User specification and manual

Github repo: https://github.com/lichterberci/ProgramozasAlapjai/tree/main/nagyhf

Summary

This is a program that solves the MNIST problem, which means it can classify 28x28 pixel greyscale images into 10 classes, based on the digits present on the images. In other words, it can recognize eg. the number 7 on an image, which is 28x28 and greyscale.

The dataset

The MNIST dataset is an open-source dataset, consisting of 60000 training images and 10000 images for testing. It can be downloaded from this website.

Solution

The program uses a (fully-connected) deep neural network. The model's architecture can be selected using command-line arguments (or automatically recognized when reading a model from a file).

How to use it

The program's behaviour can be specified using cmd parameters. The following section will list all the possible parameters and how to use them. The order of the arguments doesn't matter, as long as every non-flag argument (those that start with a simple '-' as opposed to '–') is followed by its value(s). So `--train --show-images` is the same as `--show-images --train`, but `--train -data-folder ./data/` is not the same as `-data-folder --train ./data/`.

Specify folder of the data

By setting the `-data-folder` parameter, followed by a path to the folder, we can specify where the data is located. **It is important not to change the file names!** The file names should be: - train-images.idx3-ubyte - train-labels.idx1-ubyte - t10k-images.idx3-ubyte - t10k-labels.idx1-ubyte These files can be downloaded from this website.

Training a model

The user can train the model by running the program with the `--train` flag.
If we want to define a new model, its architecture can be specified by the `-model` option, followed by one or more `-layer` arguments. Every layer keyword should be followed by the number of neurons in the given layer and its activation function. Eg.: `-model -layer 800 RELU -layer 400 SIGMOID` will create a model with 2 hidden-layers: one with 800 neurons and ReLU as the activation

function and one with 400 neurons and the sigmoid function as its activation function. Other hyperparameters can also be specified using cmd arguments:

- `-num-epochs` defines for how many epochs should the model be trained. Eg.: `-num-epochs 3`
- `-learning-rate` defines the learning rate of the model. It should be in the form of xe-y where x and y are (positive) integers. Eg.: `-learning-rate 2e-5`

Loading a previously saved model

A model can be loaded using the `-load` parameter, followed by the path of the .model file. If a `-load` parameter is present, manually set model layout will be ignored!

Saving the model

A model can be saved with the `-save` parameter, followed by the path of a .model file. We can also add the `--save-continuously` option, which will save it every 1000 iterations (as opposed to only saving at the end.

Measuring the accuracy of a model

We can do that by using the `--test-accuracy` flag.

Show the result with images side-by-side

By setting the `--show-images` flag, the program will print the images of the test dataset with the given model's predictions beneath it. With the `--only-bad-images` flag, it will only show images, where the model is wrong.

Reproducability

The seed can be set with the `-seed` option with the seed after it as an integer. The default seed is 0, so even if there is no seed argument present, the results can be reproduced.

Examples

These assume that mnist is the name of the .exe file created locally.

Train a new model and save it `mnist --train -save ./models/example.model -model -layer 500 relu -layer 100 sigmoid -num-epochs 3 -learning-rate 1e-7`

Load an existing model and test its accuracy `mnist -load ./models/example.model --test-accuracy`

Train an existing model for 2 more epochs `mnist -load ./models/example1.model --train -num-epochs 2 -learning-rate 1e-8 -save ./models/example2.model`