Distributed Asynchronous Online Learning for Natural Language Processing

Kevin Gimpel     Dipanjan Das     Noah A. Smith
Introduction

- Two recent lines of research in speeding up large learning problems:
  - Parallel/distributed computing
  - Online (and mini-batch) learning algorithms:
    - stochastic gradient descent, perceptron, MIRA, stepwise EM

- How can we bring together the benefits of parallel computing and online learning?
Introduction

- We use **asynchronous** algorithms
  (Nedic, Bertsekas, and Borkar, 2001; Langford, Smola, and Zinkevich, 2009)

- We apply them to structured prediction tasks:
  - Supervised learning
  - Unsupervised learning with both convex and non-convex objectives

- Asynchronous learning speeds convergence and works best with small mini-batches
Problem Setting

- Iterative learning
  - Moderate to large numbers of training examples
  - Expensive inference procedures for each example
  - For concreteness, we start with gradient-based optimization

- Single machine with multiple processors
  - Exploit shared memory for parameters, lexicons, feature caches, etc.
  - Maintain one master copy of model parameters
Single-Processor Batch Learning

Parameters: $\theta_t$
Processors: $P_i$
Dataset: $D$
Single-Processor Batch Learning

\[
\begin{array}{|c|}
\hline
\theta \\
\hline
\mathcal{P}_1 \\
\hline
\end{array}
\]

Parameters: \( \theta_t \)
Processors: \( \mathcal{P}_i \)
Dataset: \( D \)
Single-Processor Batch Learning

<table>
<thead>
<tr>
<th>( \theta )</th>
<th>( \theta_0 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \mathcal{P}_1 )</td>
<td>( \mathbf{g} = \text{calc}(\mathcal{D}, \theta_0) )</td>
</tr>
</tbody>
</table>

\[ \mathbf{g} = \text{calc}(\mathcal{D}, \theta) : \]
Calculate gradient \( \mathbf{g} \) on data \( \mathcal{D} \) using parameters \( \theta \)

Parameters: \( \theta_t \)
Processors: \( \mathcal{P}_i \)
Dataset: \( \mathcal{D} \)
**Single-Processor Batch Learning**

$$\theta_1 = \text{up}(\theta_0, g)$$

<table>
<thead>
<tr>
<th>$\theta$</th>
<th>$\theta_0$</th>
<th>$\theta_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{P}_1$</td>
<td>$g = \text{calc}(\mathcal{D}, \theta_0)$</td>
<td>$\theta_1 = \text{up}(\theta_0, g)$</td>
</tr>
</tbody>
</table>

- **$g = \text{calc}(\mathcal{D}, \theta)$**: Calculate gradient $g$ on data $\mathcal{D}$ using parameters $\theta$.
- **$\theta_1 = \text{up}(\theta_0, g)$**: Update $\theta_0$ using gradient $g$ to obtain $\theta_1$.

**Parameters:** $\theta_t$
**Processors:** $\mathcal{P}_i$
**Dataset:** $\mathcal{D}$
# Single-Processor Batch Learning

<table>
<thead>
<tr>
<th>θ</th>
<th>θ₀</th>
<th>θ₁</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>P₁</strong></td>
<td>$g = \text{calc}(\mathcal{D}, \theta₀)$</td>
<td>$\theta₁ = \text{up}(\theta₀, g)$</td>
</tr>
</tbody>
</table>

- **g = calc(\mathcal{D}, \theta)**: Calculate gradient $g$ on data $\mathcal{D}$ using parameters $\theta$.
- **$\theta₁ = \text{up}(\theta₀, g)$**: Update $\theta₀$ using gradient $g$ to obtain $\theta₁$.

Parameters: $\theta_t$

Processors: $\mathcal{P}_i$

Dataset: $\mathcal{D}$
## Parallel Batch Learning

<table>
<thead>
<tr>
<th>θ</th>
<th>θ₀</th>
</tr>
</thead>
<tbody>
<tr>
<td>P₁</td>
<td>g₁ = calc(D₁, θ₀)</td>
</tr>
<tr>
<td>P₂</td>
<td>g₂ = calc(D₂, θ₀)</td>
</tr>
<tr>
<td>P₃</td>
<td>g₃ = calc(D₃, θ₀)</td>
</tr>
</tbody>
</table>

- Divide data into parts, compute gradient on parts in parallel

Parameters: \( \theta_t \)
Processors: \( P_i \)
Dataset: \( D = D₁ \cup D₂ \cup D₃ \)
Gradient: \( g = g₁ + g₂ + g₃ \)
## Parallel Batch Learning

<table>
<thead>
<tr>
<th>0</th>
<th>Time</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th></th>
<th>$\theta$</th>
<th>$\theta_0$</th>
<th>$\theta_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>$g_1 = \text{calc}(\mathcal{D}_1, \theta_0)$</td>
<td>$\theta_1 = \text{up}(\theta_0, g)$</td>
<td></td>
</tr>
<tr>
<td>$P_2$</td>
<td>$g_2 = \text{calc}(\mathcal{D}_2, \theta_0)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$P_3$</td>
<td>$g_3 = \text{calc}(\mathcal{D}_3, \theta_0)$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Divide data into parts, compute gradient on parts in parallel
- One processor updates parameters

Parameters: $\theta_t$
Processors: $P_i$
Dataset: $\mathcal{D} = \mathcal{D}_1 \cup \mathcal{D}_2 \cup \mathcal{D}_3$
Gradient: $g = g_1 + g_2 + g_3$
## Parallel Batch Learning

<table>
<thead>
<tr>
<th>£</th>
<th>£0</th>
<th>£1</th>
</tr>
</thead>
<tbody>
<tr>
<td>£0</td>
<td>£0 = calc(£1, £0)</td>
<td>£1 = up(£0, g)</td>
</tr>
<tr>
<td>£1</td>
<td>g1 = calc(£1, £0)</td>
<td>g1 = calc(£1, £1)</td>
</tr>
<tr>
<td>£2</td>
<td>g2 = calc(£2, £0)</td>
<td>g2 = calc(£2, £1)</td>
</tr>
<tr>
<td>£3</td>
<td>g3 = calc(£3, £0)</td>
<td>g3 = calc(£3, £1)</td>
</tr>
</tbody>
</table>

### Parameters:
- £t

### Processors:
- £i

### Dataset:
- £ = £1 ∪ £2 ∪ £3

### Gradient:
- £ = £1 + £2 + £3

- Divide data into parts, compute gradient on parts in parallel
- One processor updates parameters
### Parallel Synchronous Mini-Batch Learning

**Finkel, Kleeman, and Manning (2008)**

<table>
<thead>
<tr>
<th></th>
<th>$\theta_0$</th>
<th>$\theta_1$</th>
<th>$\theta_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{P}_1$</td>
<td>$g_1 = c(B^1_1, \theta_0)$</td>
<td>$\theta_1 = u(\theta_0, g)$</td>
<td>$g_1 = c(B^1_2, \theta_1)$</td>
</tr>
<tr>
<td>$\mathcal{P}_2$</td>
<td>$g_2 = c(B^2_1, \theta_0)$</td>
<td>$g_2 = c(B^2_2, \theta_1)$</td>
<td>$g_2 = c(B^2_3, \theta_2)$</td>
</tr>
<tr>
<td>$\mathcal{P}_3$</td>
<td>$g_3 = c(B^3_1, \theta_0)$</td>
<td>$g_3 = c(B^3_2, \theta_1)$</td>
<td>$g_3 = c(B^3_3, \theta_2)$</td>
</tr>
</tbody>
</table>

- **Parameters:** $\theta_t$
- **Processors:** $\mathcal{P}_i$
- **Mini-batches:** $B_t = B^1_t \cup B^2_t \cup B^3_t$
- **Gradient:** $g = g_1 + g_2 + g_3$

- **Same architecture, just more frequent updates**
## Parallel Asynchronous Mini-Batch Learning

*Nedic, Bertsekas, and Borkar (2001)*

<table>
<thead>
<tr>
<th>$\theta$</th>
<th>$\theta_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{P}_1$</td>
<td></td>
</tr>
<tr>
<td>$\mathcal{P}_2$</td>
<td></td>
</tr>
<tr>
<td>$\mathcal{P}_3$</td>
<td></td>
</tr>
</tbody>
</table>

0 \hspace{2em} \text{Time}

### Parameters:
- $\theta_t$

### Processors:
- $\mathcal{P}_i$

### Mini-batches:
- $\mathcal{B}_j$

### Gradient:
- $g_k$
**Parallel Asynchronous Mini-Batch Learning**

Nedic, Bertsekas, and Borkar (2001)

<table>
<thead>
<tr>
<th>$\theta$</th>
<th>$\theta_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{P}_1$</td>
<td>$g_1 = c(\mathcal{B}_1, \theta_0)$</td>
</tr>
<tr>
<td>$\mathcal{P}_2$</td>
<td>$g_2 = c(\mathcal{B}_2, \theta_0)$</td>
</tr>
<tr>
<td>$\mathcal{P}_3$</td>
<td>$g_3 = c(\mathcal{B}_3, \theta_0)$</td>
</tr>
</tbody>
</table>

Parameters: $\theta_t$

Processors: $\mathcal{P}_i$

Mini-batches: $\mathcal{B}_j$

Gradient: $g_k$
### Parallel Asynchronous Mini-Batch Learning

**Nedic, Bertsekas, and Borkar (2001)**

<table>
<thead>
<tr>
<th>$\theta$</th>
<th>$\theta_0$</th>
<th>$\theta_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{P}_1$</td>
<td>$g_1 = c(B_1, \theta_0)$</td>
<td>$\theta_1 = u(\theta_0, g_1)$</td>
</tr>
<tr>
<td>$\mathcal{P}_2$</td>
<td>$g_2 = c(B_2, \theta_0)$</td>
<td></td>
</tr>
<tr>
<td>$\mathcal{P}_3$</td>
<td>$g_3 = c(B_3, \theta_0)$</td>
<td></td>
</tr>
</tbody>
</table>

**Parameters:** $\theta_t$

**Processors:** $\mathcal{P}_i$

**Mini-batches:** $B_j$

**Gradient:** $g_k$
### Parallel Asynchronous Mini-Batch Learning

*Nedic, Bertsekas, and Borkar (2001)*

<table>
<thead>
<tr>
<th>Time</th>
<th>Parameters: $\theta_t$</th>
<th>Processors: $P_i$</th>
<th>Mini-batches: $B_j$</th>
<th>Gradient: $g_k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>$P_1$</td>
<td>$B_1$</td>
<td>$g_1 = c(B_1, \theta_0)$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$P_2$</td>
<td>$B_2$</td>
<td>$g_2 = c(B_2, \theta_0)$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$P_3$</td>
<td>$B_3$</td>
<td>$g_3 = c(B_3, \theta_0)$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$B_4$</td>
<td>$\theta_1 = u(\theta_0, g_1)$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$g_1 = c(B_4, \theta_1)$</td>
</tr>
</tbody>
</table>
Parallel Asynchronous Mini-Batch Learning
Nedic, Bertsekas, and Borkar (2001)

<table>
<thead>
<tr>
<th>$\theta$</th>
<th>$\theta_0$</th>
<th>$\theta_1$</th>
<th>$\theta_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{P}_1$</td>
<td>$g_1 = c(B_1, \theta_0)$</td>
<td>$\theta_1 = u(\theta_0, g_1)$</td>
<td>$g_1 = c(B_4, \theta_1)$</td>
</tr>
<tr>
<td>$\mathcal{P}_2$</td>
<td>$g_2 = c(B_2, \theta_0)$</td>
<td></td>
<td>$\theta_2 = u(\theta_1, g_2)$</td>
</tr>
<tr>
<td>$\mathcal{P}_3$</td>
<td>$g_3 = c(B_3, \theta_0)$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Parameters: $\theta_t$
Processors: $\mathcal{P}_i$
Mini-batches: $B_j$
Gradient: $g_k$
Parallel Asynchronous Mini-Batch Learning
Nedic, Bertsekas, and Borkar (2001)

<table>
<thead>
<tr>
<th>Parameters: $\theta_t$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processors: $P_i$</td>
</tr>
<tr>
<td>Mini-batches: $B_j$</td>
</tr>
<tr>
<td>Gradient: $g_k$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\theta$</th>
<th>$\theta_0$</th>
<th>$\theta_1$</th>
<th>$\theta_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>$g_1 = c(B_1, \theta_0)$</td>
<td>$\theta_1 = u(\theta_0, g_1)$</td>
<td>$g_1 = c(B_4, \theta_1)$</td>
</tr>
<tr>
<td>$P_2$</td>
<td>$g_2 = c(B_2, \theta_0)$</td>
<td>$\theta_2 = u(\theta_1, g_2)$</td>
<td>$g_2 = c(B_5, \theta_2)$</td>
</tr>
<tr>
<td>$P_3$</td>
<td>$g_3 = c(B_3, \theta_0)$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Time
## Parallel Asynchronous Mini-Batch Learning

Nedic, Bertsekas, and Borkar (2001)

<table>
<thead>
<tr>
<th>θ</th>
<th>θ₀</th>
<th>θ₁</th>
<th>θ₂</th>
<th>θ₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>ℙ₁</td>
<td>( g₁ = c(B₁, θ₀) )</td>
<td>( θ₁ = u(θ₀, g₁) )</td>
<td>( g₁ = c(B₄, θ₁) )</td>
<td></td>
</tr>
<tr>
<td>ℙ₂</td>
<td>( g₂ = c(B₂, θ₀) )</td>
<td>( θ₂ = u(θ₁, g₂) )</td>
<td>( g₂ = c(B₅, θ₂) )</td>
<td></td>
</tr>
<tr>
<td>ℙ₃</td>
<td>( g₃ = c(B₃, θ₀) )</td>
<td>( θ₃ = u(θ₂, g₃) )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Parameters:** \( θ_t \)

**Processors:** \( ℙ_i \)

**Mini-batches:** \( B_j \)

**Gradient:** \( g_k \)
Parallel Asynchronous Mini-Batch Learning  
Nedic, Bertsekas, and Borkar (2001)

<table>
<thead>
<tr>
<th>$\theta$</th>
<th>$\theta_0$</th>
<th>$\theta_1$</th>
<th>$\theta_2$</th>
<th>$\theta_3$</th>
<th>$\theta_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{P}_1$</td>
<td>$g_1 = c(\mathcal{B}_1, \theta_0)$</td>
<td>$\theta_1 = u(\theta_0, g_1)$</td>
<td>$g_1 = c(\mathcal{B}_4, \theta_1)$</td>
<td>$\theta_4 = u(\theta_3, g_1)$</td>
<td>$g_1 = c(\mathcal{B}_6, \theta_4)$</td>
</tr>
<tr>
<td>$\mathcal{P}_2$</td>
<td>$g_2 = c(\mathcal{B}_2, \theta_0)$</td>
<td>$\theta_2 = u(\theta_1, g_2)$</td>
<td>$g_2 = c(\mathcal{B}_5, \theta_2)$</td>
<td>$\theta_5 = u(\theta_4)$</td>
<td>$g_2 = c(\mathcal{B}_7, \theta_5)$</td>
</tr>
<tr>
<td>$\mathcal{P}_3$</td>
<td>$g_3 = c(\mathcal{B}_3, \theta_0)$</td>
<td>$\theta_3 = u(\theta_2, g_3)$</td>
<td>$g_3 = c(\mathcal{B}_6, \theta_3)$</td>
<td>$\theta_6 = u(\theta_5)$</td>
<td>$g_3 = c(\mathcal{B}_8, \theta_6)$</td>
</tr>
</tbody>
</table>

- Gradients computed using stale parameters
- Increased processor utilization
- Only idle time caused by lock for updating parameters

Parameters: $\theta_t$  
Processors: $\mathcal{P}_i$  
Mini-batches: $\mathcal{B}_j$  
Gradient: $g_k$
Theoretical Results

- How does the use of stale parameters affect convergence?

- Convergence results exist for convex optimization using stochastic gradient descent
  - Convergence guaranteed when max delay is bounded (Nedic, Bertsekas, and Borkar, 2001)
  - Convergence rates linear in max delay (Langford, Smola, and Zinkevich, 2009)
# Experiments

| Task                              | Model      | Method                     | Convex? | $|\mathcal{D}|$ | $|\theta|$ | $m$ |
|-----------------------------------|------------|----------------------------|---------|----------|-----------|-----|
| Named-Entity Recognition          | CRF        | Stochastic Gradient Descent| Y       | 15k      | 1.3M      | 4   |
| Word Alignment                    | IBM Model 1| Stepwise EM                | Y       | 300k     | 14.2M     | 10k |
| Unsupervised Part-of-Speech Tagging| HMM        | Stepwise EM                | N       | 42k      | 2M        | 4   |

- To compare algorithms, we use wall clock time (with a dedicated 4-processor machine)
- $m$ = mini-batch size
Experiments

| Task                   | Model | Method                  | Convex? | $|D|$ | $|\theta|$ | $m$ |
|------------------------|-------|-------------------------|---------|-----|----------|-----|
| Named-Entity Recognition | CRF   | Stochastic Gradient Descent | Y       | 15k | 1.3M     | 4   |

- CoNLL 2003 English data
- Label each token with entity type (person, location, organization, or miscellaneous) or non-entity
- We show convergence in F1 on development data
Asynchronous Updating Speeds Convergence

All use a mini-batch size of 4
Comparison with Ideal Speed-up

Asynchronous (4 processors)

Ideal

Wall clock time (hours)
Why Does Asynchronous Converge Faster?

- Processors are kept in near-constant use
- Synchronous SGD leads to idle processors → need for load-balancing
Clearer improvement for asynchronous algorithms when increasing number of processors
Artificial Delays

After completing a mini-batch, 25% chance of delaying

Delay (in seconds) sampled from
\[ \max(\mathcal{N}(\mu, (\mu/5)^2), 0) \]

Avg. time per mini-batch = 0.62 s
Experiments

| Task                     | Model   | Method     | Convex? | $|\mathcal{D}|$ | $|\theta|$ | $m$ |
|-------------------------|---------|------------|---------|---------|----------|------|
| Word Alignment          | IBM Model 1 | Stepwise EM | Y       | 300k    | 14.2M    | 10k  |

- Given parallel sentences, draw links between words:

  konnten sie es übersetzen ?

  could you translate it ?

- We show convergence in log-likelihood (convergence in AER is similar)
Stepwise EM  
(Sato and Ishii, 2000; Cappe and Moulines, 2009)

- Similar to stochastic gradient descent in the space of sufficient statistics, with a particular scaling of the update
- More efficient than incremental EM  
  (Neal and Hinton, 1998)
- Found to converge much faster than batch EM  
  (Liang and Klein, 2009)
Word Alignment Results

For stepwise EM, mini-batch size = 10,000

- Asynch. Stepwise EM (4 processors)
- Synch. Stepwise EM (4 processors)
- Synch. Stepwise EM (1 processor)
- Batch EM (1 processor)
Word Alignment Results

For stepwise EM, mini-batch size = 10,000

Asynchronous is no faster than synchronous!
Word Alignment Results

Asynchronous is no faster than synchronous!

For stepwise EM, mini-batch size = 10,000
Comparing Mini-Batch Sizes

Wall clock time (minutes)

Log-Likelihood

Asynch. (m = 10,000)
Synch. (m = 10,000)
Asynch. (m = 1,000)
Synch. (m = 1,000)
Asynch. (m = 100)
Synch. (m = 100)
Comparing Mini-Batch Sizes

Asynchronous is faster when using small mini-batches
Comparing Mini-Batch Sizes

Wall clock time (minutes)

Log-Likelihood

Asynch. (m = 10,000)
Synch. (m = 10,000)
Asynch. (m = 1,000)
Synch. (m = 1,000)
Asynch. (m = 100)
Synch. (m = 100)

Error from asynchronous updating
Word Alignment Results

For stepwise EM, mini-batch size = 10,000
Comparison with Ideal Speed-up

For stepwise EM, mini-batch size = 10,000
MapReduce?

- We also ran these algorithms on a large MapReduce cluster (M45 from Yahoo!)

- Batch EM
  - Each iteration is one MapReduce job, using 24 mappers and 1 reducer

- Asynchronous Stepwise EM
  - 4 mini-batches processed simultaneously, each run as a MapReduce job
  - Each uses 6 mappers and 1 reducer
MapReduce?

-40 -35 -30 -25 -20 -15 -10 -5 0 5 10 15 20 25 30 35 40

Log-Likelihood

Asynch. Stepwise EM (4 processors)
Synch. Stepwise EM (4 processors)
Synch. Stepwise EM (1 processor)
Batch EM (1 processor)
MapReduce?

Asynch. Stepwise EM (4 processors)
-20
-25
-30
-35
-40
Synch. Stepwise EM (4 processors)
Synch. Stepwise EM (1 processor)
Batch EM (1 processor)
Log-Likelihood

Asynch. Stepwise EM (MapReduce)
Batch EM (MapReduce)

Wall clock time (minutes)
## Experiments

| Task                  | Model  | Method    | Convex? | $|D|$ | $|\theta|$ | $m$ |
|-----------------------|--------|-----------|---------|-----|------------|-----|
| Unsupervised Part-of-Speech Tagging | HMM    | Stepwise EM | N       | 42k | 2M         | 4   |

- Bigram HMM with 45 states

- We plot convergence in likelihood and many-to-1 accuracy
Part-of-Speech Tagging Results

mini-batch size = 4 for stepwise EM

Asynch. Stepwise EM (4 processors)
Synch. Stepwise EM (4 processors)
Synch. Stepwise EM (1 processor)
Batch EM (1 processor)
Comparison with Ideal

Log-Likelihood

Wall clock time (hours)

Accuracy (%)

Asynch. Stepwise EM (4 processors)
Ideal

ARK
lti
Carnegie Mellon
Comparison with Ideal

Asynchronous better than ideal?
Conclusions and Future Work

- Asynchronous algorithms speed convergence and do not introduce additional error.
- Effective for unsupervised learning and non-convex objectives.
- If your problem works well with small mini-batches, try this!

Future work
- Theoretical results for non-convex case
- Explore effects of increasing number of processors
- New architectures (maintain multiple copies of $\theta$)
Thanks!