Asynchronous Multi-Task Learning

Inci M. Baytas, Ming Yan, Anil K. Jain and Jiayu Zhou

December 14th, 2016
Outline

1. Introduction
2. Solving Regularized MTL
3. Distributed MTL
4. Asynchronous Multi-Task Learning (AMTL)
5. Experimental Results
6. Conclusion and Future Work
Introduction

- Many real-world machine learning applications involve multiple learning tasks.
- Tasks are related.
  - Shared hidden layers
  - Shared parameters
  - Regularization based; shared subspace, joint feature.
- Multi-Task Learning (MTL)
  - Simultaneously learn related tasks.
  - Perform inductive knowledge transfer among tasks.
Introduction

• How to pose task relatedness?
  • Subspace learning; regularized MTL.

\[ \min_W \sum_{t=1}^{T} \ell_t(w_t) + \lambda \|W\|_* \]

where \( W \in \mathbb{R}^{d \times T} \) is the model matrix, and \( \lambda \) is the regularization parameter.
Solving Regularized MTL

- Centralize the data
- Apply forward-backward splitting
  - Calculate gradient of loss function \( (\sum_{t=1}^{T} \ell_t(w_t)) \)
  - Move in the negative gradient direction \( \hat{W} = W^k - \eta \nabla_{W^k} \sum_{t=1}^{T} \ell_t(w_t) \).
  - Apply proximal mapping to obtain the updated model matrix.
    - Project the intermediate model matrix \( \hat{W} \) to the solution domain.
- **Limitation:** Data centralization is not always possible!
Regularized MTL: Limitations

- Data servers are in different locations.
- Data cannot be frequently transferred over the network.
  - Privacy reasons
  - Communication cost; large scale datasets and limited bandwidth
- Solution:
  - Distributed MTL
Distributed MTL

- **Synchronized distributed MTL (SMTL)**
  - One central node and $T$ task nodes.
  - **In each task node:**
    - $\hat{w}_t = w_t^k - \eta \nabla \ell_t(w_t^k)$
    - Send $\hat{w}_t$ to central node.
  - **In central node:**
    - Wait for each task node to finish its computations and send $\hat{w}_t$.
    - Construct the model matrix $\hat{W}$ and perform proximal mapping.
Synchronous MTL: Limitations

- Slow due to data imbalance.
- Slow due to communication delays.
- Not robust against network failures.

**Proposed Solution:** Asynchronous MTL (AMTL)
- Central node does not wait for all task nodes.
- Robust against communication delays.
- Linear convergence is guaranteed.
Asynchronous MTL Overview
Asynchronous MTL Framework

• AMTL framework is based on AROCK \(^1\), an asynchronized coordinate descent framework.

• In AMTL, each model is a block of coordinates.

• A generic Krasnoselskii–Mann (KM) iteration framework to solve fixed point problems. ⇒ Linear convergence

• KM iteration update rule:

\[
x^{k+1} = x^k + \eta_k \left( F(x^k) - x^k \right)
\]

where \(F\) is a fixed point operator.

Optimization

- AMTL uses backward-forward splitting (BFS) as the fixed point operator.

\[
\begin{align*}
    w_{t}^{k+1} &= w_{t}^{k} + \eta_{k} \left( F_{BFS} \left( \hat{w}^{k} \right) - w_{t}^{k} \right) \\
    F_{BFS} \left( \hat{w}^{k} \right) &= \left( \text{Prox}_{\eta_{t}\lambda} \left( \hat{w}^{k} \right) \right)_{t} - \eta \nabla \ell_{t} \left( \left( \text{Prox}_{\eta_{t}\lambda} \left( \hat{w}^{k} \right) \right)_{t} \right)
\end{align*}
\]

- Why BFS?
  - Reduced communication between central and task nodes in each iteration.
  - Changing order does not effect convergence.
Backward-Forward Splitting

- Forward step (gradient step) can be decoupled.

\[ \nabla f(W) = \nabla \sum_{t=1}^{T} \ell_t(w_t) = [\nabla \ell_1(w_1), \ldots, \nabla \ell_T(w_T)]. \]

- However, backward step (proximal mapping) is not decoupled.

\[ \text{prox}_{\eta\lambda}(\hat{W}) = \arg \min_W \frac{1}{2\eta} \|W - \hat{W}\|_F^2 + \lambda \|W\|_*. \]

- Backward step should be performed in the central node.
- One extra backward step is needed at the end of the last iteration.
AMTL: Update Mechanism

Multi-Task Model

Central Server

Task Node 2

Task Node 3

Task Node 4

Receive Task Model
Send Task Gradient
Receive Task Model
Dynamic Step Size

• In reality, each task node does not have the same activation rate; **network delays!**

\[ w_{t}^{k+1} = w_{t}^{k} + c_{(t,k)} \eta_{k} \left( F_{BFS} (\hat{w}^{k}) - w_{t}^{k} \right) \]

• Longer the delay, larger the step size \((\eta_{k})\); a heuristic approach:

\[ c_{(t,k)} = \log \left( \max (\bar{\nu}_{t,k}, 10) \right) \]

where \( \bar{\nu}_{t,k} = \frac{1}{k} \sum_{i=z-k}^{z} \nu_{t}^{(i)} \); history of delays.
Experimental Results

- AMTL is implemented by using AROCK framework (MPI in C++).
- Distributed environment is simulated by using shared memory architecture; network delays are artificially introduced.
- Experiments were conducted using an Intel Core i5-5200U CPU (2.20GHz x 4) machine.
- Performance is limited by the hardware specifications of the machine.
Computation Time: AMTL v. SMTL

- Synthetic data
- Square loss
Datasets

- 5 binary classification tasks for MNIST such as 0 vs 9, 1 vs 8, . . .
- 4 binary classification tasks for MTFL (Multi-Task Facial Landmark) such as male vs female, smiling vs not smiling, . . .

**Table:** Datasets used in this paper.

<table>
<thead>
<tr>
<th>Data set</th>
<th>Number of tasks</th>
<th>Sample sizes</th>
<th>Dimensionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>School</td>
<td>139</td>
<td>22-251</td>
<td>28</td>
</tr>
<tr>
<td>MNIST</td>
<td>5</td>
<td>13137-14702</td>
<td>100</td>
</tr>
<tr>
<td>MTFL</td>
<td>4</td>
<td>2224-10000</td>
<td>10</td>
</tr>
</tbody>
</table>
**Table: Training time (sec.)** comparison of AMTL and SMTL for different datasets. Training time of AMTL is less than the training time of SMTL with different network settings.

<table>
<thead>
<tr>
<th>Network</th>
<th>School</th>
<th>MNIST</th>
<th>MTFL</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMTL-1</td>
<td>299.79</td>
<td>57.94</td>
<td>50.59</td>
</tr>
<tr>
<td>AMTL-1</td>
<td><strong>194.22</strong></td>
<td><strong>54.96</strong></td>
<td><strong>50.40</strong></td>
</tr>
<tr>
<td>SMTL-2</td>
<td>298.42</td>
<td>114.85</td>
<td>92.84</td>
</tr>
<tr>
<td>AMTL-2</td>
<td><strong>231.58</strong></td>
<td><strong>83.17</strong></td>
<td><strong>77.44</strong></td>
</tr>
<tr>
<td>SMTL-3</td>
<td>593.36</td>
<td>161.67</td>
<td>146.87</td>
</tr>
<tr>
<td>AMTL-3</td>
<td><strong>460.15</strong></td>
<td><strong>115.46</strong></td>
<td><strong>103.45</strong></td>
</tr>
</tbody>
</table>
Effect of Dynamic Step Size

**Table:** Final objective values of the synthetic dataset with 5 tasks under different network settings.

<table>
<thead>
<tr>
<th>Network</th>
<th>Without dynamic step size</th>
<th>Dynamic step size</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMTL-5</td>
<td>163.62</td>
<td>144.83</td>
</tr>
<tr>
<td>AMTL-10</td>
<td>163.59</td>
<td>144.77</td>
</tr>
<tr>
<td>AMTL-15</td>
<td>163.56</td>
<td>143.82</td>
</tr>
<tr>
<td>AMTL-20</td>
<td>168.63</td>
<td>143.50</td>
</tr>
</tbody>
</table>
Conclusion and Future Work

- AMTL is a time efficient distributed MTL framework.
- Dynamic step size can boost the convergence in real world network settings.
- AMTL implementation for real world settings.\(^2\)
- Stochastic gradient framework will also be investigated for AMTL.

\(^2\)https://github.com/illidanlab/AMTL
Acknowledgements

This research is supported by:

NSF under grant IIS-1565596, IIS-1615597, and DMS-1621798

ONR under grant N00014-14-1-0631
Thank you!

Questions?