Batch-Expansion Training:
An Efficient Optimization Framework

Michał Dereziński  Dhruv Mahajan  S. Sathiya Keerthi  S. V. N. Vishwanathan  Markus Weimer

Large-scale convex optimization

Goal: minimize \( f(w) \equiv \frac{1}{N} \sum_{i=1}^{N} f_i(w) + \frac{\lambda}{2} \|w\|^2 \)

Challenges of large-scale optimization:
- Data does not fit on a single machine
- Disk-access is slow
- Not all machines are available immediately

Batch vs stochastic optimization

Batch methods:
- Slow iterations
- Need all data from the start
- Easy to parallelize
- No tuning needed

Stochastic methods:
- Quick iterations
- Adapt to gradual data availability
- Tricky to parallelize
- Require tuning

Batch-Expansion Training (BET)

Goal: Batch optimization without the tradeoffs
Idea: We do not need all data to start converging

Pick initial model \( w_0 \)
Load first \( n_1 \) data points
for \( t = 1,..T \) do
\( w_\text{slow} \leftarrow \text{run} \ n_1 \ \text{steps on} \ n_1 \ \text{data points} \)
\( n_{t+1} \leftarrow n_1 n_t \) (increase data size)
Load \( n_{t+1} - n_t \) data points
end for
return \( w_T \)

- Adapts to gradual data availability
- Easy to parallelize
- No tuning needed
- Works with any optimizer

Estimation error of expanding batches

Error

Fixed Batch
Expanding Batch

Estimation Error: \( \|\hat{f} - f\| \) (\( \hat{f} \) is computed from a batch)

Improved data-access complexity

Linear optimizer: (for any \( w \) and \( w^* = \text{argmin}_w f(w) \))
\[ f(\text{Update}(w)) - f(w^*) \leq (1 - (1/\kappa))(f(w) - f(w^*)) \]

What is the cost of computing \( \hat{w} \) such that \( f(\hat{w}) - f(w^*) \leq \epsilon \)?

Cost of batch optimization:
\[ \frac{\text{data size} \times \text{convergence rate} \times O(\log(1/\epsilon))}{\kappa} = O(N \kappa \log(1/\epsilon)) \]

Optimal BET: Double the batch every \( O(\kappa) \) iterations

Cost of optimal BET:
\[ \frac{\text{1st batch} \times (1 + 2 + 4 + \cdots + 2^{\log(N/n_t)})}{O(N)} \times \frac{O(\kappa)}{O(N)} = O(N \kappa) \]

Conclusion: BET saves a factor of \( \log(1/\epsilon) \) over Batch!

Two-track algorithm

Question: When should we double the data size in practice?
Idea: Run two parallel optimization tracks with different data sizes

<table>
<thead>
<tr>
<th>Stage ( t )</th>
<th>Data</th>
<th>Loss</th>
<th>Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fast track</td>
<td>( \hat{f}_{t-1} )</td>
<td>( w_{fast} )</td>
<td></td>
</tr>
<tr>
<td>Slow track</td>
<td>( \hat{f}_t )</td>
<td>( w_{slow} )</td>
<td></td>
</tr>
</tbody>
</table>

When \( \hat{f}_t(w_{slow}) < \hat{f}_t(w_{fast}) \), double the data:

Stage \( t + 1 \) Data | Loss | Model |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Fast track</td>
<td>( \hat{f}_{t+1} )</td>
<td>( w_{fast} )</td>
</tr>
<tr>
<td>Slow track</td>
<td>( \hat{f}_{t+1} )</td>
<td>( w_{slow} )</td>
</tr>
</tbody>
</table>

... until full dataset reached

BET in parallel and distributed settings

1. BET with Limited-memory BFGS as optimizer
2. Batch with Limited-memory BFGS as optimizer
3. ParSGD (Parallelized Stochastic Gradient Descent)

Dataset, size | Train/Test | Dim. | \( \lambda \) |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>covtype, 19GB</td>
<td>0.5M/69k</td>
<td>170k</td>
<td>1e-6</td>
</tr>
<tr>
<td>splice-site, 1.5TB</td>
<td>25M/4.6M</td>
<td>11.7M</td>
<td>2e-10</td>
</tr>
</tbody>
</table>

log RFVD: \( \log \left( \frac{(f(w) - f(w^*))}{f(w^*)} \right) \)