Asynchronous Parallel Learning for Neural Networks and Structured Models with Dense Features

Xu SUN (孙栩)
Peking University
xusun@pku.edu.cn
Neural networks -> Good Performance

- CNN, RNN, LSTM...
- Sequence labelling, parsing, machine translation...

Neural networks -> Slow Training

- Large parameter space
- Dense feature
- Complex computation

Faster Training? -> Parallel Training

- Synchronous
- Asynchronous -> AsynGrad
Many kinds

- Feed Forward Neural Networks
  - logistic regression

- Convolutional Neural Networks
  - image processing

- Recurrent Neural Networks
  - RNN, LSTM, GRU...
  - structured prediction
Recurrent Neural Network (RNN)

- **Recurrent neural network** (Elman, Cognitive Science 1990)
  - Model time series
  - Predict linear-chain structures
  - Conditioned on all previous input

\[
h_t = f(U h_{t-1} + W x_t)
\]

\[
\hat{y}_t = \text{softmax}(W^{(s)} h_t)
\]
Long Short-term Memory (LSTM)

- Long short-term memory (Hochreiter and Schmidhuber 1997)
  - A lasting linear memory
  - Capture long distance dependency
  - Three gates: input, forget and output gates
    - Control modification to the memory

![LSTM Diagram]

\[ S^{(t-1)} \rightarrow x^{(t)} \rightarrow h^{(t)} \]

\[ S^{(t)} \rightarrow \text{tanh} \rightarrow \text{sig} \rightarrow h^{(t)} \]

\[ f^{(t)} \rightarrow i^{(t)} \rightarrow o^{(t)} \]
Sequence-to-Sequence Model

- **Sequence to sequence neural network** (Sutskever et al., NIPS 2014)
  - Encoder & Decoder
  - The encoder information is stored in a fixed-length vector

- **Popular for high-level task**
  - Machine Translation
  - Summarization
  - ...

![Diagram of Sequence-to-Sequence Model](image-url)
Training large-scale neural networks is costly

- Numerous parameters
- Dense Feature
- Time-consuming

For example

- A NMT model may take weeks to train
- Days, even if with GPU clusters

How to accelerate training speed?

- Parallel training
- Especially, asynchronous (lock-free) parallel training
Basic operations in parallel training

Problem differs in

- Online vs. Mini-batch vs. Batch
- Synchronous parallel vs. Asynchronous parallel
- Dense feature model vs. Sparse feature model
Parallel Training

- **Synchronous (locked)**
  - Multiple threads
  - Only one can modify model parameters at the same time

- **Asynchronous (lock-free)**
  - Multiple threads
  - Each one can modify model parameters at the same time
### Model Types

- **Sparse feature model**
  - e.g. HMM, CRF, Perceptron, MILA...
  - features are sparse
  - less read & write time

- **Dense feature model**
  - e.g. RNN, LSTM, Sequence-to-Sequence...
  - features are dense
  - more read & write time

#### Feature Space

<table>
<thead>
<tr>
<th>Sparse feature model</th>
<th>Dense feature model</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 1 0 0 ...</td>
<td>0 1 0.1 0.5 1.0 0.3 0.7 ...</td>
</tr>
<tr>
<td></td>
<td>0.2 1.0</td>
</tr>
</tbody>
</table>
## Problem Analysis

- **How threads interact with each other?**

<table>
<thead>
<tr>
<th></th>
<th>Sparse</th>
<th>Dense</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sync.</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Async.</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>
### Problem Analysis

- **How threads interact with each other?**

<table>
<thead>
<tr>
<th></th>
<th>Sparse</th>
<th>Dense</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sync.</strong></td>
<td><img src="?" alt="Sync Sparse" /></td>
<td><img src="?" alt="Sync Dense" /></td>
</tr>
<tr>
<td><strong>Async.</strong></td>
<td><img src="?" alt="Async Sparse" /></td>
<td><img src="?" alt="Async Dense" /></td>
</tr>
</tbody>
</table>

Note: Questions marks indicate areas where more information is needed.
### Correctness

1. **Simple case**
   - Reading parameters from shared memory
   - Computing Gradients
   - Writing parameters to shared memory

### Current approach: DSGD (round-robin)

- Langford et al, NIPS 2009
- Stochastic parallel learning by locking memory
Synchronous Online Parallel Training

- **Correctness**
  - No problem at all!
  - **1. Simple case**
    - Reading parameters from shared memory
    - Computing **G**radients
    - Writing parameters to shared memory

(a) Simple case

- **Current approach:**
  - mini-batch based method
  - Computing gradients in parallel
    - such as: parallel matrix operations via GPU
## Problem Analysis

- How threads interact with each other?

<table>
<thead>
<tr>
<th></th>
<th>Sparse</th>
<th>Dense</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sync.</td>
<td><img src="#" alt="Checkmark" /></td>
<td><img src="#" alt="Checkmark" /></td>
</tr>
<tr>
<td>Async.</td>
<td><img src="#" alt="Question Mark" /></td>
<td><img src="#" alt="Question Mark" /></td>
</tr>
</tbody>
</table>
### Problem Analysis

- How threads interact with each other?

<table>
<thead>
<tr>
<th></th>
<th>Sparse</th>
<th>Dense</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sync.</strong></td>
<td><img src="https://via.placeholder.com/15" alt="✓" /></td>
<td><img src="https://via.placeholder.com/15" alt="✓" /></td>
</tr>
</tbody>
</table>
## Problem Analysis

- How threads interact with each other?

<table>
<thead>
<tr>
<th></th>
<th>Sparse</th>
<th>Dense</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sync.</strong></td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td><strong>Async.</strong></td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>
Asynchronous parallel learning is very popular for traditional **sparse** feature models.

2. This case is called Gradient Delay case.

→ More complicated, but problem solved for sparse feature models (Niu et al. NIPS 2011)
Current approach: **HogWild! and variants**

- Multiple threads updating parameters at the same time
- For sparse feature models

**Advantage**

- Actual parallel training
- Fast training speed with **no performance loss**

Problem also solved (for sparse feature models)!
## Problem Analysis

- **How threads interact with each other?**

<table>
<thead>
<tr>
<th></th>
<th>Sparse</th>
<th>Dense</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sync.</strong></td>
<td>![Checkmark]</td>
<td>![Checkmark]</td>
</tr>
<tr>
<td><strong>Async.</strong></td>
<td>![Checkmark]</td>
<td>?</td>
</tr>
</tbody>
</table>

- Sparse and dense synchronization are indicated, while asynchronous interaction is unclear.

**Note:** The question mark indicates an unknown or not clearly defined interaction in asynchronous scenarios.
How threads interact with each other?

<table>
<thead>
<tr>
<th></th>
<th>Sparse</th>
<th>Dense</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sync.</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Async.</td>
<td>✓</td>
<td>?</td>
</tr>
</tbody>
</table>
How threads interact with each other?

<table>
<thead>
<tr>
<th></th>
<th>Sparse</th>
<th>Dense</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sync.</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Async.</td>
<td>✔️</td>
<td>✔️</td>
</tr>
</tbody>
</table>
3. Even more difficult case: **Gradient Error Case**

- Happens for dense feature models, like neural networks
  - Actions (R, G & W) are time-consuming
- Read-overwrite and write-overwrite problems

→ Not well studied before, how to deal with this problem?

![Diagram showing simple case, grad delay case, and grad error case](image)
## Problem Analysis

- **How threads interact with each other?**

<table>
<thead>
<tr>
<th></th>
<th>Sparse</th>
<th>Dense</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sync.</strong></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Async.</strong></td>
<td>✓</td>
<td>?</td>
</tr>
</tbody>
</table>
### Problem Analysis

- **How threads interact with each other?**

<table>
<thead>
<tr>
<th></th>
<th>Sparse</th>
<th>Dense</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sync.</strong></td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td><strong>Async.</strong></td>
<td>✔️</td>
<td>✔️</td>
</tr>
</tbody>
</table>

AsynGrad aims to solve gradient error case.
### Problem Analysis

- **How threads interact with each other?**

<table>
<thead>
<tr>
<th></th>
<th>Sparse</th>
<th>Dense</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sync.</strong></td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td><strong>Async.</strong></td>
<td>✔️</td>
<td>AsynGrad ✔️</td>
</tr>
</tbody>
</table>

This is our proposal
Gradient error has two aspects

- How many of the gradients are wrong?
- How wrong are they?

<table>
<thead>
<tr>
<th>Gradient vectors</th>
<th>all correct</th>
<th>some are wrong</th>
<th>most are wrong</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>correct</td>
<td>slightly wrong</td>
<td>very wrong</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Gradient error is very common in asynchronous training of neural networks in real-world tasks.
Gradient error is **moderate** in asynchronous training of neural networks in real-world tasks.

**Experimental Observations**

- **Gradient error**
  - naïve case
  - practical case
  - doomed case
  - gradient vectors
    - correct
    - slightly wrong
    - very wrong

(a) Simple case

(b) Grad delay case

(c) Grad error case
Can training still converge with gradient errors?

Even though there are gradient errors, training still converges towards the optimum, when the gradient errors are bounded.

Theorem 1 (AsynGrad convergence and convergence rate). With the conditions (4), (5), (6), (7), let $\epsilon > 0$ be a target 0-approximation. Let $\tau$ denote the learning rate and $\epsilon_z$ the error on the $z$-th target $z$ such that $s(w) = \mathbb{E}_z[s_z(w)]$. Let $\gamma$ be a learning rate as

$$\gamma = \frac{c\epsilon - 2\tau q}{\beta q \kappa^2}$$

where we can set $\beta$ as any value as far as $\beta \geq 1$. Let $t$ be the number of updates as follows

$$t = \frac{\beta q \kappa^2 \log \left( qa_0 / \epsilon \right)}{c(c\epsilon - 2\tau q)}$$

where $\lceil \cdot \rceil$ means ceil-rounding of a real value to an integer, and $a_0$ is the initial distance such that $a_0 = \|w_0 - w^*\|^2$. Then, after $t$ updates of $w$, AsynGrad converges towards the optimum such that $\mathbb{E}[f(w_t) - f(w^*)] \leq \epsilon$, as far as the gradient errors are bounded such that

$$\tau \leq \frac{c\epsilon}{2q}$$
Our Theoretical Analysis

- Even if most of the gradients are wrong

- With their errors bounded, training still converge
An asynchronous parallel learning solution for fast training of neural networks

[Algorithm 1: AsynGrad]

<table>
<thead>
<tr>
<th>Algorithm 1</th>
<th>AsynGrad: Asynchronous Parallel Learning with Gradient Error</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong></td>
<td>model weights $w$, training set $S$ of $m$ samples</td>
</tr>
<tr>
<td></td>
<td>Run $k$ threads in parallel with share memory, and procedure of each thread is as follows:</td>
</tr>
<tr>
<td></td>
<td>repeat</td>
</tr>
<tr>
<td></td>
<td>Get a sample $z$ uniformly at random from $S$</td>
</tr>
<tr>
<td></td>
<td>Get the <strong>update term</strong> $s_z(w)$, which is computed as $\nabla f_z(w)$ but usually contains error</td>
</tr>
<tr>
<td></td>
<td>Update $w$ such that $w \leftarrow w - \gamma s_z(w)$</td>
</tr>
<tr>
<td></td>
<td>until Convergence</td>
</tr>
<tr>
<td></td>
<td>return $w$</td>
</tr>
</tbody>
</table>

Experiments on LSTM

Experiments show a high gradient error rate
Experiments on LSTM

Experiments show that AsynGrad still converge even with a high gradient error rate.
Experiments on LSTM

- No loss on accuracy/F-score
- With substantially faster training speed

AsynGrad

Also suitable for other dense feature models

- Dense CRF -> moderate error rate
Experiments on Dense-CRF

- No loss on accuracy/F-score
- With substantially faster training speed

AsynGrad
How threads interact with each other?

<table>
<thead>
<tr>
<th></th>
<th>Sparse</th>
<th>Dense</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sync.</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Async.</td>
<td>✔️</td>
<td>❌</td>
</tr>
</tbody>
</table>
Gradient errors are common and inevitable in asynchronous training of dense feature models

- Such as neural networks

AsynGrad survives with gradient errors

- With substantial faster training speed
- No loss at all on test accuracy
- Theoretical justification
Thanks!