A parallel version of deep learning
content

A brief summary of deep learning and google research’s work

A approximation parallel algorithm of deep learning

Experiment result and future work
content

A brief summary of deep learning and google research’s work

A approximation parallel algorithm of deep learning

Experiment result and future work
What is deep learning

Deep learning is a class of machine learning algorithms that:

- use a cascade of many layers of nonlinear processing units
- for feature extraction and transformation.

Deep learning is very popular

Deep learning is very slow
A easy sample of deep learning

A Neural Network

Can be viewed as a generalization of linear models

\[ y_k(x, w) = \sigma \left( \sum_{j=1}^{M} w^{(2)}_{kj} h \left( \sum_{i=1}^{D} w^{(1)}_{ji} x_i + w^{(1)}_{j0} \right) + w^{(2)}_{k0} \right) \]
A instruction for google research’s work

Draw a TensorFlow graph
Find the dependency
Add node to change dependency
Controlling Data Communication and Memory Usage
Figure 12: EEG visualization of multi-threaded CPU operations (x-axis is time in µs).
content

A brief summary of deep learning and Google Research’s work

A approximation parallel algorithm of deep learning

Experiment result and future work
What I want to do at first

Use gradient decent to update the parameter
Try to update a parameter more efficiently
--------by update parameter at same time
--------by sorting the data in a special way
Why Hard to implement

There are too much dependency!

You can not training one layer’s weight without other layers

\[ w^{(\tau+1)} = w^{(\tau)} - \eta \nabla E(w^{(\tau)}) \]

\[ E(w) = -\sum_{n=1}^{N} \sum_{k=1}^{K} \{t_{nk}\ln y_{nk} + (1-t_{nk})\ln(1-y_{nk}) \} \]

where \( y_{nk} \) denotes \( y_k(x_n, w) \)
Solution: A approximation algorithm

Try to update the weight later in each layer

Reduce the relevance between different layers

Let each layer to update it’s weight by itself

After training for a period communicate with other layer and update the global weight
approximation algorithm

Use some data to train the weight in a traditional way

Layer 1  Layer 2  Layer 3

Update the weight in N communication interval

......

Result
content

A brief summary of deep learning and google research’s work

A approximation parallel algorithm of deep learning

Experiment result and future work
Experiment-------- MNIST

The MNIST database is a large database of handwritten digits that is commonly used for training various image processing systems.

The MNIST database contains 60,000 training images and 10,000 testing images.
Result of different communication interval

<table>
<thead>
<tr>
<th>comm interval</th>
<th>1000</th>
<th>100</th>
<th>10</th>
<th>5</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>accuracy</td>
<td>55.39</td>
<td>55.61</td>
<td>74.35</td>
<td>83.83</td>
<td>88.68</td>
</tr>
</tbody>
</table>

![Graph showing the accuracy of communication intervals from 1000 to 1.](image)
Speed up

Former time: data size \times \text{training time per piece of data}

New algorithm’s time: \frac{\text{Former time}}{\text{the layer of the model}} + \frac{\text{data size}}{\text{comm interval}} \times \text{communication time}
Conclusion

Approximation algorithm cannot do a good job when communication interval is really big.

When communication interval is small, the algorithm could be slow.

When there is only one or two layers, the speed up is not very obvious, but when it comes to more layers, speed up will be much more obviously.
Future work

Try to implement on more layers

Try to use more core to implement it
Reference


Jeffrey Dean, Greg S. Corrado, etc. Large Scale Distributed Deep Networks. NIPS, 2012

Thank!