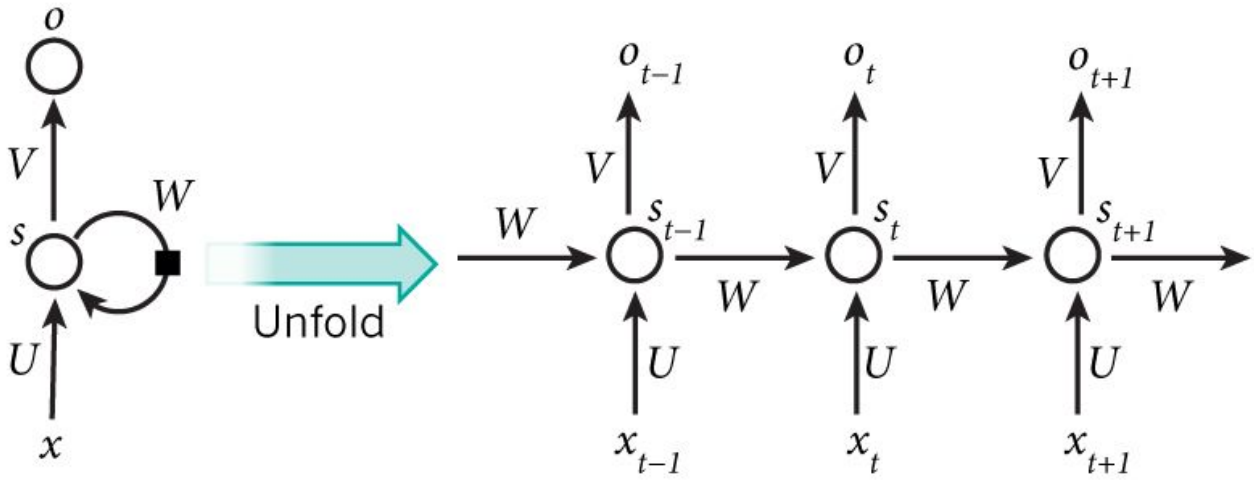
Carnegie Mellon University

Autogenerating Text via Parallel Implementations of Recurrent Neural Networks

Anirban Ghosh

Taylor Clarke



Introduction

Background: Neural networks and machine learning in general have been the focus of much computer research for the better part of 50 years, and their ubiquity is only growing. A neural network is a machine learning technique used to process multifaceted, multidimensional data by simulating classification in much the same way a human brain does. Recurrent neural networks (RNNs) are often used to process time-series or highly correlated data by forcing dependencies to exist between sequential inputs [1]. Thus, the technique is highly applicable to natural language processing and specifically the automatic generation of text, which is the focus of our study.

Problem: In order to decrease the amount of time required to train a recurrent neural network, researchers have implemented various methods of parallelizing the training algorithm. We set out to study these methods of parallelization and improve upon them by testing and implementing them in various hardware and software setups.

Approach: The first part of our study focused on **investigating the viability of RNNs on the Xeon Phi**. We found many issues regarding this problem. The second part of our study involved **studying the differences between two different RNN frameworks**.

Related Work: Our work involved testing many different software packages and tools to implement RNNs, but two major packages are of note: **char-rnn** by Andrej Karpathy, and **faster-rnnlm** by Yandex.

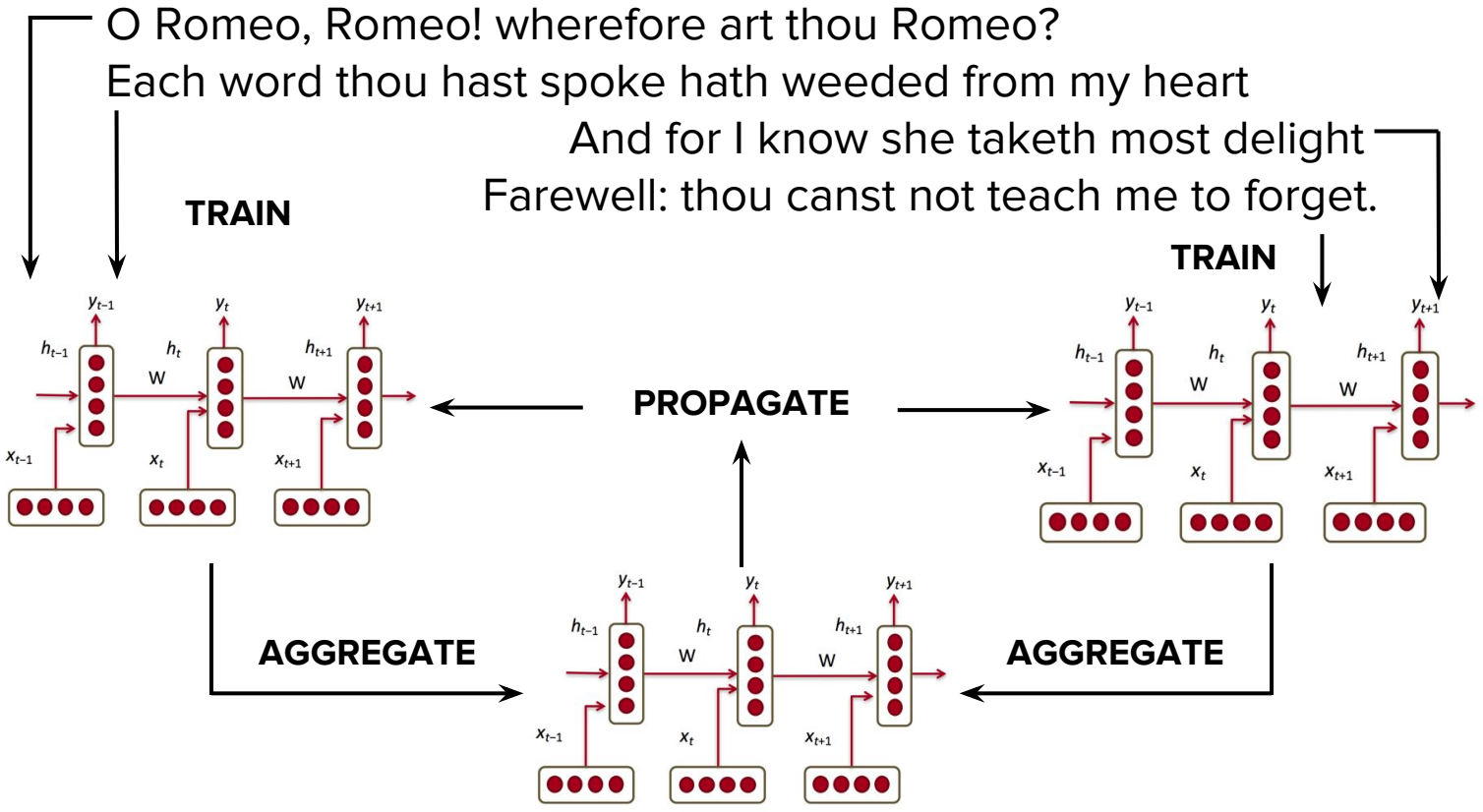
How to Parallelize an RNN?

Background: Two main methods of parallelizing neural networks: by **model** and by **data** [2].

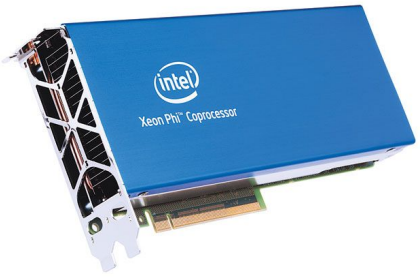
- **Model parallelism:** training data is split between layers and/or nodes in the network
- **Data parallelism:** training data is split between asynchronous threads and results are combined later

Methods: Both **char-rnn** and **faster-rnnlm** use **data parallelism**.

Training: We trained each neural network with a ~1 MB dataset consisting of concatenated Shakespeare text.



Torch vs. Xeon Phi



Torch: "a scientific computing framework with wide support for machine learning algorithms" [3]. Built in LuaJIT, C, and CUDA, it is widely used to implement neural networks in research and industry.

Xeon Phi: "Intel's first bootable host processor specifically designed for highly parallel workloads" [4]. An accelerator with 64 - 72 cores capable of running x86-64 code natively or "offloaded" from a host program.

Methods: We set out to implement an RNN via Torch on the Xeon Phi, and it was much more difficult than predicted.

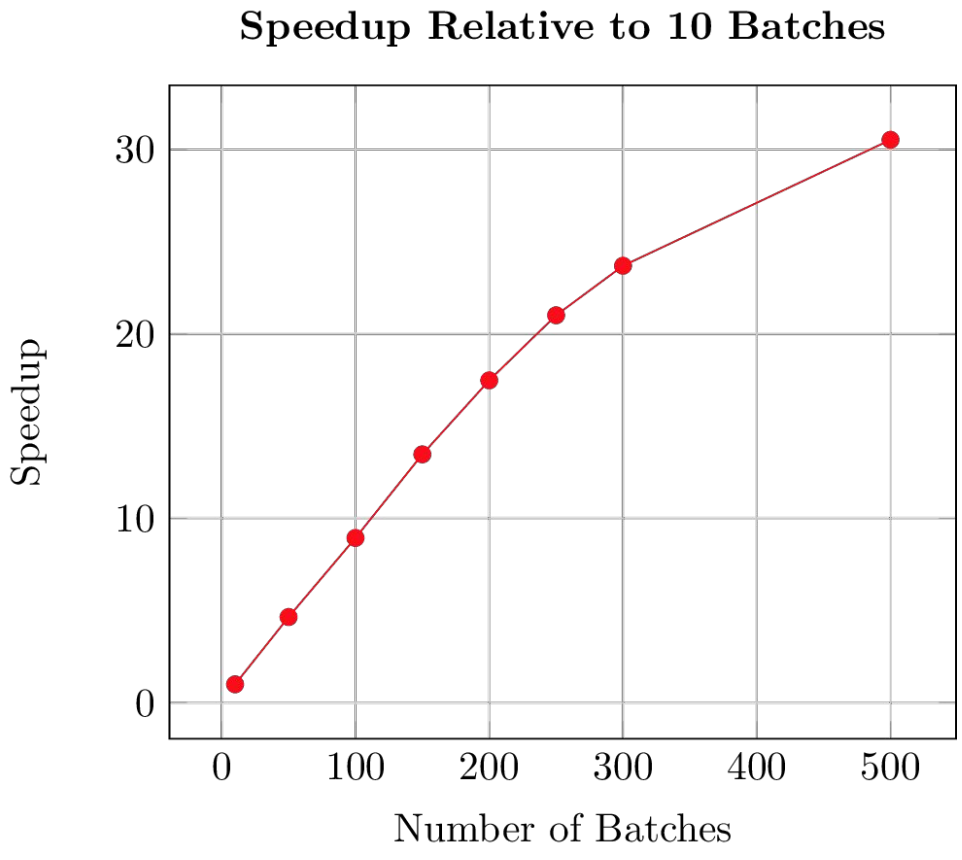
1. **OpenCL:** The Intel SDK that lets the Phi support OpenCL could not run on our OS [5].
2. **Colfax Research:** They have developed a version of Torch that runs on the Phi, but the public version still only runs CUDA [6].
3. **Intel:** Public version on Github is not up to date with the latest Torch packages, so it refuses to build [7].
4. **Faster-RNNLM:** In attempting to translate a C++ framework that does not use Torch, we found that it was impossible to run on the Phi due to the need to bitwise copy all data onto the Phi [8].

Char-RNN

Background: “takes one text file as input and trains a Recurrent Neural Network that **learns to predict the next character in a sequence**” [9]. Developed by Andrej Karpathy at Stanford. In CUDA, trains on a given number of batches simultaneously.

Sample Output: Seeded with “I”: “I' have desinutenate are Which eye onure to think “

Experimentation: Removed parallelization of small loops to test if operations take up similar computation time to Faster-RNNLM. Confirmed that removing parallelization of most tight loops does little to slow down the algorithm, since most of the bottleneck is in I/O.

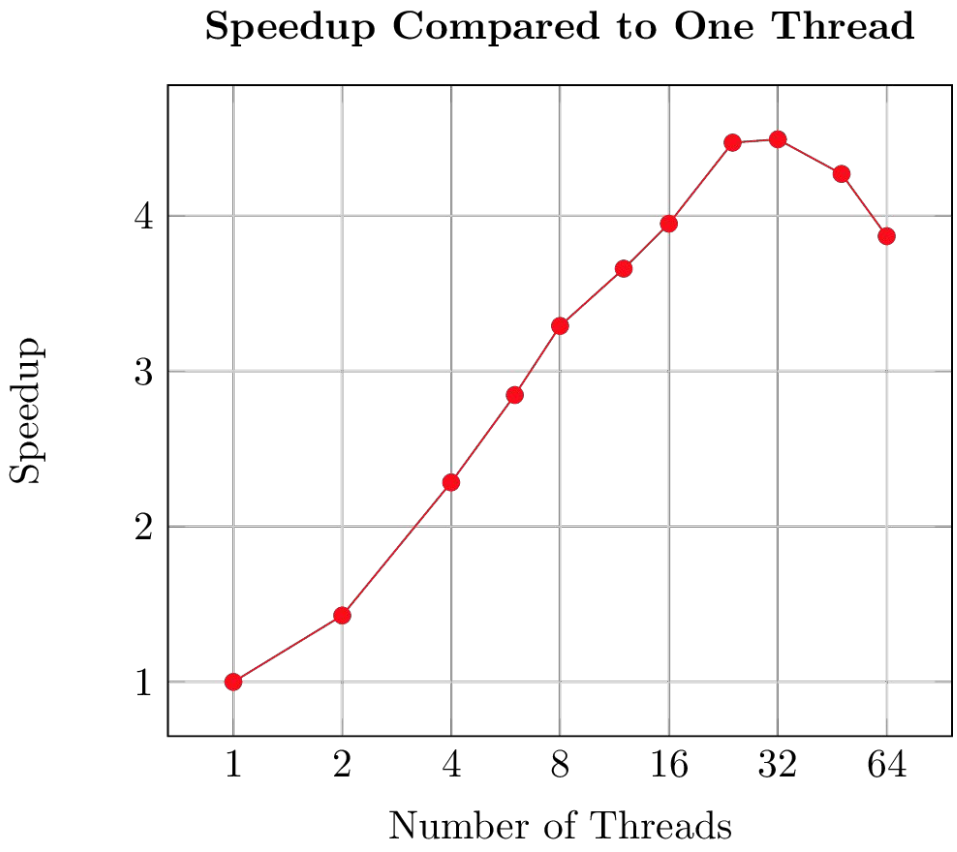


Faster-RNNLM

Background: “create an rnnlm (recurrent neural network language modeling) implementation that can be trained on huge datasets ... and very large vocabularies” [10]. Developed by Russian tech company Yandex. **Learns words instead of characters.**

Sample Output: Seeded with “I”: “I know so. DUCHESS they think shame, And man, the shout didst the queen to the curtsy”

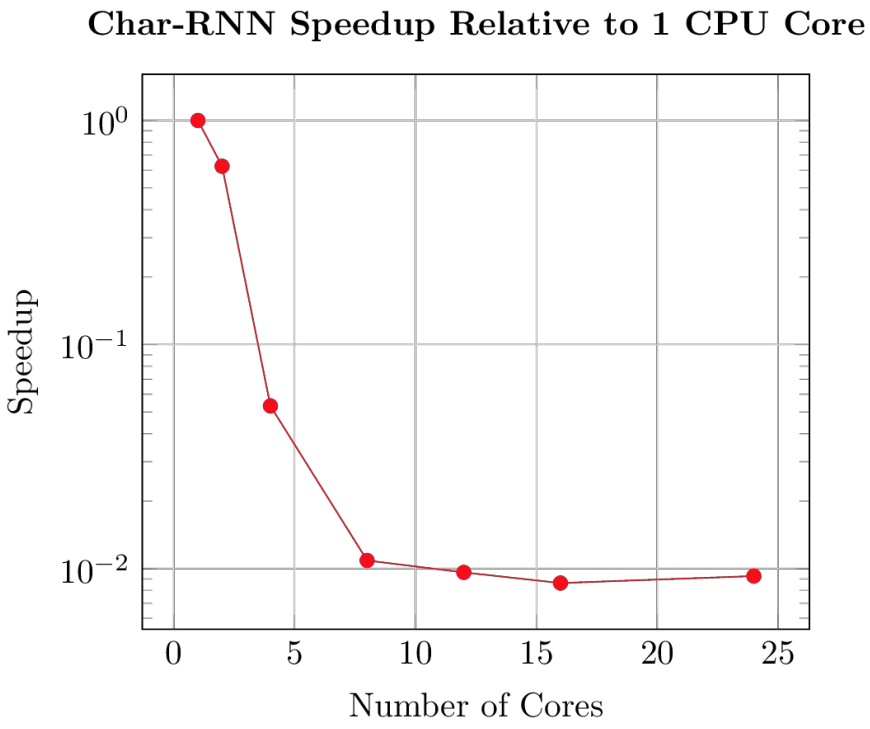
Experimentation: Refactored threads in training algorithm to read from C++ stringstream instead of text file, and C++ string instead of text file; neither approach helped due to their file reading optimizations being faster than C++ parsing.



Comparison

CUDA vs. CPU:

- Char-RNN was built to run efficiently via Torch on a CUDA GPU, so its I/O and cache misses are prohibitively costly when run on a CPU
- On the other hand, Faster-RNNLM was built solely for a CPU, so its performance increases as processing power is added



Char-RNN vs Faster-RNNLM:

- Char-RNN has the capacity to run on a GPU and run through massive amounts of text data very rapidly (necessary when training on characters instead of words)
- Faster-RNNLM is built in C++ and has the ability to run on most home computers; gives meaningful text more quickly but has less freedom since no Torch backing framework
- Char-RNN has linear speedup with number of CUDA batches due to efficient Torch implementation; Faster-RNNLM slows down after number of physical cores are saturated due to training file I/O

Summary

Conclusions:

- Developing for the Xeon Phi can be really difficult!
- Char-RNN is missing a lot of potential speedup on CPU due to interprocess communication
- Faster-RNNLM is well-structured but could take advantage of more threads by reducing file I/O

Future Work:

- Most of the work we would like to do is to continue trying to implement an RNN on the Xeon Phi. There are multiple potential paths to achieve this, and we believe OpenCL is the most viable option due to the existence of proven libraries interfacing OpenCL and Torch, and OpenCL and the Xeon Phi.
- We would also like to investigate RNNs that potentially use model parallelism, ie parallelizing over the neural network itself, instead of data parallelism and test the effects of this parallelism with various training parameters.

References:

[1] http://karpathy.github.io/2015/05/21/rnn-effectiveness/
[2] https://arxiv.org/pdf/1404.5997v2.pdf
[3] http://torch.ch/
[4] http://www.intel.com/content/www/us/en/processors/xeon/xeon-phi-processor-product-brief.html
[5] https://github.com/hughperkins/distro-cl
[6] https://github.com/ColfaxResearch/Torch-distro
[7] https://github.com/intel/torch
[8] https://software.intel.com/sites/default/files/managed/05/ba/heterogeneous-programming-model.pdf
[9] https://github.com/karpathy/char-rnn
[10] https://github.com/yandex/faster-rnnlm
[11] https://github.com/jcjohnson/torch-rnn