Optimizing Stochastic Gradient Descent Over Normalized Data

When the attribute tables cannot all entirely fit into the memory, multi-table joins introduce an optimization problem to HPIS.

Problem Statement. Suppose there are n attribute tables with size $r_i, i = 1, 2, ..., n$. Total memory is m. We assume $\sum_{i=1}^{n} r_i > m$. The amount of memory allocated to each table as m_i.

$$\begin{aligned} \min_{m_i} \quad & \prod_{i=1}^{n} \frac{h.r_i}{m_i} \\ \text{s.}t. \quad & \sum_{i=1}^{n} m_i = m, \\ & 1 \leq m_i \leq m - n \end{aligned}$$

Greedy Approach. We relax the problem to:

$$\min_{m_i} \quad \prod_{i=1}^{n} \frac{h.r_i}{m_i}$$

Since $h.r_i$ is fixed, this relaxed problem is equivalent to

$$\begin{aligned} \max_{m_i} \quad & \prod_{i=1}^{n} m_i \\ \text{s.}t. \quad & \sum_{i=1}^{n} m_i = m, \\ & 1 \leq m_i \leq m - n \end{aligned}$$

For this problem, when $m_i = \bar{m}$, we have the optima. Our greedy approach starts from here and make minimal adjustments to approach optima of the original problem.

For tables that are smaller than the initial mean, they are given memory no more than their size. The rest memory and the rest R tables become a subproblem same as the original problem. We recalculate the mean based on the rest memory and the rest R tables. Doing

this recursively, we stop when the rest R tables are all larger than the current mean.
Let P_i be the current number of partitions of R_i. For the rest of this approach, we are going to introduce 2 measures.
$cost_i = \frac{r_i \mod m_i}{P_i - 1}$
$surplus_i = \frac{m_i - r_i \mod m_i}{P_i} - 1$
R_i needs *cost* more pages to reduce its number of partitions. And it can give at most *surplus* more pages to not increase its number of partitions.
[t] Array $v \leftarrow$ 2 attribute tables' info $\bar{m} \leftarrow \frac{m}{n}$
there is v_i smaller than $\bar{m}$ Delete v_i
$n \leftarrow n - 1$
$\bar{m} = \frac{m - v_i}{n}$ v is not empty and $\sum surplus > cost_{lowest}$
Choose v_i with the lowest cost.
$v_{target} = v_i$.
Erase v_i from the array v.
Sort v on surplus.
i=1 to $v.size()$ $cost_{target} -= min(cost_{target}, surplus_i)$
update v_i
$cost_{target} == 0$ break;

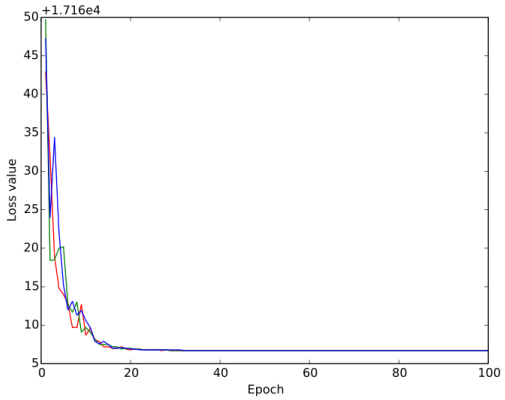

Figure 1: Comparison of convergence speed with 2 R tables

Comparing performance of different partition algorithms. The plots show the learning convergence speed of three partition methods: optimal solution which looks at all possible partitions, greedy solution mentioned above, and partitioning proportional to the size of each R table. The experiment is run on synthetic data with 2, 4, and 6 R tables. In cases we tested, all algorithms are able to converge quickly after a few epochs, and we also found that in most cases the difference in total number of chunks in R tables is not significant between the greedy search and optimal search.

4.3.2 Shared-nothing Parallelism

4.4 Mini-Batch Gradient Descent

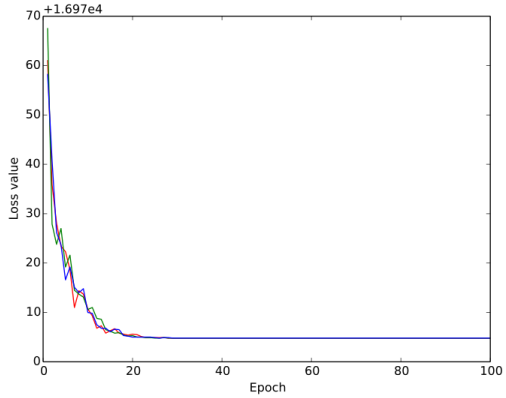

Figure 2: Comparison of convergence speed with 4 R tables

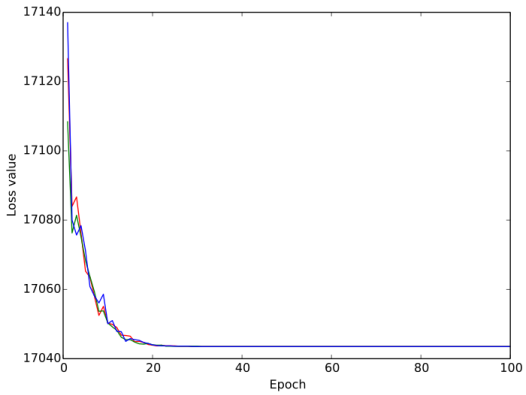

Figure 3: Comparison of convergence speed with 6 R tables

5. EXPERIMENTS

We implement our HPIS algorithm with user-defined functions in postgreSQL. It is run on machines with 2 Intel E5-2630 processors and 128 GB of ECC RAM. In our experiment we have two memory buffer settings on the database when comparing materialized learning scheme with HPIS: when memory is constrained and cannot hold the full R table, hence the HPIS algorithm will split S and R into multiple chunks. In the other setting we set memory buffer to be large enough to hold R, but not enough to include the S table, and in this case we are essentially comparing Materialize with IS. We also compare the performance when different shuffling schemes: when we shuffle the S table after each epoch, or only once at the beginning.The extension runs the SGD over 100 epochs, and performance is compared over various tuple ratios (n_S:n_R),feature ratios (d_R:d_S),and a factor of chunk sizes.
With small memory setting, we see that HPIS does

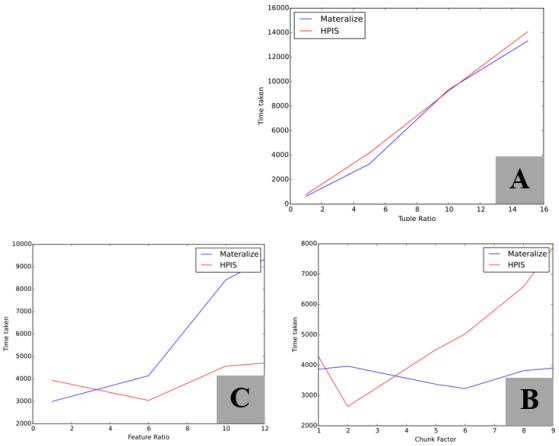

Figure 4: Performance comparison with small memory buffer, shuffling after each epoch

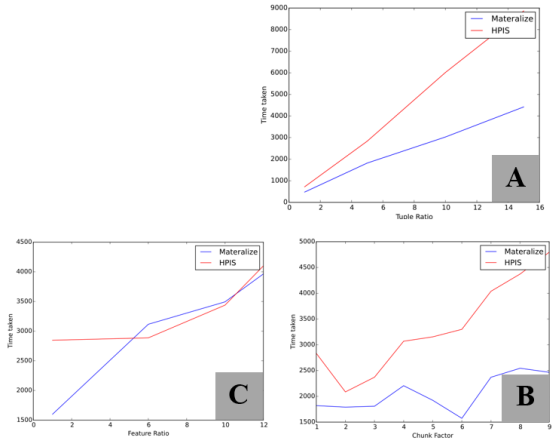

Figure 5: Performance comparison with constrained memory buffer, shuffling only at the beginning

not gain much in terms of increasing tuple ratio, but has a clear advantage as the feature ratio increases. As size of R grows, performance of HPIS wins by avoiding random I/O cost of matching hash entries in R using HPIS. We also see more benefit of HPIS in shuffle always scheme, as we save the random I/O cost of shuffling large table T.

5.0.0.1 *Effect of data shuffling when data is skewed.*

Here we consider two ways data is skewed. In one way, we get all the positive samples before we hit the first negative sample. In the other way, the two classes of sample appear in a round-robin fashion: we see a small group of positive data, then a group of negative data, and so on. In this experiment we study in both cases, how does the number of partition chunks affect

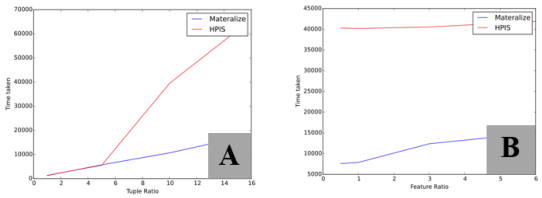

Figure 6: Performance comparison with large memory buffer, shuffling after each epoch

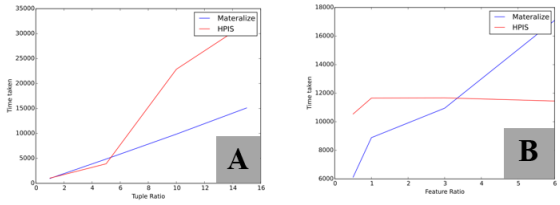

Figure 7: Performance comparison with large memory buffer, shuffling only at the beginning

the convergence speed. We present three of the plots in each case with 4, 8, and 12 partition chunks. We find that even with skewed data, the convergence speed of HPIS remains stable when number of partition chunks increases.

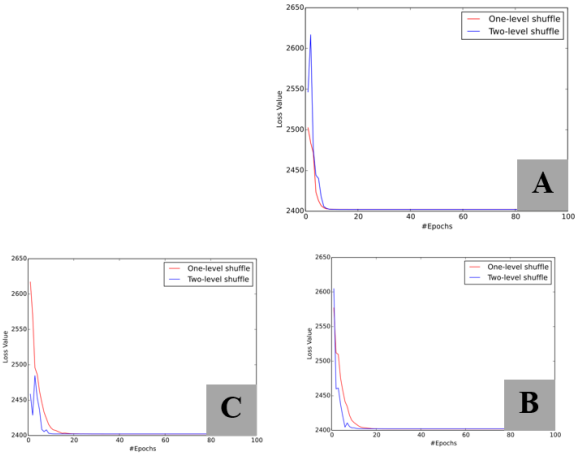

Figure 8: Convergence speed with 4,8,12 partition chunks on data separated into positive and negative groups

Real Datasets

Experimental Setup

5.1 High-level Performance and Quality

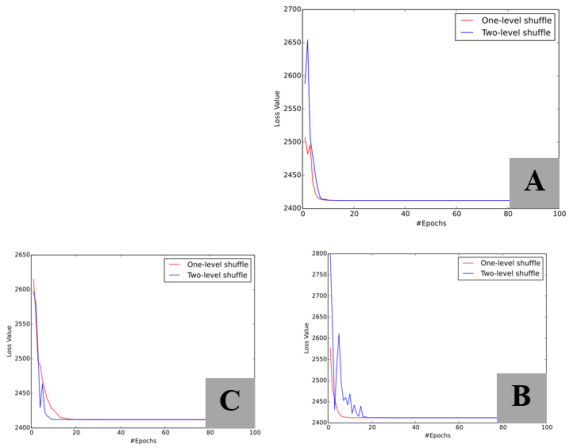

Figure 9: Convergence speed with 4,8,12 partition chunks on data in groups and round-robin fashion

5.2 Performance Drill-Down

5.3 Evaluation of Extensions

6. RELATED WORK

7. CONCLUSION AND FUTURE WORK