Bio - Intelligent Algorithms - Exercise 1

Table of Contents

[Installation 2](#_Toc73612568)

[General Prerequisites 2](#_Toc73612569)

[CPU Accelerated (NumPy) 2](#_Toc73612570)

[Prerequisites 2](#_Toc73612571)

[GPU Accelerated (CuPy) 2](#_Toc73612572)

[Prerequisites 2](#_Toc73612573)

[Models location 3](#_Toc73612574)

[Features Highlights 3](#_Toc73612575)

[Implementation 3](#_Toc73612576)

[Application CLI Usage 4](#_Toc73612577)

[Common Usage Options 5](#_Toc73612578)

[Fit 5](#_Toc73612579)

[Predict 5](#_Toc73612580)

[Compute accuracy 6](#_Toc73612581)

[Run time performance 6](#_Toc73612582)

[Results 6](#_Toc73612583)

[System information 6](#_Toc73612584)

[OS 6](#_Toc73612585)

[CPU 7](#_Toc73612586)

[GPU 7](#_Toc73612587)

# Installation

This section gives the instructions and prerequisites to run the model locally.

We supply both CPU vectorized accelerated implementation and **GPU vectorized accelerated** implementation. The app will run by default with GPU acceleration, if the *CuPy* module is available, and fallback to CPU acceleration otherwise.

In addition, this could be controlled statically by setting the *lib* variable at the top of the application code. See the doc string in the first few lines of the code.

## General Prerequisites

* Install Python:
  + [Python](https://python.org/): v3.6.0+ / v3.7.0+ / v3.8.0+ / v3.9.0+
* Install Python external modules:
  + pip install -U setuptools pip
  + pip install pandas matplotlib numpy

## CPU Accelerated (NumPy)

The CPU accelerated version uses *Python NumPy* vectorized implementation.

### Prerequisites

No need to install additional components, general prerequisites from previous section are sufficient.

## GPU Accelerated (CuPy)

The NVIDIA GPU accelerated version uses *Python CuPy* GPU vectorized implementation.

### Prerequisites

* Install CuPy:
  + All the details regarding CuPy installation can be seen in the link below as well:
    - [Installation — CuPy 10.0.0a1 documentation](https://docs.cupy.dev/en/latest/install.html)
  + NVIDIA GPU is a requirement, see in the link below the compatibility.
    - [NVIDIA CUDA GPU](https://developer.nvidia.com/cuda-gpus) with the Compute Capability 3.0 or larger.
  + Install CUDA toolkit:
    - [CUDA Toolkit](https://developer.nvidia.com/cuda-toolkit): v9.2 / v10.0 / v10.1 / v10.2 / v11.0 / v11.1 / v11.2
      * If you have multiple versions of CUDA Toolkit installed, CuPy will automatically choose one of the CUDA installations. See [Working with Custom CUDA Installation](https://docs.cupy.dev/en/stable/install.html#install-cuda) for details.
  + Install Python CuPy lib:
    - Wheels (precompiled binary packages) are available for Linux (x86\_64) and Windows (amd64). Package names are different depending on your CUDA Toolkit version.

| **CUDA** | **Command** |
| --- | --- |
| v9.2 | $ pip install cupy-cuda92 |
| v10.0 | $ pip install cupy-cuda100 |
| v10.1 | $ pip install cupy-cuda101 |
| v10.2 | $ pip install cupy-cuda102 |
| v11.0 | $ pip install cupy-cuda110 |
| v11.1 | $ pip install cupy-cuda111 |
| v11.2 | $ pip install cupy-cuda112 |

# Models location

Our model *Pickle* dump files, both CuPy model and NumPy model weight about 55 MB each. They can be downloaded from here:

# Features Highlights

* GPU and CPU accelerated implementations
* Generic application for any kind of data
* All the features are controllable through the CLI
* 2 hidden layers controllable architecture
* Various activation functions: sigmoid, relu and leaky\_relu
* Fitting a generic model
* Predict from a generic model
* Ability to dump and load fitted models
* KFold cross validation
* Mini Batch Gradient Descent
* Input noise regularization
* Dropout regularization
* L2 regularization
* Static learning rate
* Learning rate decay
* Early stop
* Gaussian weights initialization
* Z-Score normalization
* Plotting accuracy and loss

# Implementation

We have implemented a generic Multi Class Neural Network Python application, which gives the user the ability to run the model and control its various features for different kind of datasets from the CLI.

We have the support for 2 accelerated implementations:

* GPU accelerated using vectorized implementation with *CuPy*
* CPU accelerated using vectorized implementation with *NumPy*

See *Installation* section for more details regarding installation.

See *Run Time Performance* section for more details regarding the performance of the implementations.

### Application CLI Usage

usage: **accelerated\_model.py** [-h] [-f] [-p] [-kf K\_FOLD] [-d DUMP\_PATH] [-l LOAD\_PATH] [-q] [-z] [-k TRAIN\_FILE\_PATH] [-j VALIDATION\_FILE\_PATH]  
 [-x TEST\_FILE\_PATH] [-y PREDICTION\_FILE\_PATH] [-t LABELS\_FILE\_PATH] [-i] [-a {sigmoid,relu,leaky\_relu}]  
 [-l1 L1\_HIDDEN\_SIZE] [-l2 L2\_HIDDEN\_SIZE] [-c NUM\_OF\_CLASSES] [-g LR] [-ir INITIAL\_LR] [-e EPOCHS] [-b BATCH\_SIZE]  
 [-r REG] [-n INPUT\_NOISE\_P] [-u DROPOUT\_P] [-s EARLY\_STOP\_MAX\_EPOCHS] [-o] [-m INIT\_WEIGHTS\_MU] [-w INIT\_WEIGHTS\_SIGMA]  
  
Multi Class Neural Network Classification App. NVIDIA GPU acceleration supported when CuPy module installed.  
  
optional arguments:  
 **-h, --help** show this help message and exit  
 **-f, --fit** Run model training with the supplied parameters, default: False. This option dumps the model as a Pickle file by default named:  
 'model\_dump.bin' to the current working directory. This can be controlled by '-d','--dump-path' option.  
 **-p, --predict** Run prediction on the trained model, use '-x', '--test-file-path' to choose the test file, default: False.  
 **-kf K\_FOLD, --k-fold K\_FOLD**  
 Run KFold cross validation using the supplied K, default: Not set.  
 **-d DUMP\_PATH, --dump-path DUMP\_PATH**  
 Dump the model into the supplied full path as Pickle serialized file, default: model\_dump.bin.  
 **-l LOAD\_PATH, --load-path LOAD\_PATH**  
 Load the model from the supplied full path Pickle serialized file.Note: The module used should be compitable with CuPy/NumPy on  
 the current runtime. See the note at the top of application code.  
 **-q, --quiet** Don't print detailed information during the run, default: False.  
 **-z, --plot** Plot accuracy and loss graphs, default: False.  
 **-k TRAIN\_FILE\_PATH, --train-file-path TRAIN\_FILE\_PATH**  
 Full path to a CSV file of the training dataset, default: train.csv. File format: Labale, feature\_1, feature\_2, ... , feature\_n  
 **-j VALIDATION\_FILE\_PATH, --validation-file-path VALIDATION\_FILE\_PATH**  
 Full path to a CSV file of the validation dataset, default: validate.csv. File format: Labale, feature\_1, feature\_2, ... ,  
 feature\_n  
 **-x TEST\_FILE\_PATH, --test-file-path TEST\_FILE\_PATH**  
 Full path to a CSV file of the test dataset, default: test.csv. File format: ?, feature\_1, feature\_2, ... , feature\_n  
 **-y PREDICTION\_FILE\_PATH, --prediction-file-path PREDICTION\_FILE\_PATH**  
 Full path to the location where to save the prediction file, default: prediction.txt.  
 **-t LABELS\_FILE\_PATH, --labels-file-path LABELS\_FILE\_PATH**  
 Full path to the location of the labels file for computing accuracy, default: labels.txt. File format: Label in each row  
 **-i, --compute-accuracy**  
 Compute accuracy of given predictions and labels files, use with '-y', '--prediction-file-path' and '-t', '--labels-file-path'  
 options, default: False.  
 **-a {sigmoid,relu,leaky\_relu}, --activation-func {sigmoid,relu,leaky\_relu}**  
 Activation function name, default: relu.  
 **-l1 L1\_HIDDEN\_SIZE, --l1-hidden-size L1\_HIDDEN\_SIZE**  
 Number of neurons in the first hidden layer, default: 1500.  
 **-l2 L2\_HIDDEN\_SIZE, --l2-hidden-size L2\_HIDDEN\_SIZE**  
 Number of neurons in the second hidden layer, default: 1500.  
 **-c NUM\_OF\_CLASSES, --num-of-classes NUM\_OF\_CLASSES**  
 Number of categorial class of the data, default: 10.  
 **-g LR, --lr LR** Learning rate for the optimization algorithm, default: 0.  
 **-ir INITIAL\_LR, --initial-lr INITIAL\_LR**  
 Initial learning rate for the optimization algorithm, default: 0.1. This feature has higher priority than the static lr feature.  
  **-e EPOCHS, --epochs EPOCHS**  
 Max number of epochs to do, the algorithm might stop before, due to early stop, default: 300.  
 **-b BATCH\_SIZE, --batch-size BATCH\_SIZE** Batch size to use for the Mini-Batch Gradient Descent, default: 65.  
 **-r REG, --reg REG** Regularization factor, default: 0.1.  
 **-n INPUT\_NOISE\_P, --input-noise-p INPUT\_NOISE\_P**  
 The probability of non-active input features, default: 0.  
 **-u DROPOUT\_P, --dropout-p DROPOUT\_P** Dropout probability of non-active neurons, default: 0.2.  
 **-s EARLY\_STOP\_MAX\_EPOCHS, --early-stop-max-epochs** EARLY\_STOP\_MAX\_EPOCHS  
 Maximum number of epochs without substantial improvement in model accuracy before stopping the training, default: 18.  
 **-o, --input-z-score-normalization**  
 Controls whether Z-Score Normalization on the input dataset on or off, default: Off.  
 **-m INIT\_WEIGHTS\_MU, --init-weights-mu INIT\_WEIGHTS\_MU** Expectation of the normal distribution for model weights initialization, default: 0.  
 **-w INIT\_WEIGHTS\_SIGMA, --init-weights-sigma INIT\_WEIGHTS\_SIGMA**  
 Standard deviation of the normal distribution for model weights initialization, default: 0.01.

# Common Usage Options

Full usage options are available in *Implementation* section. Below are examples of common usage options. The examples below could be extended with additional arguments the application supports.

## Fit

Fit the model using the default parameters and dump a Pickle model file for future use to the designated path.

terminal> python accelerated\_model.py --fit --train-file-path <TRAIN\_FILE\_PATH> --validation-file-path <VALIDATION\_FILE\_PATH> --dump-path <DUMP\_FILE\_PATH>

## Predict

Predict test dataset using a pre-trained model and write the predictions to a file.

terminal> python accelerated\_model.py --load-path <MODEL\_DUMP\_FILE\_PATH> --predict --test-file-path <TEST\_FILE\_PATH> --prediction-file-path <PREDICTION\_FILE\_PATH>

## Compute accuracy

Compute the accuracy of given predictions and labels files and print it.

terminal> python accelerated\_model.py --compute-accuracy --prediction-file-path <PREDICTION\_FILE\_PATH> --labels-file-path <LABEL\_FILE\_PATH>

# Run time performance

We have used NVIDIA GPU accelerated library for mathematical vectorized operations on a GPU, in addition to the CPU accelerated vectorized implementation. As expected, the GPU acceleration is much faster! The difference is **~18,820 %** in run time.

## Results

**18,819.28 %**

18,819.27 %

## System information

We have used *Google Colab* platform for development and testing.

If needed, we can also supply a *Google* *Colab* Notebook with the implementation.

The details below describe our development setup.

## OS

* Linux 18.04.5 LTS (Bionic Beaver)

## CPU

* Architecture: x86\_64
* CPU op-mode(s): 32-bit, 64-bit
* Byte Order: Little Endian
* CPU(s): 4
* On-line CPU(s) list: 0-3
* Thread(s) per core: 2
* Core(s) per socket: 2
* Socket(s): 1
* NUMA node(s): 1
* Vendor ID: GenuineIntel
* CPU family: 6
* Model: 85
* Model name: Intel(R) Xeon(R) CPU @ 2.00GHz
* Stepping: 3
* CPU MHz: 2000.168
* BogoMIPS: 4000.33
* Hypervisor vendor: KVM
* Virtualization type: full
* L1d cache: 32K
* L1i cache: 32K
* L2 cache: 1024K
* L3 cache: 39424K
* NUMA node0 CPU(s): 0-3
* Flags: fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ss ht syscall nx pdpe1gb rdtscp lm constant\_tsc rep\_good nopl xtopology nonstop\_tsc cpuid tsc\_known\_freq pni pclmulqdq ssse3 fma cx16 pcid sse4\_1 sse4\_2 x2apic movbe popcnt aes xsave avx f16c rdrand hypervisor lahf\_lm abm 3dnowprefetch invpcid\_single ssbd ibrs ibpb stibp fsgsbase tsc\_adjust bmi1 hle avx2 smep bmi2 erms invpcid rtm mpx avx512f avx512dq rdseed adx smap clflushopt clwb avx512cd avx512bw avx512vl xsaveopt xsavec xgetbv1 xsaves arat md\_clear arch\_capabilities

## GPU

* Card: NVIDIA Corporation GV100GL [Tesla V100 SXM2 16GB] (rev a1)
* Driver Version: 460.32.03
* CUDA Version: 11.2