

# 当前训练神经网络最快的方式：AdamW优化算法+超级收敛

机器之心

已认证的官方帐号

选自fast.ai

作者：Sylvain Gugger、Jeremy Howard

机器之心编译

参与：思源、王淑婷、张倩

最优化方法一直是机器学习非常重要的部分，也是学习过程的核心算法。而 Adam 自 14 年提出以来就受到广泛关注，目前该论文的引用量已经达到了 10047。不过自去年以来，很多研究者发现 Adam 优化算法的收敛性得不到保证，ICLR 2017 的最佳论文也重点关注它的收敛性。在本文中，作者发现大多数深度学习库的 Adam 实现都有一些问题，并在 fastai 库中实现了一种新型 AdamW 算法。根据一些实验，作者表示该算法是目前训练神经网络最快的方式。

## Adam 过山车

Adam 优化器之旅可以说是过山车（roller-coaster）式的。该优化器于 2014 年推出，本质上是一个出于直觉的简单想法：既然我们明确地知道某些参数需要移动得更快、更远，那么为什么每个参数还要遵循相同的学习率？因为最近梯度的平方告诉我们每一个权重可以得到多少信号，所以我们可以除以这个，以确保即使是最迟钝的权重也有机会发光。Adam 接受了这个想法，在过程中加入了标准方法，就这样产生了 Adam 优化器（稍加调整以避免早期批次出现偏差）！

首次发表之时，深度学习社区都为来自原文论文的一些图表（如下图）兴奋不已：

The chart shows training cost on a logarithmic scale (10^-1 to 10^-2) versus iterations over the entire dataset (0 to 200). Five optimizers are compared: AdaGrad (blue), RMSProp (green), SGNesterov (red), AdaDelta (cyan), and Adam (magenta). Adam shows the fastest decrease in training cost, reaching the lowest value by iteration 200.

## Adam 和其他优化器的对比

训练速度提高 200%！「总体来看，我们发现 Adam 非常鲁棒，而且广泛适用于机器学习领域的各种非凸优化问题」论文结尾这样写道。那是二年前，深度学习的黄金时期。然而，事情并没有按照我们期望的方向发展。使用 Adam 训练模型的研究文章少之又少，新的研究开始明显地抑制了它的应用，并在几个实验中表明，SGD+momentum 可能比复杂的 Adam 表现更好。2018 fast.ai 课程开课之际，可怜的 Adam 被从早期课程中删除。

但是到了 2017 年末，Adam 似乎又重获新生。Ilya Loshchilov 和 Frank Hutter 在他们的论文《Fixing Weight Decay Regularization in Adam》中指出，每个库在 Adam 上实施的权重衰减似乎都是错误的，并提出了一种简单的方法（他们称之为 AdamW）来修复它。尽管结果略有不同，但他们确实给出了一些类似下图的令人鼓舞的图表：

The chart shows training loss (cross-entropy) on a logarithmic scale (10^-4 to 10^-1) versus epochs (0 to 1800). Two optimizers are compared: Adam (blue) and AdamW (magenta). AdamW shows a faster decrease in training loss, reaching the lowest value by epoch 1800.

## Adam 和 AdamW 对比

我们希望人们恢复对 Adam 的热情，因为该优化器的一些早期结果似乎可以复现。但事与愿违。实际上，应用它的唯一框架就是使用 Sylvain 编码的 fastai。由于缺乏可用的广泛框架，日常实践者就只能固守又旧又不好用的 Adam。

但这不是唯一的问题。前面还有很多阻碍。两篇论文指出了 Adam 在收敛性证明方面的明显问题，尽管其中一篇提出了名为 AMSGrad 的修正（并在享有盛誉的 ICLR 大会上赢得了「最佳论文」奖）。但是，如果说我们从这种戏剧化的生活（至少按照优化器的标准来说是戏剧化的）简史中学到了什么，那就是，没有什么它是表面看起来的样子。的确，博士生 Jeremy Bernstein 指出，所谓的收敛问题其实只是选择不当的超参数的迹象，也许 AMSGrad 也解决不了问题。另一名博士生 Filip Korzeniewski 展示了一些早期成果，似乎支持了 AMSGrad 这种令人沮丧的观点。

## 启动过山车

那么我们这些只希望快速训练精确模型的人该做些什么呢？我们选择用数百年来解决科学辩论的方式——科学实验——来解决这一争议！稍后将呈现所有细节，但首先让我们来看一下大致结果：

适当调参之后，Adam 真的可以用！我们在以下几个任务中得到了训练时间方面的最新结果：

在含有测试时间增加的仅仅 18 个 epoch 或 30 个 epoch 上训练 CIFAR10，直到其准确率超过 94%，如 DAWN-Bench 竞赛；

对 ResNet50 进行调参，直至其在斯坦福汽车数据集上的准确率达到 90%，只需训练 60 个 epoch（之前达到相同的准确率需要 600 个 epoch）；

从零开始训练一个 AWD LSTM or QRNN，历经 90 个 epoch（或在一个 GPU 上训练 1 个半小时），其困惑度在 Wikitext-2 上达到当前最优水平（之前的 LSTM 需要 750 个 epoch，QRNN 需要 500 个 epoch）。

这意味着我们已经看到使用 Adam 的超收敛！超收敛是训练学习率高的神经网络时出现的一种现象，它表示选择的超参数省了一半训练过程。在 AdamW 之前，训练 CIFAR10 至 94 % 的准确率需要大约 100 个 epoch。

与之前的工作相比，我们发现只要调整得当，Adam 在我们尝试过的每一个 CNN 图像问题上都可以获得与 SGD+Momentum 一样好的准确率，而且几乎总是快一点。

关于 AMSGrad 是一个糟糕的「解决方案」的建议是正确的。我们一直发现，AMSGrad 在准确率（或其他相关指标）上没有获得比普通 Adam / AdamW 更高的增益。

当你听到人们说 Adam 的泛化性能不如 SGD+Momentum 时，你基本上总会发现他们为自己的模型选择的超参数不咋地。通常 Adam 需要的正则化比 SGD 多，因此在从 SGD 转向 Adam 时，确保调整正则化超参数。

## 文章结构：

### 1. AdamW

1. 理解 AdamW
2. 实现 AdamW
3. AdamW 实验和 AdamW-ish

### 2. AMSGrad

1. 理解 AMSGrad
2. 实现 AMSGrad
3. AMSGrad 实验的结果

### 3. 完整结果图表

## AdamW

### 理解 AdamW：权重衰减与 L2 正则化

L2 正则化是减少过拟合的经典方法，它会向损失函数添加由模型所有权重的平方和组成的惩罚项，并乘上特定的超参数以控制惩罚力度。以下本文所有的方程式都是用 Python、NumPy 和 PyTorch 风格的表达方式：

```
final_loss = loss + wd * all_weights.pow(2).sum() / 2
```

其中 wd 为我们设置的超参数，用以控制惩罚力度。这也可以称为权重衰减，因为每一次运用原版 SGD 时，它都等价于使用如下方程式更新权重：

```
w = w - lr * w.grad - lr * wd * w
```

其中 lr 表示学习率、w.grad 表示损失函数对 w 的导数，而后面的 wd \* w 则表示惩罚项对 w 的求导结果。在这个等式中，我们会看到每一次更新都会减去一小部分权重，这也就是「衰减」的来源。

fast.ai 查看过的所有库都使用第一种形式。在实践中，几乎都是通过向梯度 wd\*w 而实现算法，而不是真正地改变损失函数。因为我们并不希望增加额外的计算量来修正损失，尤其是还有其它简单方法的时候。

既然它们是同一种表达，那么我们为什么需要区分这两种概念呢？原因在于它们只对于原版 SGD 是等价的，而当我们添加动量或使用如 Adam 那样复杂的优化方法，L2 正则化（第一个方程）和权重衰减（第二个方程）就会存在很大的不同。在本文其余的部分中，我们讨论权重衰减指的都是第二个方程，而讨论 L2 正则化都是讨论第一个经典方式。

如下在带动量的 SGD 中，L2 正则化与权重衰减是不等价的。L2 正则化会将 wd\*w 添加到梯度中，但现在权重并不是直接减去梯度。首先我们需要计算移动均值：

```
moving_avg = alpha * moving_avg + (1-alpha) * (w.grad + wd*w)
```

然后权重才能通过减去乘上了学习率的移动均值而得到更新。所以 w 更新中涉及到的正则化是 lr\*(1-alpha)\*wd \* w 加上已经在 moving\_avg 中前面权重的组合。

因此，权重衰减的更新方式可以表示为：

```
moving_avg = alpha * moving_avg + (1-alpha) * w.grad  
w = w - lr * moving_avg - lr * wd * w
```

我们可以观察到，从 w 中减去有关正则化的部分在两种方法中是不同的。当我们使用 Adam 优化器时，权重衰减的部分可能相差更大。因为 Adam 中的 L2 正则化需要添加 wd\*w 到梯度中，并分别计算梯度及其平方的移动均值，然后再能更新权重。然而权重衰减方法只是简单地更新权重，并每次从权重中减去一点。

显然这是两种不同的方法，在进行了实验后，Ilya Loshchilov 和 Frank Hutter 建议我们应该在 Adam 算法中使用权重衰减方法，而不是像经典深度学习库中实现的 L2 正则化。

## 实现 AdamW

那么我们更如何才能实现 AdamW 算法呢？如果你们在使用 fastai 的库，那么在使用 fit 函数时添加参数 use\_wd\_sched=True 就能简单地实现：

```
learn.fit(lr, 1, wds=1e-4, use_wd_sched=True)
```

如果你更喜欢新的训练 API，你就能在每一个训练阶段中使用参数 wd\_loss=False：

```
phases = [TrainingPhase(1, optim.Adam, lr, wds=1-e4, wd_loss=False)]  
learn.fit_opt_sched(phases)
```

以下简要地概述了 fastai 是如何实现 AdamW 的。在优化器中的阶梯函数，我们只需要使用梯度修正参数，根本不使用参数本身的值（除了权重衰减，我们将在外部处理它）。然后我们可以在最优优化器上通简单的实现权重衰减，但这仍需要在计算梯度后才能完成，否则它就会影响梯度的值。所以在训练循环中，我们必须确定计算权重衰减的位置。

```
loss.backward()  
#Do the weight decay here!  
optimizer.step()
```

当然，最优优化器应该设定 wd=0，否则它还会做一些 L2 正则化，这也是我们不希望看到的。现在在权重衰减的位置中，我们可以在所有参数上写一个循环语句，并依次采用权重衰减的更新。而我们的参数应该存储在优化器的字典 param\_groups 中，所以这个循环应该表示为如下语句：

```
loss.backward()  
for group in optimizer.param_groups():  
    for param in group['params']:  
        param.data = param.data.add(-wd * group['lr'], param.data)  
optimizer.step()
```

## AdamW 实验的结果：它真的能行吗？

我们首先在计算机视觉问题上进行测试，效果非常好。具体来说，Adam 和 L2 正则化在 30 个 epoch 中获得的平均准确率为 93.96%，在两次中有一次超过 94%。我们选择 30 个 epoch 是因为通过 1cycle 策略和 SGD 可以获得 94% 准确率。当我们使用 Adam 与权重衰减方法，我们持续获得 94% 到 94.25% 的准确率。为此，我们发现使用 1cycle 策略时的最优 beta2 值为 0.99。我们将 beta1 参数视为 SGD 中的动量，这也就意味着它学习率的增长由 0.95 降低到 0.85，然后随学习率的降低而又增加到 0.95。

L2 正则化或权重衰减准确率

更令人印象深刻的是，使用训练时间增加（即在测试集的一个图像和它四个增加数据的版本上取预测的平均值），我们可以在仅仅 18 个 epoch 内达到 94 % 的准确率（平均 93.98 %）！通过简单的 Adam 和 L2 正则化，每尝试 20 次就会出现一次超过 94 % 的情况。

在这些比较中需要考虑的一点是，改变正则化方式会改变权重衰减或学习率的最佳值。在我们进行的测试中，L2 正则化的最佳学习率为 1e-6（最大学习率为 1e-3），而权重衰减的最佳值为 0.3（学习率为 3e-3）。在我们的所有测试中，数量级的差异都是非常一致的，主要是因为 L2 正则化被梯度的平均范数（相当低）有效地划分，并且 Adam 的学习率相当小（所以权重衰减的更新需要更强的系数）。

那么，权重衰减总是比 Adam 的 L2 正则化更好？我们还没有发现明显更糟的情况，但无论是迁移学习问题（例如斯坦福汽车数据集上 Resnet50 的微调）还是 RNNs，它都没有给出更好的结果。

## AMSGrad

### 理解 AMSGrad

AMSGrad 是由 Sashank J. Reddi、Satyen Kale 和 Sanjiv Kumar 在近期的一篇文章中介绍的。通过分析 Adam 优化器收敛的证明，他们在更新规则中发现了一个错误。该错误可能导致算法收敛到次优点。他们设计了理论实验，展示 Adam 失败的情形，并提出了一个简单的解决方案。机器之心也曾从适应性学习率算法出发分析过这一篇最佳论文：[Beyond Adam](#)。

为了更好地理解错误和解决方案，我们来看一下 Adam 的更新规则：

```
avg_grads = beta1 * avg_grads + (1-beta1) * w.grad  
avg_squared = beta2 * (avg_squared) + (1-beta2) * (w.grad ** 2)  
w = w - lr * avg_grads / sqrt(avg_squared)
```

我们刚刚跳过了偏差校正（对训练的开头很有用），把重心放在了主要点上。作者发现 Adam 收敛证明中的错误之处在于：

```
lr / sqrt(avg_squared)
```

这是我们朝着平均梯度方向迈出的一步，在训练中逐渐减少。由于学习率常常是恒定或递减的，作者提出的解决方案是通过添加另一个变量来跟踪它们的最大值，从而迫使 avg\_square 量增加。

## 实现 AMSGrad

相关文章在 ICLR 2018 中获得一项大奖并广受欢迎，而且它已经在两个主要的深度学习库——PyTorch 和 Keras 中实现。所以，我们只需传入参数 amsgrad = True 即可。

```
avg_grads = beta1 * avg_grads + (1-beta1) * w.grad  
avg_squared = beta2 * (avg_squared) + (1-beta2) * (w.grad ** 2)  
max_squared = max(avg_squared, max_squared)  
w = w - lr * avg_grads / sqrt(max_squared)
```

## AMSGrad 实验结果：大量噪音都是没用的

AMSGrad 的结果令人非常失望。在所有实验中，我们都发现它没有丝毫帮助。即使 AMSGrad 发现的最小值有时比 Adam 达到的最小值稍低（在损失方面），其度量（准确率、f1 分数...）最终总是更糟（详见引言中的表格）。

Adam 优化器在深度学习中的收敛证明（因为它针对凸问题）和他们在其中发现的错误对于与现实问题无关的合成实验很重要。实际测试表明，当这些 avg\_square 梯度想要减小时，这么做能得到最好的结果。

这表明，即使把重点放在理论上有助于获得一些新想法，也没有什么可以取代实验（而且很多实验！）以确保这些想法实际上有助于从业人员训练更好的模型。

## 附录：所有结果

从零开始训练 CIFAR10（模型是 Wide-ResNet-22，以下为五个模型的平均结果）：

Method	Without amsgrad	With amsgrad
AdamW	94.34%	93.69%
Adam	93.94%	93.36%

Method	Without amsgrad	With amsgrad
AdamW	89.2%/90.5%	89.9%/90.5%
Adam	89.6%/91%	89.9%/91%

Method	Raw model	With cache pointer
Adam	68.7/65.5	52.9/50.9
Adam + amsgrad	69.4/66.5	53.1/51.3
AdamW	69.3/66	54.1/51.9
AdamW + amsgrad	72.7/69	57/54.7

Method	Raw model	With cache pointer
Adam	69.6/66.7	53.6/51.7
Adam + amsgrad	71.5/68.4	54.2/52.2
AdamW	70.5/67.3	55.5/53.3
AdamW + amsgrad	74.3/70.9	57.8/55.6

针对这一具体任务，我们采用了 1cycle 策略的修改版本，加快了学习速度，之后长时间保持较高的恒定学习速度，然后再往下降。

The figure contains two line plots. The left plot shows the learning rate over 800 iterations. It starts at 0.001, increases to 0.01 at iteration 100, stays constant until iteration 400, and then decreases back to 0.001 by iteration 800. The right plot shows the momentum over 800 iterations. It starts at 0.95, decreases to 0.85 at iteration 100, stays constant until iteration 400, and then increases back to 0.95 by iteration 800.

## Adam 和其它优化器之间的对比

所有相关超参数的值以及用于产生这些结果的代码地址如下：[github.com/sgugger/Adam...](#)

原文链接：[fast.ai/2018/07/02/adam...](#)

发布于 2018-07-03

机器学习

神经网络

人工智能