Optimizers

optimizers 通用参数

- 待优化参数：w, 目标函数：$f(w)$, 初始learning rate：a
- 在每一个epoch t 中：
  1. 计算目标函数关于当前参数的梯度： $g_t = \nabla f(w_t)$
  2. 根据历史梯度计算一阶动量和二阶动量：
     $m_t = \phi(g_1, g_2, \cdots, g_t); V_t = \psi(g_1, g_2, \cdots, g_t)$
     1. 计算当前时刻的下降梯度：$\eta_t = \alpha \cdot m_t / \sqrt{V_t}$
  3. 根据下降梯度进行更新： $w_{t+1} = w_t - \eta_t$

SGD

- 一阶动量：$m_t = g_t$; 二阶动量：$V_t = I^2$
- 下降梯度：$\eta_t = a \cdot g_t$

现在通用的SGD通常指代mini-batch SGD，其迭代次数为每个batch的个数n。对于每次迭代，SGD对每个batch求样本的梯度均值然后进行梯度更新。

$$w_{t+1} = w_t - a \cdot \frac{1}{n} \sum \nabla f(w_t)$$

SGD-Momentum

- 一阶动量：$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$
  - 一阶动量是哥哥时刻梯度方向的指数移动平均值，约等于最近$1/(1-\beta_1)$个时刻的梯度向量和的平均值
- 主要由此前累积的下降方向所决定

AdaGrad

自适应学习率

- 二阶动量：$V_t = \sum g_t^2$：迄今为止所有梯度的平方和

- 梯度下降：$\eta_t = a \cdot \frac{m_t}{\sqrt{V_t}}$

实质上，learning rate 由 a 变成了 $\frac{a}{\sqrt{V_t}}$ 随着迭代时候二阶动量越大，学习率越小

RMSProp

只关注过去一段时间的梯度变化而非全部变化，这里区别于AdaGrad。避免二阶动量持续累积，导致训练提前结束。

- 二阶动量：$V_t = \beta_2 \cdot V_{t-1} + (1-\beta_2) \cdot g_t^2$

Adam

adaptive + Momentum

结合了一阶动量和二阶动量

- 一阶动量：$m_t = \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot g_t$
- 二阶动量：$V_t = \beta_2 \cdot V_{t-1} + (1-\beta_2) \cdot g_t^2$

实际使用过程，参数默认：$\beta_1 = 0.9; \beta_2 = 0.999$

Comparison

- Adam收敛问题
  - 由于RMSProp和Adam的二阶动量是固定时间窗口的累积，随着时间窗口变化，遇到的数据可能发生巨变，是的V_t可能会时大时小，不是单调变化。这在后期的训练有可能导致模型无法收敛。
    - 修正方法：$V_t = max(\beta_2 \cdot V_{t-1} + (1-\beta_2) \cdot g_t^2, V_{t-1})$
- 错过全剧最优解问题
  - **自适应学习率算法可能会对前期的特征过拟合，后期才出现的特征难以纠正前期的拟合效果**
    - **修正方法：充分shuffle数据集**
  - 尽管收敛速度快，但是收敛的效果没有SGD好。主要因为后期的学习速率太低了，影响有效收敛。

    - 修正方法：对Adam的学习率下界做约束。

- 核心差异：下降方向
    - SGD下降方向就是该位置的梯度的反方向
    - 带一阶动量的SGD下降方向就是该位置一阶动量的方向
    - 自适应学习率算法每个参数设定了不同的学习率，在不同维度上设定不同的步长。因此其下降方向为scaled的一阶动量方向

- Adam + SGD组合策略
    - 继承了Adam的快速收敛和SGD的精度收敛

- 数据是稀疏的(分类问题)，可以优先考虑自适应学习率算法。回归问题SGD通常更好

Batch Normalization

目的：

- 原论文：BN是为了减少 Internal Convariate Shift（训练集数据分布与预测集数据分布不一致）
- MIT新论文：BN的输入归一化使得优化梯度更加平滑，震荡更少。

优势：

1. **减少了人为选择参数**。在某些情况下可以取消 dropout 和 L2 正则项参数,或者采取更小的 L2 正则项约束参数；
2. **减少了对学习率的要求**。现在我们可以使用初始很大的学习率或者选择了较小的学习率，算法也能够快速训练收敛；
3. 可以不再使用局部响应归一化。BN 本身就是归一化网络(局部响应归一化在 AlexNet 网络中存在)
4. **破坏原来的数据分布，一定程度上缓解过拟合**（防止每批训练中某一个样本经常被挑选到，文献说这个可以提高 1% 的精度）。
5. **减少梯度消失，加快收敛速度**，提高训练精度。

算法流程：

下面给出 BN 算法在训练时的过程

输入：上一层输出结果 X=x1,x2,...,xm，学习参数 γ,β

1. 计算上一层输出数据的均值，其中，m 是此次训练样本 batch 的大小。

$$\mu_\beta = \frac{1}{m}\sum_{i=1}^{m}(x_i)$$

2. 计算上一层输出数据的标准差

$$\sigma_\beta^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_\beta)^2$$

3. 归一化处理，得到

$$\hat{x}_i = \frac{x_i + \mu_\beta}{\sqrt{\sigma_\beta^2 + \epsilon}}$$

其中 ϵ 是为了避免分母为 0 而加进去的接近于 0 的很小值

4. 重构，对经过上面归一化处理得到的数据进行重构，得到

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

其中，γ,β 为可学习参数。

注：上述是 BN 训练时的过程，但是当在投入使用时，往往只是输入一个样本，没有所谓的均值 μβ 和标准差 σ2β。此时，均值 μβ 是计算所有 batch μβ 值的平均值得到，标准差 σ2β 采用每个batch σ2β 的无偏估计得到。