochastic gradient descent），简称**SGD**。

1.1.1 SGD

用数学式可以将SGD写成如下式：

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial L}{\partial \boldsymbol{W}}$$

SGD是朝着梯度方向只前进一定距离的简单方法。

1.1.2 SGD的缺点

虽然SGD简单，并且容易实现，但是在解决某些问题时可能没有效率。

$$z = \frac{1}{20}x^2 + y^2$$

上式表示的函数是向 x 轴方向延伸的“碗”状函数。

SGD的缺点是，如果函数的形状非均向（anisotropic），比如呈延伸状，搜索的路径就会非常低效。因此，我们需要比单纯朝梯度方向前进的SGD更聪明的方法。SGD低效的根本原因是，梯度的方向并没有指向最小值的方向。

1.1.3 Momentum

$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \eta \frac{\partial L}{\partial \boldsymbol{W}} \tag{1.1a}$$

$$\boldsymbol{W} \leftarrow \boldsymbol{W} + \boldsymbol{v} \tag{1.1b}$$

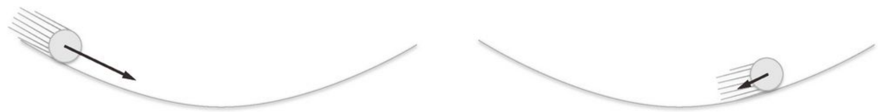

Figure 1.1: Momentum-The ball rolls on an incline

这里新出现了一个变量$\boldsymbol{v}$，对应物理上的速度。 Equation 1.1a表示了物体在梯度方向上受力，在这个力的作用下，物体的速度增加这一物理法则。如Figure 1.1所示，Momentum方法给人的感觉就像是小球在地面上滚动。

式Equation 1.1a中有$\alpha\boldsymbol{v}$这一项。在物体不受任何力时，该项承担使物体逐渐减速的任务（α设定为0.9之类的值），对应物理上的地面摩擦或空气阻力。

1.1.4 AdaGrad

在关于学习率的有效技巧中，有一种被称为**学习率衰减**（learning rate decay）的方法，即随着学习的进行，使学习率逐渐减小。实际上，一开始“多” 学，然后逐渐“少”学的方法，在神经网络的学习中经常被使用。

逐渐减小学习率的想法，相当于将“全体”参数的学习率值一起降低。而AdaGrad进一步发展了这个想法，针对“一个一个”的参数，赋予其“定制”的值。AdaGrad会为参数的每个元素适当地调整学习率，与此同时进行学习。

$$\boldsymbol{h} \leftarrow \boldsymbol{h} + \frac{\partial L}{\partial \boldsymbol{W}} \odot \frac{\partial L}{\partial \boldsymbol{W}} \tag{1.2a}$$

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{1}{\sqrt{\boldsymbol{h}}} \frac{\partial L}{\partial \boldsymbol{W}} \tag{1.2b}$$

式中，$\odot$表示对应矩阵元素的乘法，在更新参数时，通过乘以$\frac{1}{\sqrt{\boldsymbol{h}}}$，就可以调整学习的尺度。这意味着，参数的元素中变动较大（被大幅更新）的元素的学习率将变小。也就是说， 可以按参数的元素进行学习率衰减，使变动大的参数的学习率逐渐减小。

> AdaGrad会记录过去所有梯度的平方和。因此，学习越深入，更新的幅度就越小。实际上，如果无止境地学习，更新量就会变为 0，完全不再更新。为了改善这个问题，可以使用 RMSProp方法。 RMSProp方法并不是将过去所有的梯度一视同仁地相加，而是逐渐地遗忘过去的梯度，在做加法运算时将新梯度的信息更多地反映出来。这种操作从专业上讲，称为“指数移动平均”，呈指数函数式地减小过去的梯度的尺度。

1.1.5 Adam

Adam是2015年提出的新方法。它的理论有些复杂，直观地讲，就是融合了Momentum和AdaGrad的方法。通过组合前面两个方法的优点，有望实现参数空间的高效搜索。此外，进行超参数的“偏置校正”也是Adam的特征。

> Adam会设置3个超参数。一个是学习率（论文中以α出现），另外两个是一次momentum系数β_1和二次momentum系数β_2。根据论文，标准的设定值是β_1为0.9，β_2 为0.999。设置了这些值后，大多数情况下都能顺利运行。

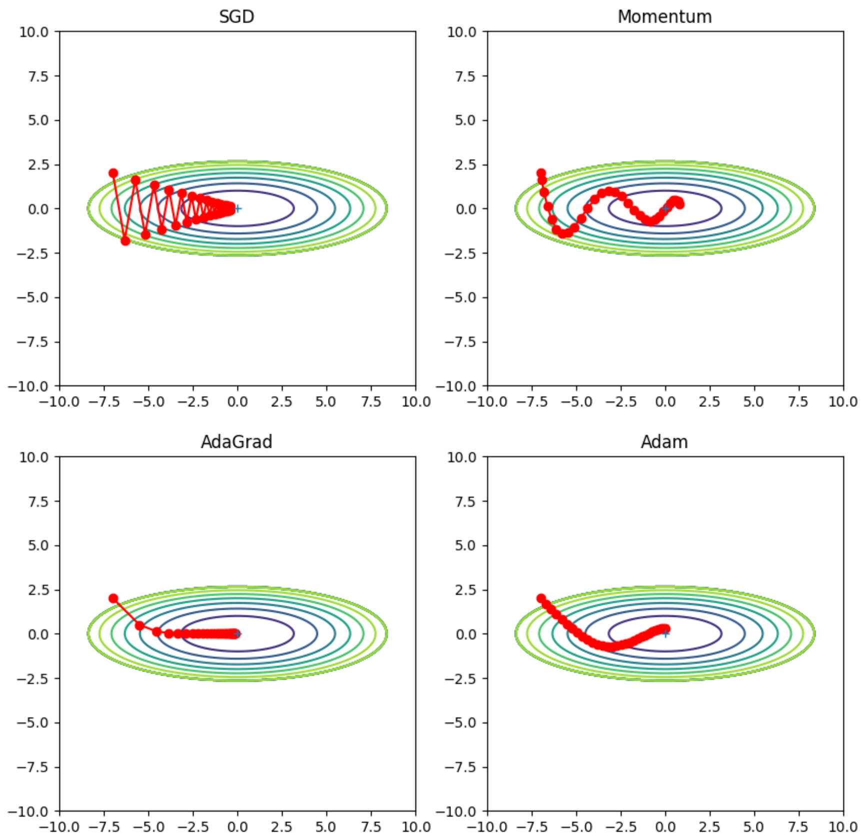

Figure 1.2: Comparison of Optimization Methods

1.1.6 使用哪种更新方法呢

如Figure 1.2所示，根据使用的方法不同，参数更新的路径也不同。只看这个图的话，AdaGrad似乎是最好的，不过也要注意，结果会根据要解决的问题而变。并且，很显然，超参数（学习率等）的设定值不同，结果也会发生变化。

非常遗憾，（目前）并不存在能在所有问题中都表现良好的方法。这4种方法各有各的特点，都有各自擅长解决的问题和不擅长解决的问题。

1.1.7 基于MNIST数据集的更新方法的比较

1.2 权重的初始值

在神经网络的学习中，权重的初始值特别重要。实际上，设定什么样的权重初始值，经常关系到神经网络的学习能否成功。

1.2.1 可以将权重初始值设为0吗

权值衰减(weights decay)就是一种以减小权重参数的值为目的进行学习的方法。通过减小权重参数的值来抑制过拟合的发生。从结论来说，将权重初始值设为0不是一个好主意。事实上，将权重初始值设为 0的话，将无法正确进行学习。

为了防止“权重均一化”（严格地讲，是为了瓦解权重的对称结构），必须随机生成初始值。

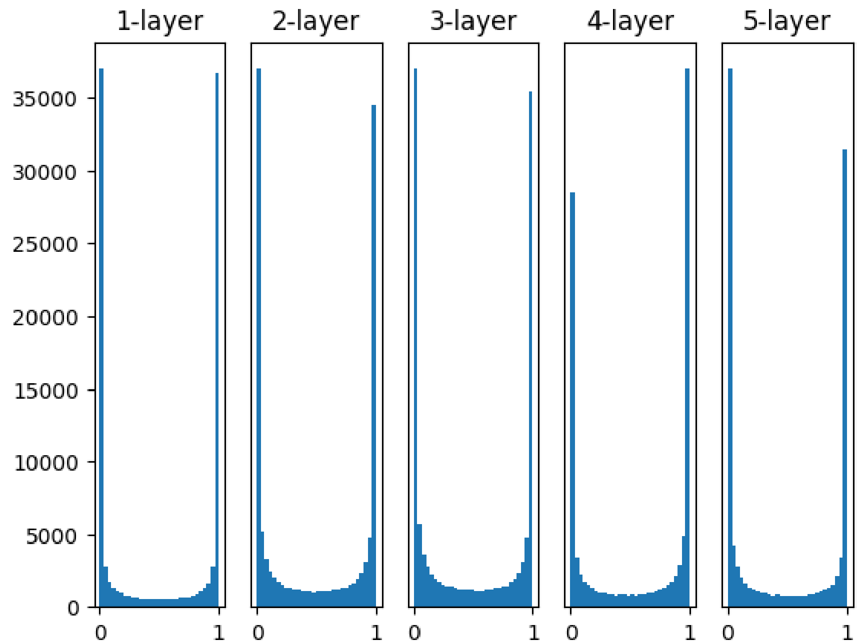

Figure 1.3: The distribution of the activation value of each layer when using a Gaussian distribution with a standard deviation of 1 as the initial weight value

1.2.2 隐藏层的激活值的分布

观察隐藏层的激活值[1]（激活函数的输出数据）的分布，可以获得很多启发。 从Figure 1.3可知，各层的激活值呈偏向0和1的分布。这里使用的sigmoid 函数是S型函数，随着输出不断地靠近0（或者靠近1），它的导数的值逐渐接近0。因此，偏向0和1的数据分布会造成反向传播中梯度的值不断变小，最后消失。这个问题称为梯度消失（gradient vanishing）。层次加深的深度学习中，梯度消失的问题可能会更加严重。

> 各层的激活值的分布都要求有适当的广度。为什么呢？因为通过在各层间传递多样性的数据，神经网络可以进行高效的学习。反过来，如果传递的是有所偏向的数据，就会出现梯度消失或者“表现力受限”的问题，导致学习可能无法顺利进行。

Xavier的论文中，为了使各层的激活值呈现出具有相同广度的分布，推导了合适的权重尺度。推导出的结论是，如果前一层的节点数为n，则初始值使用标准差为$\sqrt{\frac{1}{n}}$ 的分布。

使用Xavier初始值后的结果如Figure 1.5所示。从这个结果可知，越是后面的层，图像变得越歪斜，但是呈现了比之前更有广度的分布。因为各层间传递的数据有适当的广度，所以sigmoid函数的表现力不受限制，有望进行高效的学习。

> Figure 1.5的分布中，后面的层的分布呈稍微歪斜的形状。如果用tanh 函数（双曲线函数）代替sigmoid函数，这个稍微歪斜的问题就能得到改善。实际上，使用 tanh函数后，会呈漂亮的吊钟型分布。tanh 函数和sigmoid函数同是S型曲线函数，但tanh函数是关于原点(0,0) 对称的S型曲

[1]这里我们将激活函数的输出数据称为“激活值”，但是有的文献中会将在层之间流动的数据也称为“激活值”。

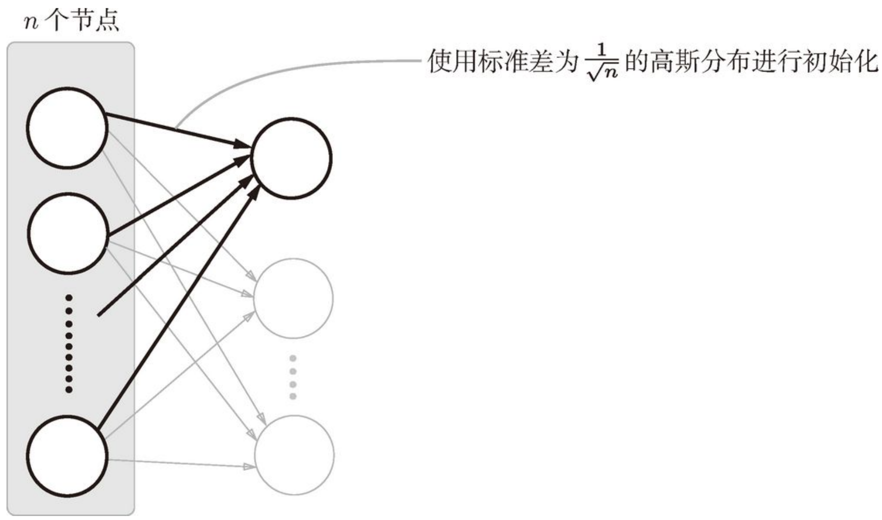

Figure 1.4: Xavier initial value

线，而 sigmoid函数是关于$(x,y)=(0,0.5)$对称的S型曲线。众所周知，用作激活函数的函数最好具有关于原点对称的性质。

1.2.3 ReLU的权重初始值

Xavier 初始值是以激活函数是线性函数为前提而推导出来的。因为 sigmoid函数和 tanh函数左右对称，且中央附近可以视作线性函数，所以适合使用Xavier初始值。但当激活函数使用ReLU时，一般推荐使用ReLU专用的初始值，也就是Kaiming He等人推荐的初始值，也称为“He初始值”。当前一层的节点数为n时，He 初始值使用标准差为$\sqrt{\frac{2}{n}}$ 的高斯分布。

总结一下，当激活函数使用ReLU时，权重初始值使用He初始值，当激活函数为 sigmoid或 tanh等S型曲线函数时，初始值使用Xavier初始值。这是目前的最佳实践。

1.2.4 基于MNIST数据集的权重初始值的比较

这个实验中，神经网络有5层，每层有100个神经元，激活函数使用的是ReLU。从Figure 1.6的结果可知，std = 0.01时完全无法进行学习。这和刚才观察到的激活值的分布一样，是因为正向传播中传递的值很小（集中在0 附近的数据）。因此，逆向传播时求到的梯度也很小，权重几乎不进行更新。相反，当权重初始值为Xavier初始值和He初始值时，学习进行得很顺利。并且，我们发现He初始值时的学习进度更快一些。

1.3 Batch Normalization

在上一节，我们观察了各层的激活值分布，并从中了解到如果设定了合适的权重初始值，则各层的激活值分布会有适当的广度，从而可以顺利地进行学习。那么，为了使各层拥有适当的广度，“强制性”地调整激活值的分布会怎样呢？实际上，Batch Normalization 方法就是基于这个想法而产生的。

1.3.1 Batch Normalization 的算法

什么Batch Norm这么惹人注目呢？因为Batch Norm有以下优点。

- 可以使学习快速进行（可以增大学习率）。

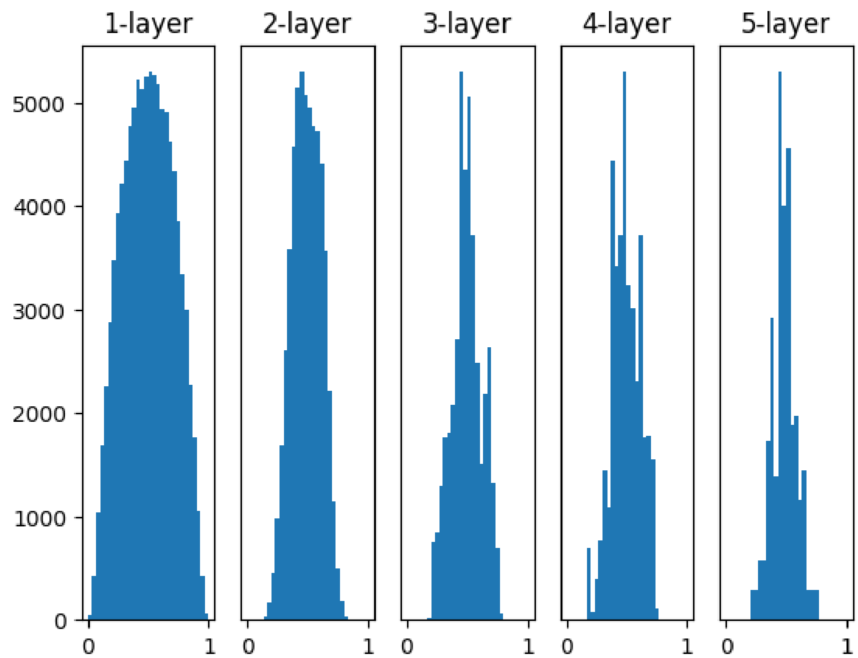

Figure 1.5: Distribution of activation values of each layer when using Xavier initial value as weight initial value

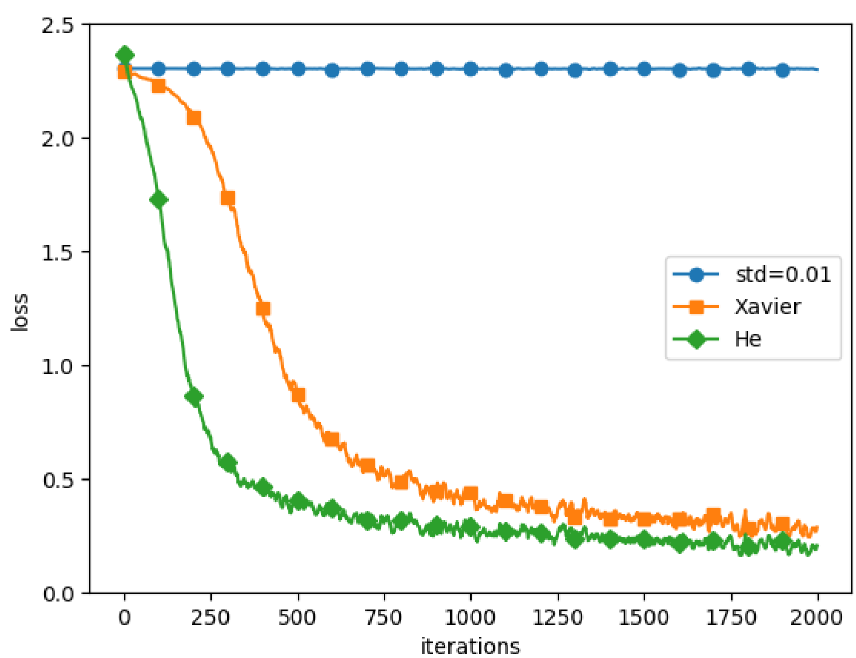

Figure 1.6: Comparison of weight initial values based on MNIST dataset

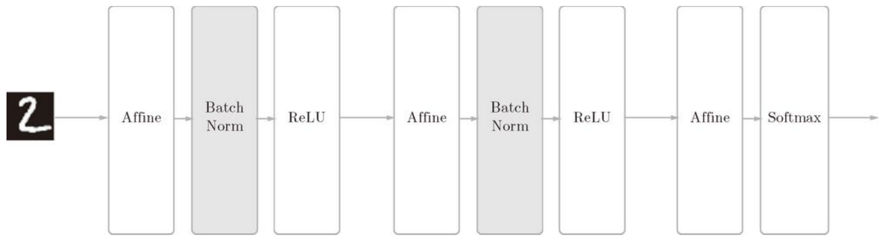

Figure 1.7: An example of a neural network using Batch Normalization

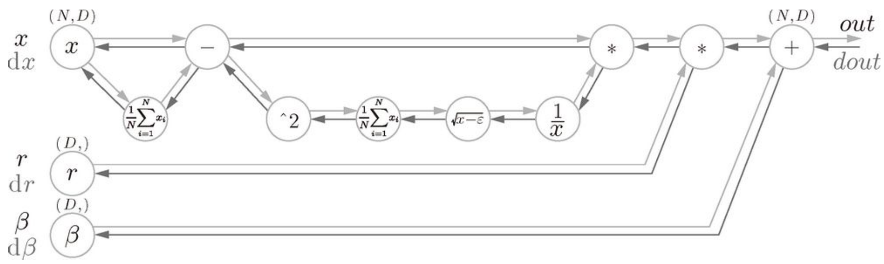

Figure 1.8: Computational graph of Batch Normalization

- 不那么依赖初始值（对于初始值不用那么神经质）。
- 抑制过拟合（降低Dropout等的必要性）。

考虑到深度学习要花费很多时间，第一个优点令人非常开心。另外，后两点也可以帮我们消除深度学习的学习中的很多烦恼。

Batch Norm 的思路是调整各层的激活值分布使其拥有适当的广度。为此，要向神经网络中插入对数据分布进行正规化的层，即Batch Normalization层（下文简称Batch Norm层），如Figure 1.7所示。Batch Norm，顾名思义，以进行学习时的mini-batch为单位，按mini-batch进行正规化。具体而言，就是进行使数据分布的均值为0、方差为1的正规化。用数学式表示的话，如下所示：

$$\mu_B \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$$

$$\sigma_B^2 \leftarrow \frac{1}{m-1}\sum_{i=1}^{m}(x_i-\mu_B)^2$$

$$\hat{x}_i \leftarrow \frac{x_i-\mu_B}{\sqrt{\sigma_B^2+\varepsilon}}$$

Batch Norm层会对正规化后的数据进行缩放和平移的变换，用数学式可以如下表示：

$$y_i \leftarrow \gamma\hat{x}_i+\beta$$

这里，γ和β是参数。一开始$\gamma=1$，$\beta=0$，然后再通过学习调整到合适的值。

如果使用Figure 1.8的计算图来思考的话，Batch Norm的反向传播或许也能比较轻松地推导出来。Frederik Kratzert 的博客“Understanding the backward pass through Batch Normalization Layer”里有详细说明。

1.3.2 Batch Normalization的评估

1.4 正则化

1.5 超参数的验证