Distributed Convolutional Neural Network with Apache Spark

Yuwei Jiao, Vivi Ma
Outline

• Background
• Fundamental CNN Workflow
• Challenges
• Implementation with Apache Spark
• Implementation with TensorFlow
• Future Work
Background
Fundamental CNN Workflow
Setting

Language: Python 3
Dataset: CIFAR10
Convolution: 32 filters(5x5x3), stride 1, zero-padding 2;
Pooling: 2x2 filter size, stride 2, max pooling;
Fully Connected: 16x16x32 => 10 classifications;
Challenges

- Time Complexity
- Space Complexity
Profiling of Naive Implementation

![Graphs showing space and time vs. image size.](image-url)
## Profiling of Naive Implementation

### 2000 Images per Iteration

<table>
<thead>
<tr>
<th>Layer</th>
<th>Forward Time</th>
<th>Backward Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conv</td>
<td>4.252s</td>
<td>10.411s</td>
</tr>
<tr>
<td>ReLU</td>
<td>0.504s</td>
<td>0.458s</td>
</tr>
<tr>
<td>Pooling</td>
<td>2.155s</td>
<td>3.049s</td>
</tr>
<tr>
<td>FC</td>
<td>0.159s</td>
<td>0.380s</td>
</tr>
</tbody>
</table>
Profiling of Naive Implementation

- **Convolution:**

<table>
<thead>
<tr>
<th></th>
<th>Forward</th>
<th>Backward</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>im2col()</td>
<td>dot()</td>
</tr>
<tr>
<td></td>
<td>0.871s</td>
<td>1.150s</td>
</tr>
<tr>
<td>dot()</td>
<td>1.150s</td>
<td></td>
</tr>
<tr>
<td></td>
<td>col2im</td>
<td>im2col</td>
</tr>
<tr>
<td></td>
<td>0.289s</td>
<td>0.877s</td>
</tr>
<tr>
<td>sum()</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Profiling of Naive Implementation

**Pooling:**

<table>
<thead>
<tr>
<th>Forward</th>
<th>Operation</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>im2col()</td>
<td>1.647s</td>
</tr>
<tr>
<td></td>
<td>argmax()</td>
<td>0.257s</td>
</tr>
<tr>
<td></td>
<td>transformation</td>
<td>0.249s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Backward</th>
<th>Operation</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>im2col()</td>
<td>1.661s</td>
</tr>
<tr>
<td></td>
<td>argmax()</td>
<td>0.219s</td>
</tr>
<tr>
<td></td>
<td>transformation</td>
<td>0.480s</td>
</tr>
<tr>
<td></td>
<td>col2im()</td>
<td>0.512s</td>
</tr>
</tbody>
</table>
Can we solve with Spark?

- Matrix Multiplication
- im2col()
Can we solve with Spark?

**BALANCE BETWEEN COMMUNICATION COST AND EXECUTION TIME**

**Matrix size:**
- $A = (1000 \times 32 \times 32) \times (5 \times 5 \times 3)$
- $B = (5 \times 5 \times 3) \times 32$
- $C = A \times B$
- $A + B = 1.08 \times 10^{10}$ bits ~ **1GB**

**doesn’t work!!!!!**

**Calculation:**
- NumPy.dot(): 0.672s
- Naive: $O(n^3)$ - REALLY SLOW!
- Outer Product: TOO MUCH MEM!

**Communication:**
- Speed: high-speed network

**Speed: high-speed network doesn’t work!!!!!!**
Can we solve with Spark?

- Batch Processing
## Spark Implementation

<table>
<thead>
<tr>
<th></th>
<th>Foward</th>
<th>Backward</th>
<th>Update</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAIVE</td>
<td>2.315</td>
<td>4.512</td>
<td>2.503</td>
<td>6.833</td>
</tr>
<tr>
<td>HDFS</td>
<td>21.030</td>
<td>5.504</td>
<td>2.506</td>
<td>34.701</td>
</tr>
</tbody>
</table>
Problems Arise

Returning intermediate results from forward run and reuse them in backward run: transferring huge amount of data back and forth, and creates gigantic RDD for backward run. - doesn’t work too well with Spark
State of the Art: TensorFlow
Observations

- Parameter Tuning
- Deployment of Trained CNN
Future Work

- Make forward execution and backward propagation for each batch executed on the same worker to reduce communication cost. - awareness of locality
- Polling: ensure all batch accepted by nodes, handle failure
- Compare Spark-CNN performance with GPU-CNN.
Reference

• Image CNN:
  http://i.imgur.com/qMs50Ma.png

• Wiki Convolutional Neural Network:

• CS231n Convolutional Neural Networks:
  http://cs231n.github.io/convolutional-networks/
Q & A

Thanks !