# NeuralPy

Github: https://github.com/teamsoccult/NeuralPy

## 1. Introduction to the structure of the code

We decided early that we wanted to document the code as well as possible, and then upload the whole project to github. We opted for an overall structure in which tasks of similar nature were grouped into different files. The functions in these files can then be loaded as modules and used in other documents. This is useful in order to avoid having one huge and messy file. In addition, other projects could potentially use only the modules relevant to them. We will now provide a quick overview of the file-structure before discussing design choices, plotting and testing.

The partitioning of tasks resulted in six files: read_write_helper.py, math_helper.py, plots_helper.py, network_helper.py, tests.py, learn.py.

The "helper"-files reflect our attempt at partitioning the tasks into coherent blocks. These form the core of the project and constitute the modules referred to earlier. "read_write_helper.py" handles tasks B, C, F and G. The functions in this module all have to do with reading, writing and converting files. "math_helper.py" handles tasks H, I, J, K, L and M. It contains functions which deal with linear algebra and math. It basically contains functions that one would usually turn to the numpy library for. All subsequent files rely on this module. We started on task S (optional) but did not quite finish it. That is also in here. "plots_helper.py" handles tasks D, N and O, which are the plotting related tasks. It contains functions which plot images of digits as well as visualize the weights of a neural network as heatplots. "network_helper.py" handles tasks P, Q, R and more. This module contains functions more directly related to updating, training and testing neural networks. (NB: there might be optionals here).

"test.py" does not handle any specific tasks. It tests the assumptions of the functions from previous modules and showcases functionality. We will say more about this in the section on testing. "learn.py" does not handle any specific tasks either. It uses the functions of previous modules to train an initially random neural network on the training dataset from MNIST. This network is then saved, and is then validated against the test dataset from MNIST. In addition, the plots displayed in this report are generated here.

A quick note on documentation. We wrote extensive docstrings for all the functions we made in this project. These can be found in the respective files, as specified above. We will therefore not discuss the details of all functions here and instead refer the reader to the docstrings.

## 2. Design choices

### 2.1. General considerations

The most characteristic design choices for the project were made in the "math_helper.py" module. This module is the backbone of the project upon which all the other parts stand. The reason for this is twofold. The first being that all subsequent files use the functions from

this module. The second being that we need to establish how we represent matrices.

### 2.2. Object oriented vs functional programming

Initially, we implemented an object-oriented solution to the problem, by making a class "Matrix", which contained all the functionality we needed concerning matrices. The issue we encountered was that while most functions lended themselves nicely to an object oriented solution, not all of them did. While it is definitely possible to pursue this strategy, we instead shifted to making functions instead. This allowed us to gather everything related to matrices and math more generally in one module. This seemed more harmonious to us.

### 2.3. Representing a matrix

Another consideration we had was whether we wanted to conceptualize a matrix in the form, $L[r][c]$ or $L[c][r]$, where $L$ is a list of lists, $c$ refers to columns and $r$ refers to rows. We changed back and forth, and ended up choosing the first option, $L[r][c]$. This choice was made on the basis that the format matched the files that were provided as well as matching the general notation of indexing from linear algebra.

In more conceptual terms, the interpretation of lists as matrices is something we had difficulties wrapping our minds around. The core issue is that the dimension of a list is different from the dimensions of a matrix. We became aware of this issue when we realized that we had trouble conveying this during the write-up of docstrings. When possible, we tried to convey what the word "dimension" referred to in the docstrings explicitly i.e. "the dimension of the list" or "the number of columns and rows of the matrix".

## 3. Dimensions

The key assumption in the operations of linear algebra concerns how two matrices align in terms of number of rows and columns. To this end, we found the function "dim_recursive", which recursively finds the dimensions of a list. With two-dimensional lists this will always return two values corresponding to rows and columns. However, when testing this function we realized that this function was not sufficient in itself, because it only returns one value for inputs that are row vectors (one-dimensional lists) while we need it to output both rows and columns.

To generalize the function, we initially sought to make a decorator function to handle the special case of a one-dimensional list. However, the decorator function is called each time a recursive call is made, so this did not work out. We therefore settled on making a new function, "dim" which uses "dim_recursive" and solves the issue.

## 4. Speed is the issue

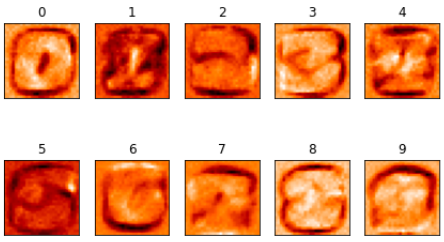
The culmination of all the functions of this project is arguably the "learn"-function. While we did solve the task and have tested that this function works as intended, we realized (too late) that the "learn"-function took an absurd amount of time to run. We estimated that to run five epochs with a batch size of 100 images (as is stated in the assignment) would take almost 150 hours, which was too much.

By timing the different parts of the function, we deduced that the computation that took time was not the complex matrix products, but calculating the accuracy of the network. For this reason, we made a new function, "fast_learn". This function is almost identical to the original "learn"-function, except for the key difference that it does not calculate the accuracy of any network during the

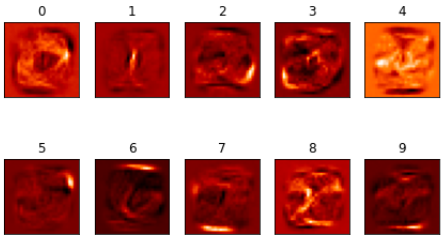
learning process. This also means that it is impossible to save the best performing network, as we do not calculate how accurate the different networks are. Instead, "fast_learn" only saves the last network of the learning phase. Although this is not optimal, the function allowed us to save immense amounts of time, and validation against test data (MNIST) shows that it reaches an accuracy of 86% which we count as a success.

## 5. Plotting

Figure 1: **weights comparison**




(a) weights provided in task




(b) our network

The outputs of our plotting functions resemble the example pictures in the project description. The main focus for us here was not to make the plots more fancy, but rather to make them flexible and useful. The first plotting associated task (task D) will serve as an example of our approach. The task asks us to create the function with two arguments, images and labels, and to show the first few images with an appropriate color map. The function we created can indeed be executed with only these two arguments, as they are the only required arguments of the function. When only these two arguments are specified the function will display the first ten images arranged in a two by five grid of subplots. Having done this however, we wanted to provide the user with more flexibility. Specifically we wanted the user to be able to control how many images (and which ones) they wanted displayed, and to have some control over the way the subplots of the function are organized in the output. To achieve this we gave the function two optional arguments which allow the user to specify which images to use and to control some of the aesthetics. For more information we refer the reader to the docstring.

## 6. Testing

We were especially keen on testing the code properly, because a lack of testing has (in previous projects) proved to be a continuous nuisance. One issue is that code can run without returning errors, even though they are not operating as intended. To alleviate this issue we wanted to make sure that the functions which only make sense when the input is meaningful do not produce nonsensical output when these assumptions are not met. Instead we want them to give the user an error. For this we implemented exceptions when we deemed it appropriate. We checked that the functions work with proper input, and return errors with improper input in the "test.py" document. Because we intentionally specify inputs to the functions which violate their assumptions, this document returns several errors (mainly ValueError). This is on purpose.

Besides implementing exceptions into our code to test that our functions work as intended, we also used doctest to test many of our functions - and to show the user

examples of usage. This implementation was especially smooth and appropriate in the "math_helper.py" document, because all the functions there revolve around mathematical operations in which it is easy to specify what the output should be. It was less clear for us how to use this approach for the functions related to plotting, as their output is different from a list or a number. The testing we did of our plotting functions consisted in trying to feed them different input and verify that they worked as intended. Examples of the functionality of our plotting functions can be found in "test.py".