INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

Using Fork-Join in the Tests of a Perceptron

Programming Languages - Final Project Report

Melannie Isabel Torres Soto A01361808

November 20, 2018

Context of the problem

Artificial Neuronal Networks (ANNs) are used in multiple domains, among them we have medicine, voice recognition, fraud detection, credit rating and targeted marketing (Singth, 2017). To use a ANN first we have to train it, after it is trained, testing for a new result requires a dot product calculation. Testing an ANN may not seem too costly, but if we have hundreds of data records to test with multiple attributes each one, it could be, in order to reduce the execution time of this calculation, a *Fork-Join* implementation will be presented.

Fork-join is a Java framework that allows us to take advantage of multiple processors. This framework uses an algorithm named *work-stealing* in which the threads that finish their job may steal tasks from those who haven't finished. *Fork* is the phase in which the tasks are broken into smaller tasks and *join* in the phase where the result of this task are reunited to deliver a final result.

To measure the increment in the velocity of the fork-join version versus the sequential one, a metric named *speed up* will be used. The *speed up* is the ratio between the sequential execution time and the concurrent execution time.

$$S_p = \frac{T_1}{T_p}$$

Where T_1 is the execution time of the sequential program and T_p is the execution time in a concurrent or parallel way using p processors.

Training the ANN

An Artificial Neuronal Network (ANN) is a collection of nodes interconnected by links whose weight show the strength of the relationship between those nodes. They work in the following way: given some input, it calculates the output, it compares this output to the expected one, then it make an adjustment to the strength of the relationships and it repeats the process over and over again until the expected output and the actual output are equal or with an acceptable error. ANN is a supervised learning algorithm since it's trained with a dataset that contains the expected result. The most basic implementation of this algorithm is known as Perceptron.

The *Perceptron* algorithm contains a single node that receives n inputs, it adds an additional "input node" with a value of 1, uses random strengths, makes a dot product between the inputs and the strengths, with this result it calculates the output using a threshold or step function, if the result is greater than 0 the output is 1, otherwise it is 0, this output is compared to the expected one, if they're not equal, the strengths are adjusted, and the output is recalculated until the output is equal to the expected output. The only possible outputs for the *Perceptron* algorithm are 0 and 1.

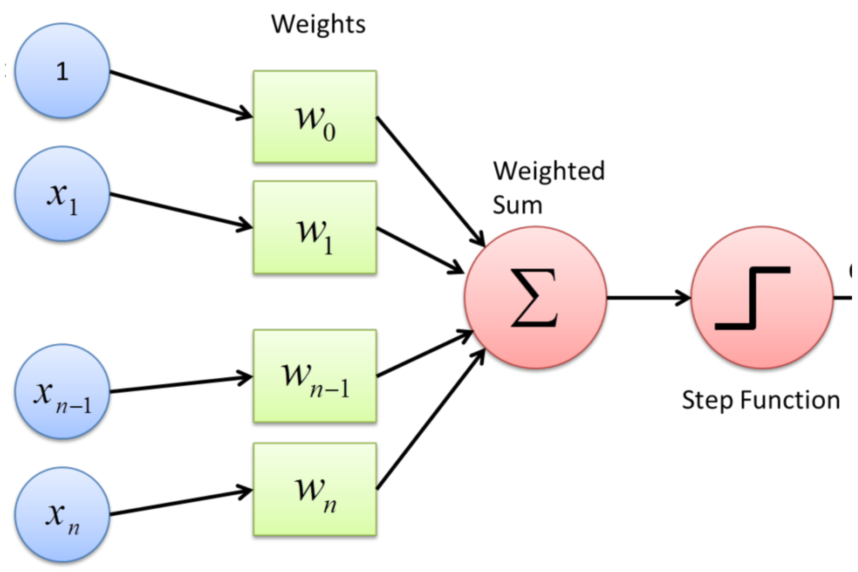

Image from Abhay - A hands-on tutorial on the Perceptron learning algorithm

The *Perceptron* algorithm only works for linearly separable data. For example the perceptron algorithm would work for the data on the left, but not for the one on the right.

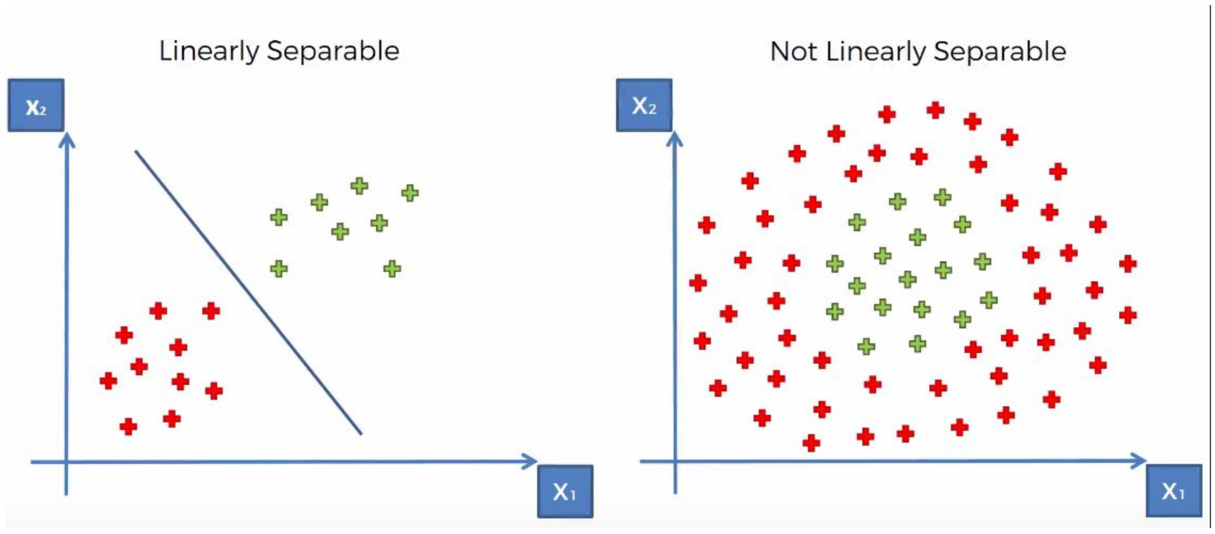

Image from Arun Manglick - Artificial Intelligence & Machine/Deep Learning

Testing the ANN

Once the ANN is trained, we may proceed to test it. For each test case it will calculate the output doing a dot product between the inputs and the strengths obtained in the training phase. If we have a few test cases, doing a fork-join implementation wouldn't help us to improve the execution time, however if we are trying to obtain the results of a huge dataset, we could do this task faster using Fork-Join or with some other concurrent/parallel technology.

Solution

This is the part of the algorithm that was implemented using the Fork-Join framework and also in a sequential way to see impact of the Fork-Join.

Fork-Join works in the following way, if the task is to big, it breaks the task in two and assigns each task to a thread, if the task has an acceptable size the thread proceeds to process it.

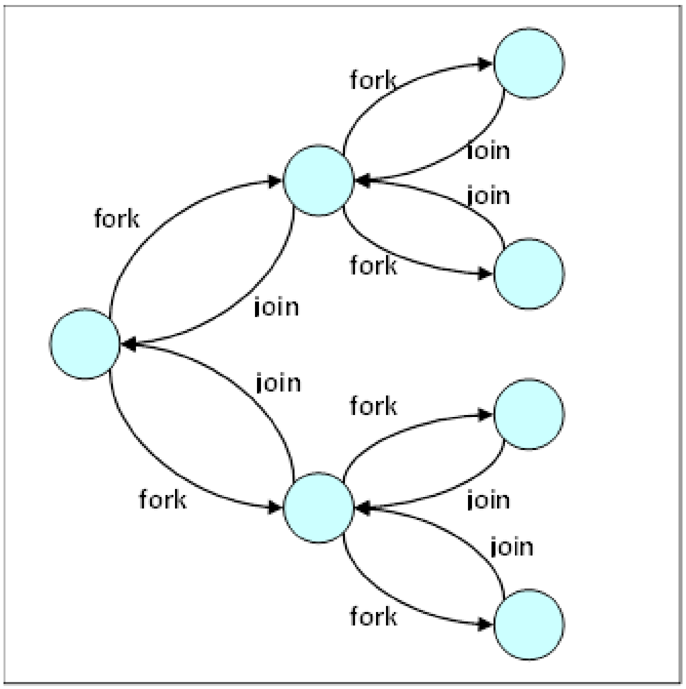

Image from Nitin - Java 7 Fork/Join Framework

For this code, the task is considered acceptable if the number of test cases are 6,000 or less. This means that if you have more test cases it will break it into several threads until each thread cas 6,000 test cases or less.

The results of the test cases are saved in an array which can be printed by adding the flag -true when running the program.

The code of this solution is in the following repository:
https://github.com/MelannieTorres-academico/fork-join-perceptron

Set up Instructions

To use it you can clone it or download it:

git clone https://github.com/MelannieTorres-academico/fork-join-perceptron

To compile use: javac *.java

To run use: java MainForkJoinPerceptron < path_to_dataset

You can use the given datasets to test it, to know more about the test files and how to use your own dataset consult:
https://github.com/MelannieTorres-academico/fork-join-perceptron/blob/master/tests/tests_expectations.txt

Results

To test that the algorithm was working as expected some test cases given by Ruben Stranders in his AI course were used. This test cases include non-linearly separable data and some linearly separable data. The algorithm works only for linearly separable data and works as expected.

After validating it was working, the program was given a larger dataset containing 60,000 tests. This file is available at:
https://github.com/MelannieTorres-academico/fork-join-perceptron/blob/master/tests/breast_cancer.txt

This dataset was obtained from the UC Irvine Machine Learning Repository, it was modified to make the data linearly separable and the tests were repeated over and over in order to have more evaluations.The algorithm does not train with the tests given.

To calculate the speed up the program was run with the previously mentioned dataset in a computer with the following characteristics:

- o Mac OS Sierra 10.12.6
- o Processor: 1.3 GHz Intel Core i5
- o Memory: 4 DB 1600 MHz DDR3

The results were the following:

```
Sequential:
  avg time: 2.2

Fork Join:
  avg time: 1.5

Speed up: 1.4666666666666666
```

This means that the fork-join is 1.46 times faster than the sequential version.

The *speed up* was also calculated for smaller files, these didn’t show a speed up since the amount of calculations don’t require a fork, there for they were also run in a single thread as the sequential one.

Areas of opportunity

To have a better performance the training of the ANN could be parallelized but it should use a different algorithm than the *Perceptron* since this one cannot be parallelized. For example a full ANN could be parallelized by making the calculations of each layer in a different thread. Another improvement that could be made is using other technologies for this algorithm like CUDA, tbb, Openmp to see which of these is the most adequate to our needs and gives us the best speed up.

Conclusion

The fork-join framework can help us to reduce the execution time of a program if it requires a lot of calculations, but if there are just a few calculations the execution time won't improve using this framework.

References

Abhay. (2016). A hands-on tutorial on the Perceptron learning algorithm. Retrieved from http://abhay.harpale.net/blog/machine-learning/a-hands-on-tutorial-on-the-perceptron-learning-algorithm/

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

ForkJoinPool (Java Platform SE 8). (2018). Recuperado de https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ForkJoinPool.html

Java - Fork/join, the work-stealing thread pool. (2017). Retrieved from https://www.logicbig.com/tutorials/core-java-tutorial/java-multi-threading/fork-and-join.html

Manglick, A. (2017). Artificial Intelligence & Machine/Deep Learning http://arun-aiml.blogspot.com/2017/07/kernel-svm-support-vector-machine.html

Nitin. K. (2012). Java 7 Fork/Join Framework. Retrieved from https://www.developer.com/java/java-7-forkjoin-framework.html

Russell & Norvig Introduction to Artificial Intelligence a Modern Approach, 3rd Edition

Singth, N. (2017). Artificial Neural Networks, Neural Networks Applications and Algorithms. Retrieved from https://www.xenonstack.com/blog/data-science/artificial-neural-networks-applications-algorithms/

Wegner, Brittany (N.A.) How to make a Neural Network in your bedroom, TED x CERN Talks http://www.youtube.com/watch?v=n-YbJi4EPxc last visited 11 July 2014