关于 CNN 优化的思考

郝杰东

2017 年 7 月 4 日

目录

在正式讨论神经网络的优化之前，先对神经网络给出形式化的表示，并对常用的一些概念进行简单的介绍，方便后续的讨论。

1 神经网络的形式化表示

我们经常使用的 CNN，它的参数优化方法本质上和我们在一些经典教材上看到的加上正则项的曲线拟合很类似。我们可以把 CNN 整体看作一个函数 $f(x;W,b)$，这里 x 就代表经过预处理的图像 (这里预处理通常指的是减去 RGB 3 个通道的均值)。W 代表卷积层 kernel 的权重，b 代表偏置项。函数 $f(\cdot)$ 对输入图像进行了一系列数学运算，其中最常见的有卷积、pooling、非线性变化（(ReLU)。经过这些变换，最后我们得到输出 $\hat{y}$，这个图像还有一个真实的 label，用 y 来表示。我们训练神经网络的目的就是想要使得经过网络的变换，$\hat{y}$ 尽可能接近 y，这样网络才有实际用处。

要想达到上述的目的，必须寻找适当的参数，使得对于一个输入图像 $x^{(i)}$，网络的输出 $\hat{y}^{(i)}$ 接近实际的 $y^{(i)}$，对于这种接近必须要找到某种合适的方式来度量，通常在训练网络的时候我们使用“损失函数 (Loss Functions)”来衡量误差，譬如常用的 mean squared loss，Softmax with loss[1]，cross-entropy loss 等。如果损失越大，说明实际输出与我们想要的结果“偏差”越大。对于单个的样本，loss 可以写成

$$l(\hat{y}^{(i)},y)=l(f(x^{(i)};W,b),y^{(i)}) \tag{1}$$

例如，对于 Softmax with loss，假设研究的是 C 类分类问题，神经网络最后有 C 个输出，分别对应每一类出现，用 f_j 表示，然后再接一个 Softmax 层，经过这一层的作用，对应于每一类的输出变成了

$$z_i=\frac{\exp f_j}{\sum_{j=1}^{C}\exp f_j}. \tag{2}$$

所以 Softmax 作用是把输出归一化，每一类的输出就是输入图像对应于该类的概率。该样本对应的损失可以用如下式子来表达:

$$l_i=-\log\frac{\exp(f_{y_i})}{\sum_{j=1}^{C}\exp(f_j)}. \tag{3}$$

如何来衡量网络性能的好坏呢？仅仅看单个样本损失显然是不合适的，我们需要一个指标来表示对于所有样本，网络预测性能的好坏。通常使用平均损失来表示:

$$Loss=\frac{1}{m}\sum_{i=1}^{m}l(f(x^{(i)};W,b),y^{(i)}) \tag{4}$$

Anyway，得到这个评价网络性能的指标以后，我们实际上就可以使用 backpropagation 来更新网络的参数了。但是通常直接优化这个目标函数得到的网络参数，会产生过拟合，这和曲线拟合的道理是一样的：参数学习的时候把数据里面的噪声分布也学习进去了，这并不是我们所希望的。因此我们需要对参数进行一些限制，就是所谓的正则化 (Regularization)，使得参数的值不那么大。一般正则化只对网络的参数 W 进行，不对偏置项 b 进行，正

[1]实际上并不存在 Softmax loss，Softmax 指的是 Softmax function，也就是把一个向量各元素压缩到 [0,1]，并且各元素之和为 1，变成了一个概率分布，损失函数使用的还是 cross-entropy loss，之所以叫做“Softmax loss”是由于习惯问题，更详细解释参见 [1]。

则项用 $R(W)$ 表示。最终要优化的目标函数 (Objective function) 其实是如下的形式:

$$obj = Loss + \lambda R(w) \tag{5}$$

上式中的 λ 用来调节正则项在总的优化目标中的权重，通常被称为 weight decay。

2 模型的优化

2.1 属于哪类优化问题?

要优化这个问题，首先要确定这个问题的类型，然后才能从已经非常完备的最优化理论里找出合适的工具。前面一直有点不明白这个优化问题的形式到底属于什么呢？在最优化的课上，我们碰到过诸如以下形式的优化问题:

$$\min f(x_1, x_2) = 2{x_1}^2 + {x_2}^2 + (x_1 + x_2)^2 - 20x_1 - 16x_2 \tag{6}$$

神经网络的优化跟上述问题的优化本质一样吗？乍一看上去好像不一样，神经网络里面有样本，这个优化问题无所谓样本。我一直是这么想的，然后就搞不清楚了，那神经网络的优化到底属于什么问题呢?

学习了最优化以后，现在再看神经网络的优化，发现这就是无约束最优化问题 (unconstrained optimization)。这些样本其实是遮掩本质的“幕布”而已。我们把这些样本的具体数值代入网络，最后其实这些样本都变成了具体的数字，剩下的未知变量就是权重 W 和偏置 b。这样，神经网路的优化和上面的优化问题是一样的，神经网络中的权重以及偏置相当于上面优化问题中的变量 x_1 和 x_2。所以神经网络的优化也是无约束最优化问题，只不过不像上面的问题那样简单。

紧接着又有一个需要考虑的问题，深度神经网络对应的函数并不是一个凸函数，这个函数有很多 local minimum，我们经常使用的优化方法都是针对凸函数的，对于非凸也非凹的函数，通常的方法在理论上是并不合适的。例如经常用到的梯度下降只有在函数是 convex 的时候才能保证一定能够找到函数的最优值，对于非凸函数，梯度下降找到的可能只是函数的 local minimum，但是研究者发现，即使梯度下降找到的不是最优值，对于所要解决的问题来说，结果已经足够好了：梯度下降找到的最优值已经接近全局的最优值。

2.2 几种方法

优化的目的就是找到合适的参数，使得目标函数能够取得最小值，对于无约束问题的优化，有很多方法，到底该使用什么方法呢？总结一下可能的方法，主要有 3 种，但是它们不一定都适用于 CNN 的优化，这是由 CNN 的特点决定的：CNN 的参数量实在是太大了。

2.2.1 启发式搜索算法

第一种类型，不使用关于梯度的信息，采用一些启发式的算法，譬如粒子群算法，基因算法，模拟退火算法等，这些算法都是采用启发式的搜索策略来寻找最优参数，但是实际中很少有人用这种方法，因为这些方法并不能保证找到最优解，还有一点就是，卷积神经网络的参数数量太大，使用这些方法效率低，时间成本更大 [2]，引用一下 Hinton [3]，他曾写到:

> Evolution is much too slow to discover the millions of features we need. In very high-dimensional spaces, searches that have efficient access to gradient information are millions of times faster than searches that do not. Evolution can optimize hundreds or even thousands of parameters, but it is hopelessly inefficient for optimizing millions of parameters because it cannot compute the gradient of the fitness of the phenotype with respect to heritable parameters. What evolution can do is explore the space of biological devices that can make effective use of gradient information. It can also explore the space of objective functions that these devices should optimize and the space of architectures in which this optimization works well.

上面的话说明了这些 metaheuristic 方法的对于大规模优化的低效，这大概也是主流深度学习的研究者不使用这些方法优化神经网络的原因（在网上也看到了一些使用 meta-heuristric 方法优化 CNN 的讨论 [4] 和文章 [5]，但是这篇文章也只是在 MNIST 和 Cifar-10 这种小数据库上做实验）。

2.2.2 使用一阶梯度信息

第二种类型就是使用一阶梯度信息的方法，譬如研究者经常使用的随机梯度下降法（SGD），梯度下降法的思想就是先给权重和偏置一个初始值，然后逐步迭代优化这个初始值，使得优化目标不断减小。迭代优化的方法就是求取目标对各个权重以及偏置的偏导数，也就是梯度，根据最优化理论，梯度的负方向是目标函数值下降的方向，知道了下降的方向，我们还需要知道沿着这个方向走多远，也就是所谓的步长（在神经网络的训练中，经常被称为学习率，learning rate），步长过小，目标值下降太少，步长过大，目标值反而可能上升，因此步长的选取是极端重要。选择步长有两种策略，一种是 exact search 方法，另外一类就是 inexact search。那为什么这两种方法都没有被大规模使用呢?

为什么不使用 line search?
首先，SGD 得到的梯度只是对真正梯度的近似，可能存在偏差（noisy gradient），因此采用精确或者非精确搜索，效果可能不会很好；其次，line search 增加了计算量，对于神经网络来说，可能需要好几个 forward pass 来更新一次参数，而 SGD 只需要一次 forward 和一次 backward pass 来更新参数，更节省时间，参数更新频率更快一些。更多的内容，可以参考 [6] 和 [7]，一篇在 SGD 中使用 line search 的论文参见 [8]。

理论上来说，使用梯度下降法的时候，需要把全部样本输入网络，计算一次目标函数，再计算目标函数对参数的偏导数，然后更新这些参数，对于大规模问题（通常训练样本数目是几十万或者上百万），使用梯度下降法的代价太大了（时间成本，存储空间考虑），因此在实践中，研究者通常使用随机梯度下降 [9] 来代替梯度下降。随机梯度下降中计算的梯度是对真正梯度的近似，因此随机梯度下降每一步的方向可能并没有梯度下降准确，但是在梯度下降更新一次参数的时间里，随机梯度下降可以进行多次参数更新，这也弥补了随机梯度下降的不足。在实践中，研究者发现随机梯度下降已经能够工作的很好了。严

格来说，随机梯度下降只是使用所有样本中的一个样本，但是这样计算的梯度通常太粗糙，不能对真正梯度有很好的近似，因此通常使用的是 mini-batch stochastic gradient descent，即从所有样本中随机选出一小部分样本（称为一个 mini-batch），计算在这个 mini batch 上的目标函数，然后计算梯度，来拟合真实的梯度。我们在各种深度学习的库里，经常看到的 *batch size* 就是这个含义。

几种 gradient descent 有什么区别?
在网络上或者文献里，经常看到不同的称呼，batch gradient descent, stochastic gradient descent, mini-batch gradient descent, 这些名词有什么区别呢？Batch gradient descent 其实就是使用全部的训练数据来更新参数的方式，这种方式非常慢（因为需要使用全部训练样本来进行一次参数更新），并且不适合用 online 的方式来更新权重；stochastic gradient descent 上面已经说过，只使用一个样本，对真实梯度的拟合不好；mini-batch gradient descent，介于两者之间，每次从训练样本中随机抽出一部分，虽然得到的梯度可能是 noisy 的，但是更新频率高，最终也能得到一个较好的局部最优解。很多时候 SGD 和 mini-batch 两个是混用的，譬如某个文献说使用的方法是 SGD，其实使用的是 mini-batch stochastic gradient descent。

使用上面 naive 版本的梯度下降方法的时候，在遇到梯度处于“狭长山谷”的时候，寻优的路径会出现震荡，因而迭代需要很长时间才能达到最优值，因此实际中更常用的方法是 SGD with momentum [10]，momentum 方法的思想，是对于权重的更新，不能仅仅依赖当前的梯度，还需要考虑前面时刻的梯度，这使得当寻优方法进入参数空间的“狭长山谷”时，能够尽早脱身，而不是在山谷来回震荡，浪费时间。网上有两种关于这种方法的公式，第一种 [11, 12, 13]：

$$v^{t+1} = \mu v^t + \alpha \nabla_\theta obj \tag{7}$$

$$\theta^{t+1} = \theta^t - v^{t+1} \tag{8}$$

第二种 [14, 15, 16][17, Chapter 8]：

$$v^{t+1} = \mu v^t - \alpha \nabla_\theta obj \tag{9}$$

$$\theta^{t+1} = \theta^t + v^{t+1} \tag{10}$$

上述两组公式中，μ 是 momentum 项的系数，α 是权重的学习率，θ 是某个要优化的权重，上标中的 $t+1$，t 代表第几次迭代，初始的时候 v 等于 0。初看上去，这两种公式不一样，其实化简一下，这两中表达方式得到的结果是一模一样的。使用含有 momentum 的 SGD 训练网络，通常收敛速度更快，也更稳定，这也是该方法被大量使用的原因。需要指出，含有 momentum 项的 SGD 只是 vanilla SGD 的一种简单改进，还有很多对 SGD 的改进，譬如 Nesterov’s Accelerated Gradient，RMSprop。更多关于 SGD 的内容，可以参考 [15, 13, 11, 18, 19]。

除了梯度下降，共轭梯度法也是最优化里面常用的解决无约束问题的优化方法，论文 [20] 提到了使用共轭梯度法来训练神经网络。

2.2.3 使用二阶梯度信息

第三种类型的方法是使用二阶梯度信息，使用二阶梯度的方法最原始的是牛顿法，但是由于牛顿法需要求 Hessian 矩阵的逆，计算量巨大，该逆也不

一定存在。对牛顿法的改进有 BFGS，以及 L-BFGS 等，更多信息可以参考 [21, 22]。二阶方法在 CNN 中用的也不多，一般都用来优化一些小规模的问题 [20]，大规模的问题，还是 SGD 更有效一些 [23]。

为什么二阶方法在 CNN 优化中用的不多?
最主要的原因还是二阶方法需要计算 Hessian 矩阵或者它的近似矩阵，CNN 参数量巨大，二阶方法需要的计算量太大，not feasible，所以实际中研究者更倾向于使用 SGD 来进行优化，更多讨论，参见 [23, 24, 25]。

更多关于优化应用与机器学习的讨论，可以参考 [26]。

参考资料

[1] karpathy. http://cs231n.github.io/linear-classify/#softmax.

[2] Quora. https://www.quora.com/Why-are-neural-networks-trained-with-steepest-descent-using-a-fixed-step-size-instead-of-more-sophisticated-optimization-methods.

[3] G. Hinton, "Where do features come from?," *Cognitive science*, vol. 38, no. 6, pp. 1078–1101, 2014.

[4] Quora. https://www.quora.com/In-training-deep-neural-networks-why-are-gradient-methods-commonly-used-as-opposed-to-other-metaheuristics.

[5] V. Ayumi, L. Rere, M. I. Fanany, and A. M. Arymurthy, "Optimization of convolutional neural network using microcanonical annealing algorithm," *arXiv preprint arXiv:1610.02306*, 2016.

[6] Reddit. https://www.reddit.com/r/MachineLearning/comments/4trmky/why_arent_line_search_algorithms_used_in/.

[7] Reddit. https://www.quora.com/Is-line-search-used-commonly-with-SGD-while-learning-the-parameters-for-a-deep-neural-networks.

[8] M. Mahsereci and P. Hennig, "Probabilistic line searches for stochastic optimization," in *Advances in Neural Information Processing Systems 28*, pp. 181–189, Curran Associates, Inc., 2015.

[9] H. Robbins and S. Monro, "A stochastic approximation method," *The annals of mathematical statistics*, pp. 400–407, 1951.

[10] I. Sutskever, J. Martens, G. E. Dahl, and G. E. Hinton, "On the importance of initialization and momentum in deep learning.," *ICML (3)*, vol. 28, pp. 1139–1147, 2013.

[11] S. Ruder. http://sebastianruder.com/optimizing-gradient-descent/index.html.

[12] UFLDL. http://ufldl.stanford.edu/tutorial/supervised/OptimizationStochasticGradientDescent/.

[13] WIKIPEDIA. https://en.wikipedia.org/wiki/Stochastic_gradient_descent.

[14] karpathy. http://cs231n.github.io/neural-networks-3/#sgd.

[15] BVLC. http://caffe.berkeleyvision.org/tutorial/solver.html.

[16] G. Hinton. http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

[17] R. Rojas, *Neural networks: a systematic introduction*. Springer Science & Business Media, 2013.

[18] Quora. https://www.quora.com/Whats-the-difference-between-gradient-descent-and-stochastic-gradient-descent.

[19] L. Bottou, "Stochastic gradient descent tricks," in *Neural Networks: Tricks of the Trade*, pp. 421–436, Springer, 2012.

[20] J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, Q. V. Le, and A. Y. Ng, "On optimization methods for deep learning," in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pp. 265–272, 2011.

[21] L. Vandenberghe. http://www.seas.ucla.edu/~vandenbe/236C/lectures/cg.pdf.

[22] MathWorks. https://www.mathworks.com/help/optim/ug/unconstrained-nonlinear-optimization-algorithms.html#bqa_jtu.

[23] Quora. https://www.quora.com/Why-are-second-order-optimization-techniques-for-neural-networks-better-than-first-order-ones.

[24] Quora. https://www.quora.com/Why-are-optimization-techniques-like-natural-gradient-and-second-order-methods-L-BFGS-for-eg-not-much-used-in-deep-learning.

[25] H. News. https://news.ycombinator.com/item?id=11943685.

[26] L. Bottou, F. E. Curtis, and J. Nocedal, "Optimization methods for large-scale machine learning," *arXiv preprint arXiv:1606.04838*, 2016.

3 版本更新历史

版本	日期	作者	更新说明
1.0	2016-11-25	郝杰东	初次创建
1.05	2016-11-28	-	基本完成
1.1	2016-11-29	-	增加更多的引用，修改细节
1.12	2017-01-05	-	修改一些拼写错误，增加对 softmax loss 的解释
1.2	2017-07-04	-	修改措辞，改进内容