Using Approximate Computing to Improve the Efficiency of LSTM Neural Networks

Seyed Abolfazl Ghasemzadeh
Who Am I?

- M.Sc. in Electrical Engineering – Electronics
- Fields of interest:
  - Approximate Computing in LSTM Neural Networks
  - Machine Learning
  - Internet of Things
Contents

Introduction

Approximation in Hardware

Hardware Architecture

Conclusion
Neural Networks Applications
Neural Networks Hierarchy

Artificial Intelligence

Machine Learning

Deep NNs

Recurrent NNs
- Simple
- GRU
- LSTM

Feedforward NNs
- CNN
Neural Networks Hierarchy

- Artificial Intelligence
- Machine Learning
- Deep NNs
  - Recurrent NNs
    - Simple
    - GRU
    - LSTM
  - Feedforward NNs
    - CNN

- Natural Language Processing (NLP)
- Video Recognition
- Speech Recognition
Long Short-Term Memory NNs
Long Short-Term Memory Cells

- **$W$**: weight matrix
  - **$b$**: bias vector
- **$f$**: forget gate
  - **$i$**: input gate
  - **$g$**: candidate cell
  - **$o$**: output gate
- **$x$**: input vector
  - **$h$**: output vector
  - **$c$**: cell vector
Long Short-Term Memory Cells (cont.)

\[ f_t^* = W_{fx}x_t + W_{fh}h_{t-1} + b_f \]
\[ i_t^* = W_{ix}x_t + W_{ih}h_{t-1} + b_i \]
\[ g_t^* = W_{gx}x_t + W_{gh}h_{t-1} + b_g \]
\[ o_t^* = W_{ox}x_t + W_{oh}h_{t-1} + b_o \]

\[ f_t = \text{sig}(f_t^*) \]
\[ i_t = \text{sig}(i_t^*) \]
\[ g_t = \tanh(g_t^*) \]
\[ o_t = \text{sig}(o_t^*) \]

\[ c_t = f_t \odot c_{t-1} + i_t \odot g_t \]
\[ h_t = o_t \odot \tanh(c_t) \]
Long Short-Term Memory Cells (cont.)

Data Storage

Matrix-Vector Multiplier

Activation Function

Point-wise Multiplier and Adder

Timing Controller
Hardware Realization

CPU

GPU

Overheat

Increase in Power Usage

Use Powerful Cooling System
Hardware Realization

Decrease in Power Usage

Hardware Accelerator
Challenges in Design

- Limited Power Budget
- Limited Memory Storage and Bandwidth
- Limited Hardware Recourses
Approximate Computing

- Weight Pruning
- Quantization
- Approximate Computation
## Approximate Computing Efficiency

<table>
<thead>
<tr>
<th>Hardware Model</th>
<th>DeltaRNN</th>
<th>ESE</th>
<th>FP-DNN</th>
<th>CPU</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>XC7Z100</td>
<td>XCKU060</td>
<td>GSMD5</td>
<td>i7-8700K</td>
<td>GTX 1080 Ti</td>
<td></td>
</tr>
<tr>
<td>Quantization</td>
<td>Fixed16</td>
<td>Fixed12</td>
<td>Fixed16</td>
<td>Float32</td>
<td>Float32</td>
</tr>
<tr>
<td>Effective Throughput [G(FL)Op/s]</td>
<td>1198</td>
<td><strong>2520</strong></td>
<td>315.85</td>
<td>18.78</td>
<td>123.34</td>
</tr>
<tr>
<td>Power [W]</td>
<td><strong>7.3</strong></td>
<td>41</td>
<td>25</td>
<td>35.6</td>
<td>95.9</td>
</tr>
<tr>
<td>Power Efficiency [G(FL)Op/s/W]</td>
<td><strong>164.11</strong></td>
<td>61.46</td>
<td>12.63</td>
<td>0.53</td>
<td>1.29</td>
</tr>
</tbody>
</table>
Approximate Computing Efficiency (cont.)

In the diagram, two curves are plotted to compare the speedup with and without load balance as a function of the percentage of parameters pruned away. The red curve with circles labeled "with load balance" shows a 6.2x speedup over dense, and the green curve with triangles labeled "without load balance" shows a 5.5x speedup over dense.

[Source: ESE - S. Han’s ACM/SIGDA 2016]
Matrix Vector Multiplication

<table>
<thead>
<tr>
<th></th>
<th>PE1</th>
<th>PE2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3</td>
<td>0.1</td>
<td>0.2</td>
</tr>
<tr>
<td>0.3</td>
<td>0.4</td>
<td>0.6</td>
</tr>
<tr>
<td>0.7</td>
<td>0.8</td>
<td>-0.6</td>
</tr>
<tr>
<td>0.2</td>
<td>-0.6</td>
<td>0.6</td>
</tr>
</tbody>
</table>

The diagram shows the matrix and vector multiplication process.
Pruning Patterns

Unstructured Sparse

<table>
<thead>
<tr>
<th></th>
<th>0.3</th>
<th>0.1</th>
<th>0.2</th>
<th>-0.1</th>
<th>0.4</th>
<th>-0.5</th>
<th>0.2</th>
<th>0.6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3</td>
<td>0.4</td>
<td>0.6</td>
<td>0.1</td>
<td>0.3</td>
<td>0.2</td>
<td>0.5</td>
<td>-0.5</td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td>0.8</td>
<td>-0.6</td>
<td>0.6</td>
<td>-0.2</td>
<td>0.5</td>
<td>0.3</td>
<td>0.1</td>
<td></td>
</tr>
<tr>
<td>0.2</td>
<td>-0.6</td>
<td>0.6</td>
<td>0.5</td>
<td>0.1</td>
<td>0.2</td>
<td>0.3</td>
<td>0.7</td>
<td></td>
</tr>
</tbody>
</table>
## Pruning Patterns

<table>
<thead>
<tr>
<th>Unstructured Sparse</th>
<th>PE1</th>
<th>PE2</th>
<th># of non-zeros</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.4</td>
<td>-0.5</td>
<td>0.6</td>
</tr>
<tr>
<td></td>
<td>0.4</td>
<td>0.6</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td>0.7</td>
<td>0.8</td>
<td>-0.6</td>
</tr>
<tr>
<td></td>
<td>-0.6</td>
<td>0.6</td>
<td>0.5</td>
</tr>
</tbody>
</table>

- PE1: Green
- PE2: Purple

- 3 non-zeros
- 4 non-zeros
- 5 non-zeros
- 4 non-zeros
Pruning Patterns

Bank-Balanced Sparse [C. Shijie’s ISFPGA 2019]

<table>
<thead>
<tr>
<th></th>
<th>0.3</th>
<th>0.1</th>
<th>0.2</th>
<th>-0.1</th>
<th>0.4</th>
<th>-0.5</th>
<th>0.2</th>
<th>0.6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3</td>
<td>0.4</td>
<td>0.6</td>
<td>0.1</td>
<td>0.3</td>
<td>0.2</td>
<td>0.5</td>
<td>-0.5</td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td>0.8</td>
<td>-0.6</td>
<td>0.6</td>
<td>-0.2</td>
<td>0.5</td>
<td>0.3</td>
<td>0.1</td>
<td></td>
</tr>
<tr>
<td>0.2</td>
<td>-0.6</td>
<td>0.6</td>
<td>0.5</td>
<td>0.1</td>
<td>0.2</td>
<td>0.3</td>
<td>0.7</td>
<td></td>
</tr>
</tbody>
</table>
Pruning Patterns

Bank-Balanced Sparse [C. Shijie’s ISFPGA 2019]

<table>
<thead>
<tr>
<th></th>
<th>PE1</th>
<th>PE2</th>
<th></th>
<th>PE1</th>
<th>PE2</th>
<th># of non-zeros</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3</td>
<td></td>
<td></td>
<td>-0.5</td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>0.4</td>
<td>0.6</td>
<td></td>
<td>0.5</td>
<td>-0.5</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>0.7</td>
<td>0.8</td>
<td></td>
<td>0.5</td>
<td>0.3</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>-0.6</td>
<td>0.6</td>
<td></td>
<td>0.3</td>
<td>0.7</td>
<td></td>
<td>4</td>
</tr>
</tbody>
</table>
Pruning Algorithm

Train

• Train the network

Prune

• Prune the weights

Retrain

• Fine-tune the network
Activation Functions Approximation

- Linear \((ax + b)\)
- Second-order \((ax^2 + bx + c)\)
- Taylor Series
An LSTM Accelerator

\[ f_t = \text{sig}(f_t^*) \]
\[ i_t = \text{sig}(i_t^*) \]
\[ g_t = \text{tanh}(g_t^*) \]
\[ o_t = \text{sig}(o_t^*) \]

\[ f_t^* = W_{fx}x_t + W_{fh}h_{t-1} + b_f \]
\[ i_t^* = W_{ix}x_t + W_{ih}h_{t-1} + b_i \]
\[ g_t^* = W_{gx}x_t + W_{gh}h_{t-1} + b_g \]
\[ o_t^* = W_{ox}x_t + W_{oh}h_{t-1} + b_o \]

\[ c_t = f_t \odot c_{t-1} + i_t \odot g_t \]
\[ h_t = o_t \odot \text{tanh}(c_t) \]
Experimental Setup

- **Training and Pruning**
  - On:
    - IMDB
    - Japanese Vowels
    - Chickenpox
    - Penn Tree Bank (PTB)
    - TIMIT

- **Hardware Modeling**
  - Bit width of the datapath
  - The widths of the integer and fractional parts
  - The number of intervals for implementing the activation functions

- **Implementation**
  - Verilog
  - Power Consumption
  - Resource Utilization
Approximation

• To reduce the power consumption of accelerator

Hardware Accelerator

• For customizing the implementation of neural networks
References


References (cont.)


THANKS FOR YOUR ATTENTION!

Any Questions?