dlrm2

jiafeng5513

November 2, 2022

1 Data preprocess

原始文件是 24 个 tsv 文件，每行为一条数据，label、dense、sparse 依次排列。label 是一个 int，dense 是最多 13 个 int，其中可能有空值，sparse 是最多 26 个 int，其中可能有空值。数据处理：

1. 把 label，dense，sparse 分开，空值用 0 填充。
2. $dense = ln(dense + 3)$, 转为 float32 存储。
3. sparse 每一列单独处理，转换为连续数字，即每出现一个不同的数字，就替换为 id，同时 id+1，每遇到一个已经出现过的数字，就换成它第一次出现时使用的 id。id 从 2 开始。
4. 打乱 day_0 到 day_22，作为训练集，保留 day_23 作为测试集。
5. 向模型供给数据时，dense 和 label 直接供给，sparse 转换为 KeyedJaggedTensor。

KeyedJaggedTensor：

1. value：sparse$[B, 26]$ 列优先存储成一维。
2. offset：list，$form\ 0\ tobatch\_size * 26 + 1$。
3. stride: $batch\_size$。
4. length: list, $batch\_size * 26$ 个 1。
5. key：list of string，from cat_0 to cat_25。
6. length_per_key: list，26 个 $batch\_size$。
7. offset_per_key: list，list $[from\ 0\ to\ 26] * batch\_size$。
8. 方法 to_dict()：返回要给 dict，keys from cat_0 to cat_25，value 有三个字段：
   - _values 是一个一维 tensor，就是原始的 sparse 对应 cat_x 的那一列值
   - _length 是一个长度为 batch_size 的一维 tensor，都是 1，
   - _offset 是一个长度为 batch_size+1 的一维 tensor，从 0 到 batch_size

如果使用 mulit_hot_embedding，供给给后端的 sparse 数据会经过二次处理，该处理的具体过程如下：

1. 初始化需要额外的超参数：
   - $multi\_hot\_min\_table\_size$：只有那些 $num\_embedding >= multi\_hot\_min\_table\_size$ 的 $sparse\_feature$ 会经历额外处理。
   - $multi\_hot\_size$：被额外处理的那些 sparse_feature，被处理之前是 $[batch\_size,]$，处理完了是 $[batch\_size * multi\_hot\_size,]$。
   - $collect\_multi\_hot\_freqs\_stats$：是否统计频率。
   - $multi\_hot\_distribution\_type$：初始化 $weight$ 用的分布模式。
2. 初始化 $weight$：$weight[i]$ 是一个 $shape = [num\_embeddings[i], multi\_hot\_size]$ 的矩阵，使用 i 作为随机数种子，使用超参指定的分布进行初始化。
3. 初始化 $offset\_n$：
   (a) 总长度是 $num\_sparse\_features * batch\_size + 1$。

(b) 初始化一个 $lso = ones(sparse\_features\_num * batch\_size)$。

(c) $if\ feature[i]\ has\ num\_embedding >= multi\_hot\_min\_table\_size, lso[i * B : (i+1) * B] = multi\_hot\_size$。

(d) 在 lso 的最前面加一个 0。

(e) $offset\_n[0] = lso[0], offset\_n[i] = lso[i] + offset\_n[i-1]$。

4. 处理数据

(a) 设转换完的数据为 ls_n，转换前的数据为 ls_i，注意 $ls\_i.shape = [num\_sparse\_features, batch\_size]$。

(b) 如果 $num\_embeddings[i] < multi\_hot\_min\_table\_size$，则 $ls\_n[i] = ls\_i[i]$, 注意 $shape = [batch\_size,]$。

(c) 如果 $num\_embeddings[i] >= multi\_hot\_min\_table\_size$, 则 $ls\_n[i] = mulit\_hot\_i$, 注意 $shape = [batch\_size * multi\_hot\_size$

(d) $mulit\_hot = torch.nn.functional.embedding(ls\_i[i], weight[i])$，$shape = [batch\_size, multi\_hot\_size]$。

(e) 把第一列替换为原始的 sparse：$mulit\_hot[, 0] = ls\_i[i]$。

(f) $mulit\_hot\_i = mulit\_hot.rehsape(-1)$，更新 $ls\_n[i]$。

(g) 此时如果需要统计频率，则分别计数 $ls\_n[i]$ 和 $ls\_i[i]$ 中 unique 的元素的数量，设频率计数数组为 freq，则 freq 有 $num\_sparse\_features$ 行，每行中存储一个数组，记录 $feature[i]$ 中 unique 元素出现的数量，处理前后分别统计。

(h) 把 ls_n 转换成一维的。

5. 用 ls_n 替换 dataloader 的 sparse.values，用 $offset\_n$ 替换 dataloader 的 sparse.offset。

6. mulit_hot 处理有什么效果：

(a) 一个 batch 里面仍然含有 $num\_sparse\_features * batch\_size$ 个 sparse feature。

(b) 对于某些 feature，它可能有巨大的取值范围，我们给取值范围设置了一个阈值。

(c) 对于那些取值范围在阈值之下的 feature，它传给后端的每一个单独的 sparse feature 就是一个数，和以前一样。

(d) 对于那些取值范围在阈值之上的 feature，我们预先对他进行了一步 embedding，转成了我们设置好的一个的长度。

(e) 对于那些经历过额外的 embedding 的 feature，它传给后端的每一个单独的 sparse feature 是多个数。

(f) 因为有些 sparse feature 是多个数，因此 offset 是不均匀的，在没经历过变换的区段，相邻的 offset 差值为 1，而在经历过变换的区段，相邻的 offset 之间差 $multi\_hot\_size$。

(g) 虽然统计了频率，不过除了把频率保存成文件之外，并没有任何其他的用途。

(h) 此处的 embedding 的 weight 是无法更新的。

举个例子:
设 $B = 2$，$sparse\_features\_num = 4$, 这四个 feature 的 $num\_embeddings = [6, 7, 5, 9]$,
$multi\_hot\_min\_table\_size = 8$。$multi\_hot\_size = 3$. 我们取一个 batch 的数据：

$$\begin{bmatrix} I_{0,0} & I_{0,1} & I_{0,2} & I_{0,3} \\ \\ I_{1,0} & I_{1,1} & I_{1,2} & I_{1,3} \end{bmatrix}$$

在 KeyedJaggedTensor 中，value 字段是列优先存储的：

$$[I_{0,0}, I_{1,0}, I_{0,1}, I_{1,1}, I_{0,2}, I_{1,2}, I_{0,3}, I_{1,3}]$$

对应的 offset 是：

$$[0, 1, 2, 3, 4, 5, 6, 7, 8]$$

根据规则，第 4 列会被额外处理，因为第 4 列的 $num\_embeddings = 9 > multi\_hot\_min\_table\_size = 8$，前 3 列会保持不变。第 4 列有 $batch\_size$ 个元素，处理完了会有 $batch\_size * multi\_hot\_size$ 个元素。

$$\begin{bmatrix} I_{0,3} \\ \\ I_{1,3} \end{bmatrix} \rightarrow \begin{bmatrix} I^0_{0,3} & I^1_{0,3} & I^2_{0,3} \\ \\ I^0_{1,3} & I^1_{1,3} & I^2_{1,3} \end{bmatrix}$$

此时这个 batch 的数据就变成了这样，注意，这仍然是一个 2×4 的矩阵，只不过其中两个元素从 1 个数字变成了 3 个数字：

$$\begin{bmatrix} I_{0,0} & I_{0,1} & I_{0,2} & I_{0,3}^0 \\ & & & I_{0,3}^1 \\ & & & I_{0,3}^2 \\ I_{1,0} & I_{1,1} & I_{1,2} & I_{1,3}^0 \\ & & & I_{1,3}^1 \\ & & & I_{1,3}^2 \end{bmatrix}$$

在 KeyedJaggedTensor 中，value 字段是列优先存储的：

$$[I_{0,0}, I_{1,0}, I_{0,1}, I_{1,1}, I_{0,2}, I_{1,2}, I_{0,3}^0, I_{0,3}^1, I_{0,3}^2, I_{1,3}^0, I_{1,3}^1, I_{1,3}^2]$$

对应的 offset 是：

$$[0, 1, 2, 3, 4, 5, 6, 9, 12]$$

那么多出来的数字是怎么来的呢：之前初始化了 weight[9, 3]，9 是因为这一列 feature 的 $num\_embeddings = 9$，3 是因为 $multi\_hot\_size = 3$，初始化使用的随机数种子是 3，因为这个 feature 在 4 个 sparse feature 中的下标是 3。$w_{i,j} \in [0, 9)$：

$$weight = \begin{bmatrix} w_{0,0} & w_{0,1} & w_{0,2} \\ w_{1,0} & w_{1,1} & w_{1,2} \\ w_{2,0} & w_{2,1} & w_{2,2} \\ w_{3,0} & w_{3,1} & w_{3,2} \\ w_{4,0} & w_{4,1} & w_{4,2} \\ w_{5,0} & w_{5,1} & w_{5,2} \\ w_{6,0} & w_{6,1} & w_{6,2} \\ w_{7,0} & w_{7,1} & w_{7,2} \\ w_{8,0} & w_{8,1} & w_{8,2} \end{bmatrix}$$

计算 $torch.nn.functional.embedding([I_{0,3}, I_{1,3}], weight)$，然后再把第一列替换成 $[I_{0,3}, I_{1,3}]$，即：

$$[I_{a,3}^0, I_{a,3}^1, I_{a,3}^2] = [I_{a,3}, w_{I_{a,3},1}, w_{I_{a,3},2}], a \in [0, 1]$$

其中 torch 中的这个函数的作用就是用输入的 index 取 weight 中对应的行，然后拼接成输出。
$[I_{0,3}, I_{1,3}]$ 的取值范围和 $[I_{0,3}^0, I_{0,3}^1, I_{0,3}^2, I_{1,3}^0, I_{1,3}^1, I_{1,3}^2]$ 的取值范围完全一样，都是 $[0, 9)$。