INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

Artificial Intelligence

Challenge: Neural Network Parallelization

Team:

Melannie Isabel Torres Soto	A01361808
Shara Teresa González Mena	A01205254

Justification

Artificial Neural Networks are a popular machine learning technique that attempts to replicate how the human brain works, they are able to solve problems that humans tend to do well but computers don't, but the complexity of neural networks makes them computationally expensive to simulate.

To solve problems, ANN require certain amount of training data to train themselves and be able to solve problems like humans. The high amounts of training data plus the learning techniques of ANN's makes this process very slow and decrease its performance. This is why there have been a variety of attempts to reduce the training time of neural networks.

A popular ANN is the multilayer perceptron, in which neurons are organized in at least three layers. The first is called input layer, the second hidden layer which divides into n hidden layers and the third is the output layer. Backpropagation Neural Networks (BPNN) is a type of multilayer network that uses backpropagation to adjust the weights and the biases in neurons but the computations needed for backpropagation makes this ANN.

In the process of backpropagation, a learning rate is computed, which is a rate that influences the swiftness and quality of learning. With greater learning rate, the training is faster but with smaller learning rate the training is more accurate. This is the reason why wanting to accurately train over a high amount of data, makes the computation of backpropagation really computationally expensive.

To improve the performance of neural networks, there have been works centered on hardware implementations that provide a high degree of parallelism, such as massively parallel processors, also other works that use cluster computers, among others. In the other side, there have been other attempts centered on the implementation itself such as parallelizing the algorithm or the computations in the training process, etc.

Parallelization techniques

As ANN's computations in each node are generally independent from other nodes, makes the network suitable for parallel computing. Parallelizing computations in BP training has been explored in many ways, Tomas Nordstrom and Bertil Svensson presented both of the techniques listed below in their research *Using and designing massively parallel computers for artificial networks* (2004) (Note: The implementations depend on the constraints of the hardware platform used):

Training session parallelism

This technique consists in starting different training iterations (sessions) in different processing elements, for example computers in a cluster, the dataset is divided and assigned to a processing element, each session will have its own weights and learning rates. This way, different networks can be trained simultaneously with different parts of the dataset and the objective is looking for the best result from all of the processing elements.

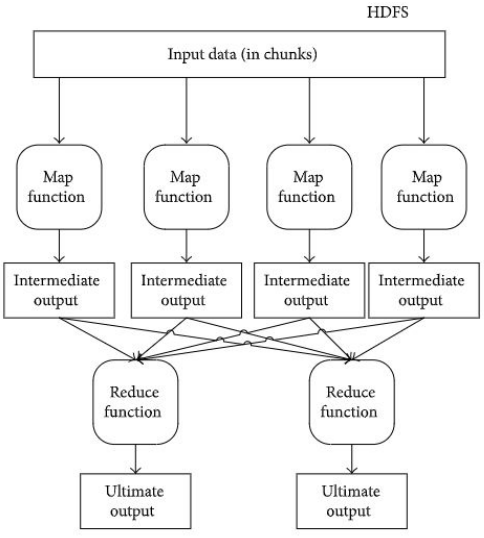
Exemplar parallelism

The dataset is divided, by a parent node, into subsets and assigned, with a random weight (the same for all the nodes), to every node (processing elements), each node will train with the given data and weight but after each epoch in the cluster, after computing its error, each processing element should report its error to the parent node. The parent node, computes the new weight with the given errors and starts the iteration again assigning that weight to all the nodes.

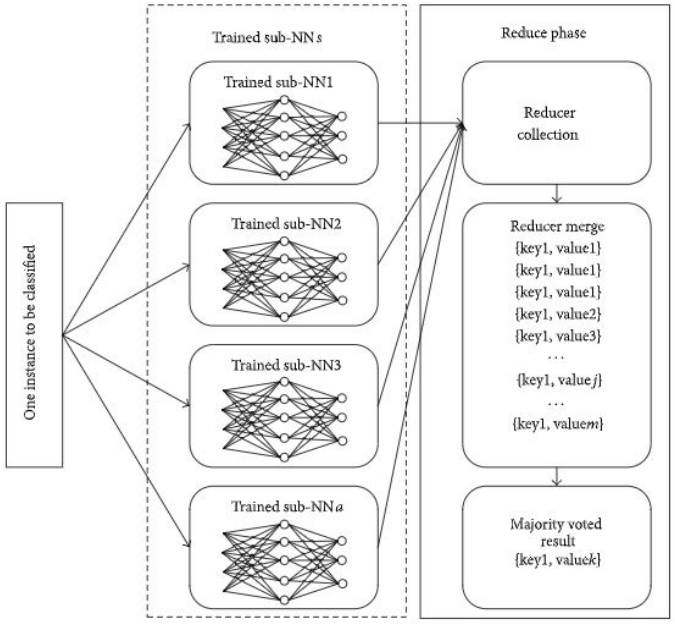
Map-Reduce approach

Map-Reduce is a distributed computing model that contains two operations: map and reduce. The map operation takes a dataset and converts it into another dataset where the elements are broken down into tuples with a key and a value, then the reduce operation is performed, which takes the output of map and combines the data into smaller set of tuples.

In 2015 Yang Liu, Weizhe Jing and Lixiong Xu proposed a parallelization of classification in BPNN's by following the Map-Reduce model:



The input data, divided into chunks as the model presents, is assigned to *n* number of BPNNs called sub-BPNNs. The intermediate output in the model represents the training instance or sub-BPNN and the reduce part was made after each separated node produced its output, reducing all the results obtained to get the final output of classifications.
The classification Map-Reduce parallelization is presented in the image below:



Documentation

We decided to parallelize the testing section of the algorithm by dividing the data into smaller blocks and assigning a processor to each block. The concurrent solutions were made using the java's framework Fork-Join and OpenMp.

Instructions to run the program

1. Download the project from the following github repository:

   https://github.com/MelannieTorres-academico/AI_Perceptron

2. To run the java program enter the java folder: cd fork-join-perceptron
3. Compile the java programs type the following command: javac *.java
4. To run the example type: java MainForkJoinPerceptron < ./tests/breast_cancer.txt
5. To run your own dataset type: java MainForkJoinPerceptron < path_to_your_dataset.txt
6. Go back to the main folder: cd ..
7. Go into the C++ folder: cd c++
8. To compile the C++ files type: g++ sequential.cpp
9. To run the program type: ./a.out < ./tests/breast_cancer.txt

10. To run the program with your own dataset type: ./a.out < path_to_your_dataset.txt
11. Go back to the main folder: cd ..
12. Go into the python folder: cd python
13. Run the python program: python3 ann.py < ./tests/breast_cancer.txt
14. To run with your own dataset: python3 ann.py < path_to_your_dataset

Take into account the following considerations when creating your own dataset:

a. The file should be a txt
b. The values should be separated by commas
c. The dataset must be all numeric
d. The dataset must be linearly separable
e. The output must be binary (0 or 1)
f. The first number indicates the number of attributes
g. The second number indicates the number of records used to train
h. The third number indicates the number of tests to run
i. The following lines should be the training records and the tests to run

Example

In the following example we used a breast cancer dataset obtained from the UCI repository. We cleaned the dataset to make it linearly separable and we copy and pasted the tests over and over again in order to have 60,000 tests and appreciate the improvement of using concurrent technologies.

```
melannie@MacBook-Air-de-Melannie ~/Desktop/ai/c++ cd ..
melannie@MacBook-Air-de-Melannie ~/Desktop/ai cd fork-join-perceptron

melannie@MacBook-Air-de-Melannie ~/Desktop/ai/fork-join-perceptron javac *.java
melannie@MacBook-Air-de-Melannie ~/Desktop/ai/fork-join-perceptron java MainForkJoinPerceptron < ./tests/breast_cancer.txt

Sequential:
  avg time: 2.1

Fork Join:
  avg time: 1.7

Speed up: 1.2352941176470589

melannie@MacBook-Air-de-Melannie ~/Desktop/ai/fork-join-perceptron cd ..
melannie@MacBook-Air-de-Melannie ~/Desktop/ai cd c++
melannie@MacBook-Air-de-Melannie ~/Desktop/ai/c++ g++ sequential.cpp
melannie@MacBook-Air-de-Melannie ~/Desktop/ai/c++ ./a.out < ./tests/breast_cancer.txt

Sequential C++ avg time = 4.785 ms

TBB avg time = 3.363 ms

Speed up: 1.42284
```

```
melannie@MacBook-Air-de-Melannie ~/Desktop/ai/c++ cd ..
melannie@MacBook-Air-de-Melannie ~/Desktop/ai cd python
melannie@MacBook-Air-de-Melannie ~/Desktop/ai/python python3 ann.py < ./tests/breast_cancer.txt
time 183.53700637817383 ms
```

The time results were the following:

Technology used	Time
Python	183.5 ms
C++	4.748 ms
OpenMp with C++	3.363 ms
Java	2.1 ms
Java Fork-Join	1.7 ms

Analysis

As seen in several works and researches, the most common way of parallelizing a neural network is by splitting the data and create several nodes or processing elements that will train with the given data, the way of interpreting or retrieving the result of each node, is what varies from technique to technique. The works presented above obtained the following results:

Exemplar parallelization test

Haidar Sharif and Osman Gursoy presented and exemplar parallelization implementation of a neural network and executed two different tests (2018), one with 10000 samples of data (Test A) and the other with 30000 (Test B) and what they observed was that as the number of processors increased, a decrease in the execution time occured:

- With just one processor, the sequential execution, lasted with the Test A 192.53 sec. while in the Test B 221.89 sec.
- With two processors Test A lasted 107.49 sec. and Test B 115.16
- With three processors Test A 84.16 sec. and Test B 84.57
- With four processors Test A 75.56 sec. and Test B 74.15

Perceptron test

We made an implementation of a perceptron and splitted the testing dataset to parallelize it. To test that the Perceptron algorithm actually worked in java, java fork-join, c++ and OpenMP with c++ we used the test cases given in the ANN lab. With these tests we didn't obtained a speed up since the amount of data is too small. This is the reason why we used a larger dataset and replicated the lines dedicated for testing over and over again. Doing this the number of calculations our program has to go are much larger and we can see the impact of parallelizing the testing algorithm.

Using this dataset we found out that using java fork-join is 107.9 times faster than using python for the Perceptron algorithm. This shows us that choosing the right technology for an AI algorithm can deeply affect the time response.

While doing this investigation we adapted an existent Neuronal Network Backtracking python program created by Shamdasani in 2017 for it to parse the dataset from the .txt file. Our plan was to parallelize that algorithm in python, but when we increased the input data from 1,000 tests to 10,000 tests the python program took too long and consumed the all memory available in the computer sending an OS error. This program is available in the python folder with the name: nnbt.py. Since we had a memory limitation in this python algorithm we couldn't parallelize it.

We found out there are better implementations than ours like the ones presented in the introduction because they parallelize the training instead of the testing.

This challenge teached us that selecting a wrong technology to develop an AI algorithm could limit you to achieve your goals like in the last python example where we were unable to even run the program with that quantity of data.

References:

Dua, D. and Karra Taniskidou, E. (2017). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science. Retrieved from https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/breast-cancer-wisconsin.data

Gursoy, O., & Sharif, H. (2018, February). Parallel Computing for Artificial Neural Network Training. Retrieved from http://pen.ius.edu.ba/index.php/pen/article/view/143/182

Ji, P., Wang, P., Zhao, Q., & Zhao, L. (2011, September). A new parallel back-propagation algorithm for neural networks. Retrieved from https://ieeexplore.ieee.org/abstract/document/6044075

Liu, Y., Jing, W., & Xu, L. (2016, April 27). Parallelizing Backpropagation Neural Network Using MapReduce and Cascading Model. Retrieved from https://www.hindawi.com/journals/cin/2016/2842780/

Nordström, T., & Svensson, B. (2004, February 19). Using and designing massively parallel computers for artificial neural networks. Retrieved from https://www.sciencedirect.com/science/article/pii/074373159290068X

Pethick, M., Liddle, M., Werstein, P., & Huang, Z. (2005). Parallelization of a Backpropagation Neural Network on a Cluster Computer. Retrieved from http://www.cs.otago.ac.nz/staffpriv/hzy/papers/pdcs03.pdf

Schroder, J. (2017, October 3). Parallelizing Over Artificial Neural Network Training. Retrieved from https://arxiv.org/pdf/1708.02276.pdf

Shamdasani, S. (2017). Build a flexible Neural Network with Backpropagation in Python. Retrieved from https://dev.to/shamdasani/build-a-flexible-neural-network-with-backpropagation-in-python