Synchronous Multithreaded Mini-Batch Gradient Descent

Alexios Spiliotopoulos, Fanourios-Stylianos Psathopoulos
Department of Informatics and Telecommunications
University of Athens
`{sdi1700147,sdi1400320}@di.uoa.gr`

Abstract

The main purpose of project was (1)to develop an efficient machine learning application, in terms of speed and memory management, that can produce satisfying results for a given dataset and (2)present the differences in training-testing execution time according to the number of threads running. The application provides clean interfaces for a *Logistic Regression* classifier, a *Vectorizer* used for transforming data into vectors and a *thread scheduler* to handle the synchronization. Lastly, we provide an alternative method of training; *Iterative Training.*

1 Introduction

1.1 Installation and Application Options

To get the file, open a terminal and use
`git clone https://github.com/aspil/Information-Systems-Project`

For compilation and execution go to *Project 3* directory.

Compile the application using `make` or `make VERBOSE=-DSHOWPROGRESS` to see a more verbose version with progress percentage on some procedures.

Below is the app's help page. At the end is an example execution:

USAGE

./bin/app -m MODE -i DATAPATH -r RELATIONSFILE -f FEATURES -l RATE -t NUMTHREADS -b BATCHSIZE -e EPOCHS -s STRATIFY [-d DEBUG]

DISCLAIMERS

Before running the program with MODE=test, ensure that you have already ran it with MODE=train. Otherwis,e the execution will fail. Also, make sure to use the same STRATIFY value for both train and test executions!

If MODE=test |validate, only provide -m, -i, -r, -t, -e and -s options. The rest (valid ones) will be ignored.

OPTIONS

-m MODE, –mode=MODE the mode of the app that will be executed, MODE is one of train, test, validate, iterative.

-i DATAPATH, –input_data=DATAPATH
DATAPATH is the path to the json files that the app needs to parse.

-r RELATIONSFILE, –relations=RELATIONSFILE
RELATIONSFILE is the csv file's name that contains all the labels.

-f FEATURES, –max_features=FEATURES
FEATURES(int) is the number of features that will be used to reduce the dimensionality of the word vectors used by the model.

-l RATE, –learning_rate=RATE
Provide learning rate into RATE(double) to be used in gradient descent.

-t NUMTHREADS, –nthreads=NUMTHREADS
The number of threads to parallelize some procedures.

-b BATCHSIZE, –batchsize=BATCHSIZE
The size of the sample batches in gradient descent.

-e EPOCHS, –epochs=EPOCHS
The number of gradient descent iterations.

-s STRATIFY, –stratify=STRATIFY
If STRATIFY=no, use the whole dataset for training. Else, use same amount of each binary labels

-d DEBUG, –debug=DEBUG
Controls whether the program's output will contain results for the cliques while parsing the initial labels. DEBUG is one of positive,negative,all.

EXAMPLE

```
./bin/app -m train -i data/ -r data/initial_labels/sigmod_large_labelled_
dataset.csv -f 1500 -l 0.05 -t 50 -b 512 -e 5 --stratify=yes
```

Train the model with data from data/camera_specs/ directory, labels from data/initial_labels/sigmod_large_labelled_dataset.csv, reduce the dataset's features down to 1500 and perform mini-batch gradient descent with learning rate = 0.001, batch size = 512 and 50 threads for 5 iterations. Also stratify the training dataset.

```
./bin/app -m test -i data/ -r data/initial_labels/sigmod_large_labelled_
dataset.csv -t 10 -b 512 --stratify=yes
```

Test the model trained above, using 10 threads and batch size = 512.

To compile the unit testing, do `make test` and execute using `./bin/test`

Remove all object files and executables using `make clean`.

1.2 System Environment

OS	Linux
Distro	Ubuntu 18.04
CPU	Intel Core i5-7400 4x3.00GHz 64-bit
Memory	8 GB
Graphics	NVIDIA GeForce GTX 1050
SSD	128GB

Table 1: System specifications

2 Remarkable Modules

2.1 Logistic Regressor

The core of the program. This structure is used throughout the program with a useful API for a thread-safe training/testing procedure. The classifier internally handles converting the datasets to word vectors and updates the weights of the model.

The basic API methods are:

- Logistic_Regression_fit
- Logistic_Regression_Init
- train
- predict
- resolve_transitivity_issues
- Logistic_Regression_Delete

The rest are internal and therefore not mentioned.

2.2 Tfidf Vectorizer

This structure handles the conversion of the dataset to word vectors. The dataset is stored with its respective vectors internally inside a hashtable for fast lookup. The feature reduction is done using a priority queue and extracting the words with the highest average tfidf value.

The basic API methods are:

- vectorizer_init
- vectorizer_fit_transform
- vectorizer_get_vector
  The rest are internal and therefore not mentioned.

2.3 Scheduler

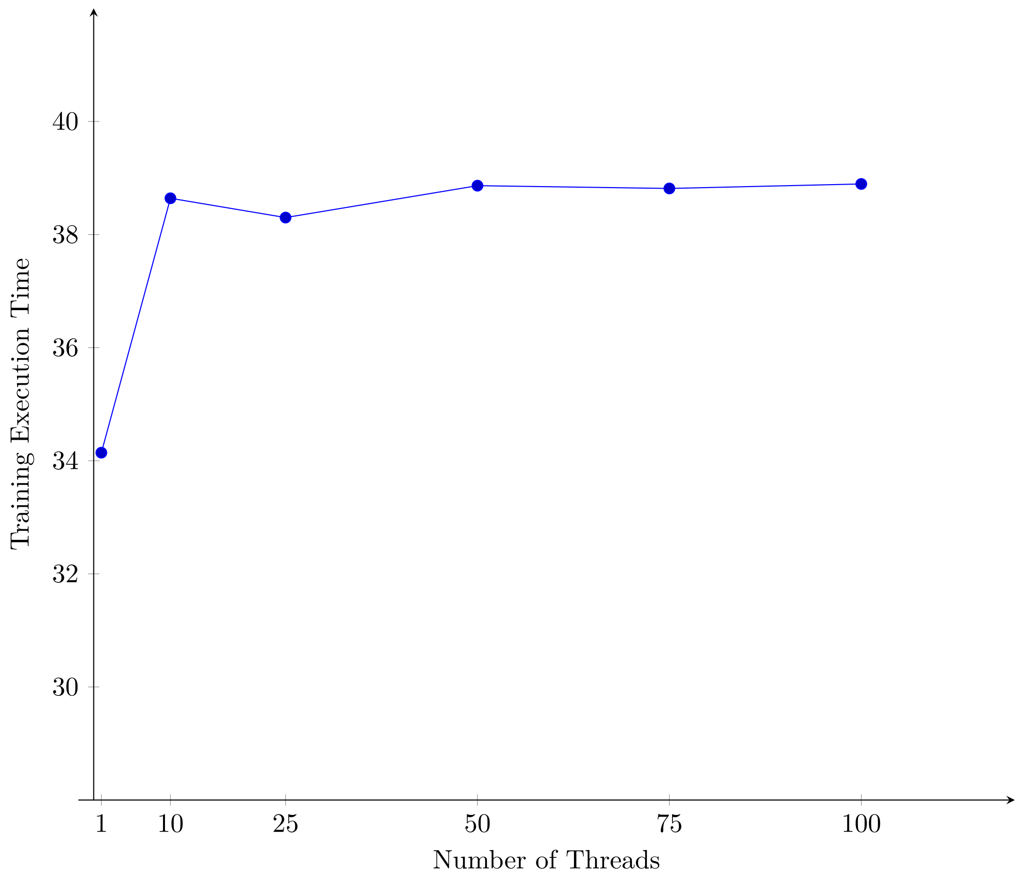
The scheduler uses an array of pthread structures as a thread pool and a simple queue to store new jobs. The scheduler is used only on training and testing procedures.

The main thread, creates and submits new jobs (a batch of the dataset) into the scheduler's queue and waits for the whole procedure(thread workers computing the gradients and a specific thread computing the model's weights) to finish before submitting new batches.

We experimented by letting the main thread run in parallel with the worker threads and didn't show any improvements in terms of execution time.

3 Experiments and Comments

1. We were expecting to get more accurate predictions with the mini-batch gradiend descent algorithm. But for some reason, maybe, an algorithm misunderstanding, the predictions were stuck in a range between $[0.45, 0.55]$. It was even more surprising when we tested the saved weights in the data from which the model was trained. The results of the predictions were again in between the same range which makes no sense.
2. The use of parallel computing didn't show any dramatic improvement in terms of execution time. It was obvious that this situation will occur because a number of threads was racing with each other on shared variables so threads had to be blocked, and the fact that the main thread blocked before submitting a new batch set, all led to an overall increase in execution time.
3. Last, but not least, if we had to choose between our implementations on stochastic gradient descent and mini-batch with the reservation that no mistakes were made in computing, we would choose the first one in this specific dataset.


Training Execution Time
40
38
36
34
32
30
1
10
25
50
75
100
Number of Threads