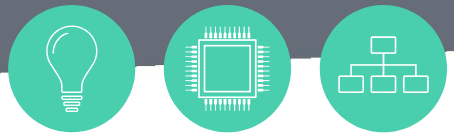
深度学习之 PyTorch 实战

优化算法 part 2

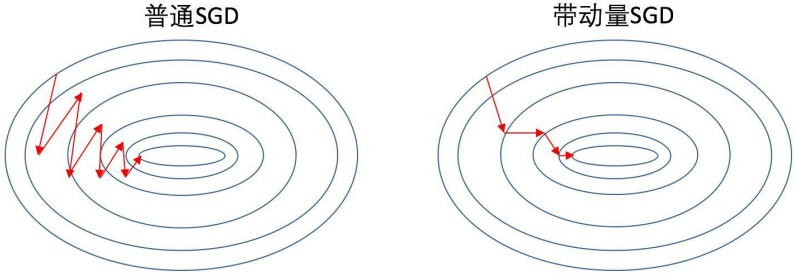


主讲老师: 土豆老师

目录

动量法

- 在「梯度下降和随机梯度下降」节中我们提到，目标函数有关自变量的梯度代表了目标函数在自变量当前位置下降最快的方向。因此，梯度下降也叫作最陡下降（steepest descent）。
- 在每次迭代中，梯度下降根据自变量当前位置，沿着当前位置的梯度更新自变量。然而，如果自变量的迭代方向仅仅取决于自变量当前位置，这可能会带来一些问题。



“Talk is cheap. Show me the code.”

- 动量法的提出是为了解决梯度下降的上述问题。

AdaGrad 算法

- 在之前介绍过的优化算法中，目标函数自变量的每一个元素在相同时间步都使用同一个学习率来自我迭代。举个例子，假设目标函数为 f，自变量为一个二维向量 $[x_1, x_2]^\top$ 该向量中每一个元素在迭代时都使用相同的学习率。例如，在学习率为 η 的梯度下降中，元素 x_1 和 x_2 都使用相同的学习率 η 来自我迭代：

$$x_1 \leftarrow x_1 - \eta \frac{\partial f}{\partial x_1}, \quad x_2 \leftarrow x_2 - \eta \frac{\partial f}{\partial x_2}$$

- 在「动量法」里我们看到当 x_1 和 x_2 的梯度值有较大差别时，需要选择足够小的学习率使得自变量在梯度值较大的维度上不发散。但这样会导致自变量在梯度值较小的维度上迭代过慢。动量法依赖指数加权移动平均使得自变量的更新方向更加一致，从而降低发散的可能。

- 本节我们介绍 AdaGrad 算法，它根据自变量在每个维度的梯度值的大小来调整各个维度上的学习率，从而避免统一的学习率难以适应所有维度的问题。

AdaGrad 算法

- AdaGrad 算法会使用一个小批量随机梯度 $\boldsymbol{g}_t$ 按元素平方的累加变量 $\boldsymbol{s}_t$ 。在时间步 0，AdaGrad 将 $\boldsymbol{s}_0$ 中每个元素初始化为 0。在时间步 t，首先将小批量随机梯度 $\boldsymbol{g}_t$ 按元素平方后累加到变量 $\boldsymbol{s}_t$：

$$\boldsymbol{s}_t \leftarrow \boldsymbol{s}_{t-1} + \boldsymbol{g}_t \odot \boldsymbol{g}_t$$

- 其中 $\odot$ 是按元素相乘。接着，我们将目标函数自变量中每个元素的学习率通过按元素运算重新调整一下：

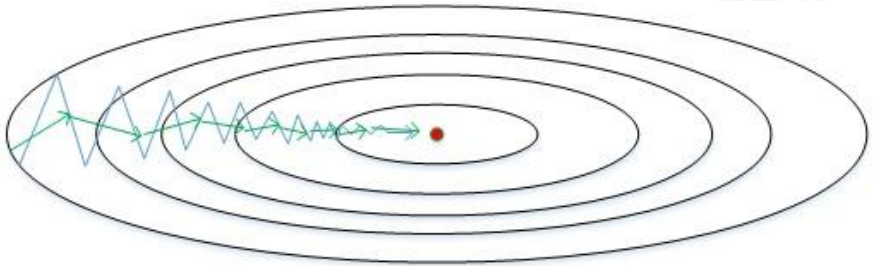
$$\boldsymbol{x}_t \leftarrow \boldsymbol{x}_{t-1} - \frac{\eta}{\sqrt{\boldsymbol{s}_t + \epsilon}} \odot \boldsymbol{g}_t$$

其中 η 是学习率，ϵ 是为了维持数值稳定性而添加的常数，如 10^{-6} 。这里开方、除法和乘法的运算都是按元素运算的。这些按元素运算使得目标函数自变量中每个元素都分别拥有自己的学习率。

“Talk is cheap. Show me the code.”

RMSProp 算法

- 我们在上节「AdaGrad 算法」中提到，因为调整学习率时分母上的变量 $\boldsymbol{s}_t$ 一直在累加按元素平方的小批量随机梯度，所以目标函数自变量每个元素的学习率在迭代过程中一直在降低（或不变）。因此，当学习率在迭代早期降得较快且当前解依然不佳时，AdaGrad 算法在迭代后期由于学习率过小，可能较难找到一个有用的解。
- 为了解决这一问题，RMSProp 算法对 AdaGrad 算法做了一点小小的修改。不同于 AdaGrad 算法里状态变量 $\boldsymbol{s}_t$ 是截至时间步 t 所有小批量随机梯度 $\boldsymbol{g}_t$ 按元素平方和，RMSProp 算法将这些梯度按元素平方做指数加权移动平均。



RMSProp 算法

- 具体来说，给定超参数 $0 \leq \gamma < 1$，RMSProp 算法在时间步 $t > 0$ 计算

$$\boldsymbol{s}_t \leftarrow \gamma \boldsymbol{s}_{t-1} + (1-\gamma)\boldsymbol{g}_t \odot \boldsymbol{g}_t.$$

和 AdaGrad 算法一样，RMSProp 算法将目标函数自变量中每个元素的学习率通过按元素运算重新调整，然后更新自变量

$$\boldsymbol{x}_t \leftarrow \boldsymbol{x}_{t-1} - \frac{\eta}{\sqrt{\boldsymbol{s}_t + \epsilon}} \odot \boldsymbol{g}_t$$

其中 η 是学习率，ϵ 是为了维持数值稳定性而添加的常数，如 10^{-6}。因为 RMSProp 算法的状态变量 $\boldsymbol{s}_t$ 是对平方项 $\boldsymbol{g}_t \odot \boldsymbol{g}_t$ 的指数加权移动平均，所以可以看作是最近 $1/(1-\gamma)$ 个时间步的小批量随机梯度平方项的加权平均。如此一来，自变量每个元素的学习率在迭代过程中就不再一直降低（或不变）。

“ Talk is cheap. Show me the code. ”

AdaDelta 算法

- 除了 RMSProp 算法以外，另一个常用优化算法 AdaDelta算法 也针对 AdaGrad 算法在迭代后期可能较难找到有用解的问题做了改进。有意思的是，AdaDelta 算法没有学习率这一超参数。

- AdaDelta 算法也像 RMSProp 算法一样，使用了小批量随机梯度 $\boldsymbol{g}_t$ 按元素平方的指数加权移动平均变量 $\boldsymbol{s}_t$ 。在时间步 0，它的所有元素被初始化为 0。给定超参数 $0 \leq \rho < 1$ (对应 RMSProp 算法中的 γ) , 在时间步 $t > 0$, 同 RMSProp 算法一样计算

$$\boldsymbol{s}_t \leftarrow \rho \boldsymbol{s}_{t-1} + (1 - \rho) \boldsymbol{g}_t \odot \boldsymbol{g}_t$$

- 与 RMSProp 算法不同的是，AdaDelta 算法还维护一个额外的状态变量 $\Delta \boldsymbol{x}_t$ ，其元素同样在时间步 0 时被初始化为 0。我们使用 $\Delta \boldsymbol{x}_{t-1}$ 来计算自变量的变化量:

$$\boldsymbol{g}_t' \leftarrow \sqrt{\frac{\Delta x_{t-1} + \epsilon}{st + \epsilon}} \odot \boldsymbol{g}_t$$

其中 ϵ 是为了维持数值稳定性而添加的常数，如 10^{-5} 。

AdaDelta 算法

- 接着更新自变量：

$$\boldsymbol{x}_t \leftarrow \boldsymbol{x}_{t-1} - \boldsymbol{g}'_t$$

- 最后，我们使用 $\Delta\boldsymbol{x}_t$ 来记录自变量变化量 $\boldsymbol{g}'_t$ 按元素平方的指数加权移动平均：

$$\Delta\boldsymbol{x}_t \leftarrow \rho\Delta\boldsymbol{x}_{t-1} + (1-\rho)\boldsymbol{g}'_t \odot \boldsymbol{g}'_t.$$

- 可以看到，如不考虑 ϵ 的影响，AdaDelta 算法跟 RMSProp 算法的不同之处在于使用 $\sqrt{\Delta\boldsymbol{x}_{t-1}}$ 来替代学习率 η 。

“ Talk is cheap. Show me the code. ”

Adam 算法

- Adam 算法在 RMSProp 算法基础上对小批量随机梯度也做了指数加权移动平均，所以 Adam 算法可以看做是 RMSProp 算法与动量法的结合。下面我们来介绍这个算法。
- Adam 算法使用了动量变量 $\boldsymbol{v}_t$ 和 RMSProp 算法中小批量随机梯度按元素平方的指数加权移动平均变量 $\boldsymbol{s}_t$，并在时间步 0 将它们中每个元素初始化为 0 。给定超参数 $0 \leq \beta_1 < 1$（算法作者建议设为 0.9），时间步 t 的动量变量 $\boldsymbol{v}_t$ 即小批量随机梯度 $\boldsymbol{g}_t$ 的指数加权移动平均：

$$\boldsymbol{v}_t \leftarrow \beta_1 \boldsymbol{v}_{t-1} + (1 - \beta_1)\boldsymbol{g}_t$$

- 和 RMSProp 算法中一样，给定超参数 $0 \leq \beta_2 < 1$（算法作者建议设为 0.999），将小批量随机梯度按元素平方后的项 $\boldsymbol{g}_t \odot \boldsymbol{g}_t$ 做指数加权移动平均得到 $\boldsymbol{s}_t$：

$$\boldsymbol{s}_t \leftarrow \beta_2 \boldsymbol{s}_{t-1} + (1 - \beta_2)\boldsymbol{g}_t \odot \boldsymbol{g}_t$$

Adam 算法

“Talk is cheap. Show me the code.”

- 由于我们将 $\boldsymbol{v}_0$ 和 $\boldsymbol{s}_0$ 中的元素都初始化为 0，在时间步 t 我们得到 $\boldsymbol{v}_t = (1-\beta_1)\sum_{i=1}^{t}\beta_1^{t-i}\boldsymbol{g}_i$
- 将过去各时间步小批量随机梯度的权值相加，得到 $(1-\beta_1)\sum_{i=1}^{t}\beta_1^{t-i} = 1-\beta_1^t$
- 需要注意的是，当 t 较小时，过去各时间步小批量随机梯度权值之和会较小。例如，当 $\beta_1 = 0.9$ 时，$\boldsymbol{v}_1 = 0.1\boldsymbol{g}_1$ 。为了消除这样的影响，对于任意时间步 t，我们可以将 $\boldsymbol{v}_t$ 再除以 $1-\beta_1^t$，从而使过去各时间步小批量随机梯度权值之和为 1。这也叫作偏差修正。
- 在Adam算法中，我们对变量 $\boldsymbol{v}_t$ 和 $\boldsymbol{s}_t$ 均作偏差修正：$\hat{\boldsymbol{v}}_t \leftarrow \dfrac{\boldsymbol{v}t}{1-\beta_1^t}$，$\hat{\boldsymbol{s}}_t \leftarrow \dfrac{\boldsymbol{s}_t}{1-\beta_2^t}$
- 接下来，Adam算法使用以上偏差修正后的变量 $\hat{\boldsymbol{v}}_t$ 和 $\hat{\boldsymbol{s}}_t$，将模型参数中每个元素的学习率通过按元素运算重新调整：

$$\boldsymbol{g}_t' \leftarrow \frac{\eta\hat{\boldsymbol{v}}_t}{\sqrt{\hat{\boldsymbol{s}}\boldsymbol{t}}+\epsilon}$$

其中 η 是学习率，ϵ 是为了维持数值稳定性而添加的常数，如 10^{-8}。和 AdaGrad 算法、RMSProp 算法以及 AdaDelta 算法一样，目标函数自变量中每个元素都分别拥有自己的学习率。最后，使用 $\boldsymbol{g}_t'$ 迭代自变量：$\boldsymbol{x}_t \leftarrow \boldsymbol{x}_{t-1} - \boldsymbol{g}_t'$.

小结

- 动量法使用了指数加权移动平均的思想。它将过去时间步的梯度做了加权平均，且权重按时间步指数衰减。
- 动量法使得相邻时间步的自变量更新在方向上更加一致。
- AdaGrad 算法在迭代过程中不断调整学习率，并让目标函数自变量中每个元素都分别拥有自己的学习率。
- 使用 AdaGrad 算法时，自变量中每个元素的学习率在迭代过程中一直在降低（或不变）。
- RMSProp 算法和 AdaGrad 算法的不同在于，RMSProp 算法使用了小批量随机梯度按元素平方的指数加权移动平均来调整学习率。
- AdaDelta 算法没有学习率超参数，它通过使用有关自变量更新量平方的指数加权移动平均的项来替代 RMSProp 算法中的学习率。
- Adam 算法在 RMSProp 算法的基础上对小批量随机梯度也做了指数加权移动平均。
- Adam 算法使用了偏差修正。

谢谢观看

更多好课，请关注万门好课APP