Coresets for Data-efficient Training of Machine Learning Models

主要内容

为了在保证系统的结果精度不下降的同时大幅减小构建系统的花费，本文提出了**CRAIG**(Coresets for Accelerating ncremental Gradient descent)*方法来选择一个带权重的训练数据子集(Coresets)，其中权重为训练时每个元素对应的训练步长。这个子集可以用于估计在全数据集上训练所得到的梯度，使用这个子集进行训练可以使模型在训练时获得一个与子集大小成反比的加速。

问题

首先构建一个目标函数L用于选取大小为r的子集S。这个目标函数找到最小的子集$S \subseteq V$t，使得对于权重空间W内所有的权重w，这个子集上得到的梯度加权和与在全集上得到的梯度和所有权重最大差值不超过ε。（对于不同的权重，梯度会不同。目标函数需要保证对于所有的权重，梯度差值都小于ε。）

$$S^* = \arg \min_{\substack{S \subseteq V, \\ \gamma_j \geq 0 \ \forall j \in |S|}} |S|, \quad \text{s.t.} \quad \max_{w \in W} \left\| \sum_{i \in V} \nabla f_i(w) - \sum_{j \in S} \gamma_j \nabla f_j(w) \right\| \leq \varepsilon. \tag{1}$$

这个优化问题无法通过这个目标函数被解决，原因如下：

- 这个目标函数的梯度差部分需要对权重空间W中的每一个权重w求梯度函数$f_i(w)$，这样做太贵了。
- 即使最大梯度差很好找到，这个问题仍然是NP-hard问题。因为它涉及到计算V的所有子集（总共有$2^{|V|}$个子集）所对应的目标函数值$L(S)$。

方法

利用Upper-bound估计误差解决问题1：快速找到目标函数中的梯度差

本文通过不断寻找梯度差的**Upper-bound**，将求梯度差的问题转变为与**权重和梯度无关**的问题。最终可以通过常数级别的计算解决。具体方法如下：

通过一个映射$\delta_w(i) \in \arg\min_{j \in S} \|\nabla f_i(w) - \nabla f_j(w)\|$，将全集$V$中每个元素$i$映射到子集$S$中梯度最相似相似元素$j$。而后再利用三角不等式，找到目标函数的**第一个Upper-bound**。

$$\min_{S \subseteq V} \left\| \sum_{i \in V} \nabla f_i(w) - \sum_{j \in S} \gamma_j \nabla f_j(w) \right\| \leq \sum_{i \in V} \min_{j \in S} \|\nabla f_i(w) - \nabla f_j(w)\| \tag{2}$$

定义**第二个Upper-bound** $d_{i,j}$：对每一个(i,j)对，$d_{i,j}$为他们的梯度差在权重空间内的最大值。

$$d_{ij} \triangleq \max_{w \in W} \|\nabla f_i(w) - \nabla f_j(w)\| \tag{3}$$

通过这个新的Upper-bound $d_{i,j}$， 将目标函数转化为与**权重无关**的形式：

$$S^* = \arg\min_{S \subseteq V} |S|, \quad \text{s.t.} \quad L(S) \triangleq \sum_{i \in V} \min_{j \in S} d_{ij} \leq \varepsilon \tag{4}$$

上面的公式解释为找到一个子集S以及对应的映射关系$\delta_w(i)$，最小化$\nabla f_i(w)$和$\nabla f_j(w)$在权重空间内最的的差$d_{i,j}$，并使得这些差的和小于ε。

随后根据之前的研究结果设置**第三个Upper-bound**将梯数据点之间的梯度差用常数限制，转化为与**梯度无关**的形式：

$$\begin{aligned} \forall w, i, j \quad & \|\nabla f_i(w) - \nabla f_j(w)\| \leq d_{ij} \\ & \leq \max_{w \in W} \mathcal{O}(\|w\|) \cdot \|x_i - x_j\| = \text{const.} \|x_i - x_j\| \end{aligned}$$

最终得到一个与**权重和梯度都无关的Upper-bound**，这意味着估计在全集训练梯度和在coreset中训练梯度误差可以独立于优化过程，作为训练前的预处理过程。

开发CRAIG算法解决问题2: 如何高效找到近似最佳子集S

这篇文章将找最佳子集S的问题转化为***submodular set cover problem***（次模函数覆盖问题），并使用***Greedy algorithm***（贪心算法）找到近似最优解。具体方法如下：

次模函数：

$$F(S \cup \{e\}) - F(S) \geq F(T \cup \{e\}) - F(T)), \text{for any } S \subseteq T \subseteq V \text{and } e \in V \setminus T \tag{5}$$

这个函数可以解释为在一个子集中添加一个新的元素对这个函数的增益大于在这个在全集中添加一个新元素带来的增益。（边际递减效应：子集中添加元素的边际效用更高）

如何转换成次模函数?

用之前的目标函数（最大梯度差总和）组合为一个单调递增的次模函数覆盖函数：

$$F(S) = L(\{s_0\}) - L(S \cup \{s_0\}) \tag{6}$$

这个函数定义为增加一个子集S后，最大梯度差总和的下降值，其中s_0为辅助元素。

目标子集S可以使目标函数值小于ε，即$L(S \cup \{s_0\}) \leq \varepsilon$。带入上面的公式可以得到目标子集在次模函数上目标：

$$S^* = \arg\min_{S \subseteq V} |S|, \quad \text{s.t.} \quad F(S) \geq L(\{s_0\}) - \varepsilon \tag{7}$$

将次模函数的边际收益定义为$F(e|S) = F(S \cup \{e\}) - F(S) \geq 0$，在贪心算法的每次迭代中，选择边际效用最高的一个元素加入子集。每个元素对应的训练步长γ_j为映射到子集中j元素的全集中i元素的数量。

算法

Algorithm 1 CRAIG (CoResets for Accelerating Incremental Gradient descent)

Input: Set of component functions f_i for $i \in V = [n]\}$.
Output: Subset $S \subseteq V$ with corresponding per-element stepsizes $\{\gamma\}_{j \in S}$.

$S_0 \leftarrow \emptyset, s_0 = 0, i = 0.$
while $F(S) < L(\{s_0\}) - \epsilon$ **do**
  $j \in \arg\max_{e \in V \setminus S_{i-1}} F(e|S_{i-1})$
  $S_i = S_{i-1} \cup \{j\}$
  $i = i + 1$
end while
for $j = 1$ to $|S|$ **do**
  $\gamma_j = \sum_{i \in V} \mathbb{I}\left[j = \arg\min_{s \in S} \max_{w \in \mathcal{W}} \|\nabla f_i(w) - \nabla f_s(w)\|\right]$
end for

实验&结论

对于**凸函数问题**的实验证明了：

- **CRAIG**选出的数据集可以高效的最小化损失函数，而随机选择相同大小的数据集不可以。
- **CRAIG**具有良好的泛化表现，与全数据集训练能达到相同效果。
- **CRAIG**可以显著加速模型训练。
- **CRAIG**可以精准估计梯度。

对于**非凸函数问题**的实验证明了：

- **CRAIG**不仅可以提升速度，也可以提升精度。
- **CRAIG**对神经网络来说数据高效。
- 选择较小的数据集并多次更新效果更好。
- 随着训练，算法会选择更难学习的数据。