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

\[ \theta \]

\[ \mathcal{P}_1 \]

Parameters: $\theta_t$

Processors: $\mathcal{P}_i$

Dataset: $\mathcal{D}$
Single-Processor Batch Learning

| $\theta$ | $\theta_0$ |
|----------|------------|
| $P_1$    | $g = \text{calc}(D, \theta_0)$ |

$g = \text{calc}(D, \theta)$:
Calculate gradient $g$ on data $D$ using parameters $\theta$

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

\[
\begin{array}{|c|c|c|}
\hline
\theta & \theta_0 & \theta_1 \\
\hline
\mathcal{P}_1 & g = \text{calc}(\mathcal{D}, \theta_0) & \theta_1 = \text{up}(\theta_0, g) \\
\hline
\end{array}
\]

- \( g = \text{calc}(\mathcal{D}, \theta) \): 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

| $\theta$ | $\theta_0$ | $\theta_1$ |
|----------|-------------|-------------|
| $\mathcal{P}_1$ | $g = \text{calc}(\mathcal{D}, \theta_0)$ | $\theta_1 = \text{up}(\theta_0, g)$, $g = \text{calc}(\mathcal{D}, \theta_1)$ |

**g = 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: $\mathcal{D}$
## Parallel Batch Learning

| $\theta$ | $\theta_0$ |
|---|---|
| $\mathcal{P}_1$ | $\mathbf{g}_1 = \text{calc}(\mathcal{D}_1, \theta_0)$ |
| $\mathcal{P}_2$ | $\mathbf{g}_2 = \text{calc}(\mathcal{D}_2, \theta_0)$ |
| $\mathcal{P}_3$ | $\mathbf{g}_3 = \text{calc}(\mathcal{D}_3, \theta_0)$ |

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

Parameters: $\theta_t$

Processors: $\mathcal{P}_i$

Dataset: $\mathcal{D} = \mathcal{D}_1 \cup \mathcal{D}_2 \cup \mathcal{D}_3$

Gradient: $\mathbf{g} = \mathbf{g}_1 + \mathbf{g}_2 + \mathbf{g}_3$
### Parallel Batch Learning

| θ | θ₀ | θ₁ |
|---|---|---|
| $\mathcal{P}_1$ | $g_1 = \text{calc}(\mathcal{D}_1, \theta_0)$ | $\theta_1 = \text{up}(\theta_0, g)$ |
| $\mathcal{P}_2$ | $g_2 = \text{calc}(\mathcal{D}_2, \theta_0)$ |
| $\mathcal{P}_3$ | $g_3 = \text{calc}(\mathcal{D}_3, \theta_0)$ |

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

Parameters: $\theta_t$
Processors: $\mathcal{P}_i$
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

| \( \theta \) | \( \theta_0 \) | \( \theta_1 \) |
|---|---|---|
| \( P_1 \) | \( g_1 = \text{calc}(D_1, \theta_0) \) | \( \theta_1 = \text{up}(\theta_0, g) \) \( g_1 = \text{calc}(D_1, \theta_1) \) \( \theta_2 = \text{up}(\theta_1, g) \) |
| \( P_2 \) | \( g_2 = \text{calc}(D_2, \theta_0) \) | \( g_2 = \text{calc}(D_2, \theta_1) \) |
| \( P_3 \) | \( g_3 = \text{calc}(D_3, \theta_0) \) | \( g_3 = \text{calc}(D_3, \theta_1) \) |

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

Parameters: \( \theta_t \)
Processors: \( P_i \)
Dataset: \( D = D_1 \cup D_2 \cup D_3 \)
Gradient: \( g = g_1 + g_2 + g_3 \)
**Parallel Synchronous Mini-Batch Learning**

Finkel, Kleeman, and Manning (2008)

| $\theta$ | $\theta_0$ | $\theta_1$ | $\theta_2$ |
|----------|-------------|-------------|-------------|
| $\mathcal{P}_1$ | $g_1 = c(\mathcal{B}_1^1, \theta_0)$ | $\theta_1 = u(\theta_0, g)$ | $g_1 = c(\mathcal{B}_2^1, \theta_1)$ | $\theta_2 = u(\theta_1, g)$ |
| $\mathcal{P}_2$ | $g_2 = c(\mathcal{B}_1^2, \theta_0)$ | $g_2 = c(\mathcal{B}_2^2, \theta_1)$ | $g_2 = c(\mathcal{B}_3^2, \theta_1)$ |
| $\mathcal{P}_3$ | $g_3 = c(\mathcal{B}_1^3, \theta_0)$ | $g_3 = c(\mathcal{B}_2^3, \theta_1)$ | $g_3 = c(\mathcal{B}_3^3, \theta_1)$ |

- Same architecture, just more frequent updates

Parameters: $\theta_t$

Processors: $\mathcal{P}_i$

Mini-batches: $\mathcal{B}_t = \mathcal{B}_1^1 \cup \mathcal{B}_2^2 \cup \mathcal{B}_3^3$

Gradient: $g = g_1 + g_2 + g_3$
Parallel Asynchronous Mini-Batch Learning
Nedic, Bertsekas, and Borkar (2001)

| θ | θ₀ |
|---|---|
| \( \mathcal{P}_1 \) |   |
| \( \mathcal{P}_2 \) |   |
| \( \mathcal{P}_3 \) |   |

0 \hspace{5cm} \text{Time}

Parameters: \( \theta_t \)
Processors: \( \mathcal{P}_i \)
Mini-batches: \( \mathcal{B}_j \)
Gradient: \( \mathbf{g}_k \)
### Parallel Asynchronous Mini-Batch Learning

*Nedic, Bertsekas, and Borkar (2001)*

| \( \theta \) | \( \theta_0 \) |
|---|---|
| \( \mathcal{P}_1 \) | \( g_1 = c(\mathcal{B}_1, \theta_0) \) |
| \( \mathcal{P}_2 \) | \( g_2 = c(\mathcal{B}_2, \theta_0) \) |
| \( \mathcal{P}_3 \) | \( g_3 = c(\mathcal{B}_3, \theta_0) \) |

- **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)

| $\theta$ | $\theta_0$ | $\theta_1$ |
|----------|-------------|------------|
| $P_1$    | $g_1 = c(B_1, \theta_0)$ | $\theta_1 = u(\theta_0, g_1)$ |
| $P_2$    | $g_2 = c(B_2, \theta_0)$ |
| $P_3$    | $g_3 = c(B_3, \theta_0)$ |

Parameters: $\theta_t$
Processors: $P_i$
Mini-batches: $B_j$
Gradient: $g_k$
**Parallel Asynchronous Mini-Batch Learning**

Nedic, Bertsekas, and Borkar (2001)

| $\theta$ | $\theta_0$ | $\theta_1$ |
|----------|-------------|-------------|
| $P_1$    | $g_1 = c(\mathcal{B}_1, \theta_0)$ | $\theta_1^* = u(\theta_0, g_1)$ $g_1 = c(\mathcal{B}_4, \theta_1)$ |
| $P_2$    | $g_2 = c(\mathcal{B}_2, \theta_0)$ |
| $P_3$    | $g_3 = c(\mathcal{B}_3, \theta_0)$ |

**Parameters:** $\theta_t$

**Processors:** $P_i$

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

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

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

### Table

| $\theta$ | $\theta_0$ | $\theta_1$ | $\theta_2$ |
|----------|-------------|-------------|-------------|
| $P_1$    | $g_1 = c(\mathcal{B}_1, \theta_0)$ | $\theta_1 = u(\theta_0, g_1)$ | $g_1 = c(\mathcal{B}_4, \theta_1)$ |
| $P_2$    | $g_2 = c(\mathcal{B}_2, \theta_0)$ |             | $\theta_2 = u(\theta_1, g_2)$ |
| $P_3$    | $g_3 = c(\mathcal{B}_3, \theta_0)$ |             |             |

### Diagram

- Parameters: $\theta_t$
- Processors: $P_i$
- Mini-batches: $\mathcal{B}_j$
- Gradient: $g_k$
### Parallel Asynchronous Mini-Batch Learning

Nedic, Bertsekas, and Borkar (2001)

|   | \( \theta_0 \) | \( \theta_1 \) | \( \theta_2 \) |
|---|-----------------|-----------------|-----------------|
| \( \mathcal{P}_1 \) | \( g_1 = c(B_1, \theta_0) \) | \( \theta_1 = u(\theta_0, g_1) \) | \( g_1 = c(B_4, \theta_1) \) |
| \( \mathcal{P}_2 \) | \( g_2 = c(B_2, \theta_0) \) | \( \theta_2 = u(\theta_1, g_2) \) | \( g_2 = c(B_5, \theta_2) \) |
| \( \mathcal{P}_3 \) | \( g_3 = c(B_3, \theta_0) \) |                       |                 |

Parameters: \( \theta_t \)  
Processors: \( \mathcal{P}_i \)  
Mini-batches: \( B_j \)  
Gradient: \( g_k \)  

Time

---

*ARK* 
*i* 
*Carnegie Mellon*
Parallel Asynchronous Mini-Batch Learning
Nedic, Bertsekas, and Borkar (2001)

| \( \theta \) | \( \theta_0 \) | \( \theta_1 \) | \( \theta_2 \) | \( \theta_3 \) |
|-----|-----|-----|-----|-----|
| \( \mathcal{P}_1 \) | \( g_1 = c(B_1, \theta_0) \) | \( \theta_1 = u(\theta_0, g_1) \) | \( g_1 = c(B_4, \theta_1) \) |
| \( \mathcal{P}_2 \) | \( g_2 = c(B_2, \theta_0) \) | \( \theta_2 = u(\theta_1, g_2) \) | \( g_2 = c(B_5, \theta_2) \) |
| \( \mathcal{P}_3 \) | \( g_3 = c(B_3, \theta_0) \) | \( \theta_3 = u(\theta_2, g_3) \) |

Parameters: \( \theta_t \)
Processors: \( \mathcal{P}_i \)
Mini-batches: \( B_j \)
Gradient: \( g_k \)
### Parallel Asynchronous Mini-Batch Learning

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

| \( \theta \) | \( \theta_0 \) | \( \theta_1 \) | \( \theta_2 \) | \( \theta_3 \) | \( \theta_4 \) |
|---|---|---|---|---|---|
| \( \mathcal{P}_1 \) | \( g_1 = c(\mathcal{B}_1, \theta_0) \) | \( \theta_1 = u(\theta_0, g_1) \) | \( g_1 = c(\mathcal{B}_4, \theta_1) \) | | | \( g_1 = c(\mathcal{B}_1, \theta_0) \) |
| \( \mathcal{P}_2 \) | \( g_2 = c(\mathcal{B}_2, \theta_0) \) | | \( g_2 = c(\mathcal{B}_5, \theta_2) \) | | \( \theta_5 = u(\theta_4) \) |
| \( \mathcal{P}_3 \) | \( g_3 = c(\mathcal{B}_3, \theta_0) \) | | | \( \theta_3 = u(\theta_2, g_3) \) | \( g_3 = c(\mathcal{B}_6, \theta_3) \) |

**Key Points:**

- 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? | $|D|$   | $|\theta|$ | $m$ |
|------------------------------------------------|-----------|--------------------|---------|-------|------------|------|
| 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? | $|\mathcal{D}|$ | $|\theta|$ | $m$ |
|--------------------------|-------|-------------------------|---------|-----------|-----------|------|
| 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

Wall clock time (hours)

F1

Asynchronous (4 processors)
Ideal

Ark
lti
Carnegie Mellon
Why Does Asynchronous Converge Faster?

- Processors are kept in near-constant use
- Synchronous SGD leads to idle processors \(\rightarrow\) 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
Word Alignment Results

Asynchronous is no faster than synchronous!

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

![Graph showing the comparison of different EM algorithms under MapReduce and batch processing.](image-url)
## 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
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!