Bio Intelligent Algorithms – Exercise 1

Orr Moran, 205948425
Amit Amar, 316534825

Validation Score

The model we used to assign the predictions for the given test set scored 41.9% accuracy on the validation set.

Implementation Process

Since implementing a neural network may be a tricky task with a lot of pitfalls, we decided to first use a “state of the art” ANN library with the given data to have an idea about what accuracy we should achieve in our implementation.
We tried several architectures and activation functions and got around 40% of accuracy on both train and validation sets.

The only two libraries our implementation uses are NumPy (for efficient matrix and mathematical operations) and pandas (for reading the CSV file and converting its data to NumPy's format).

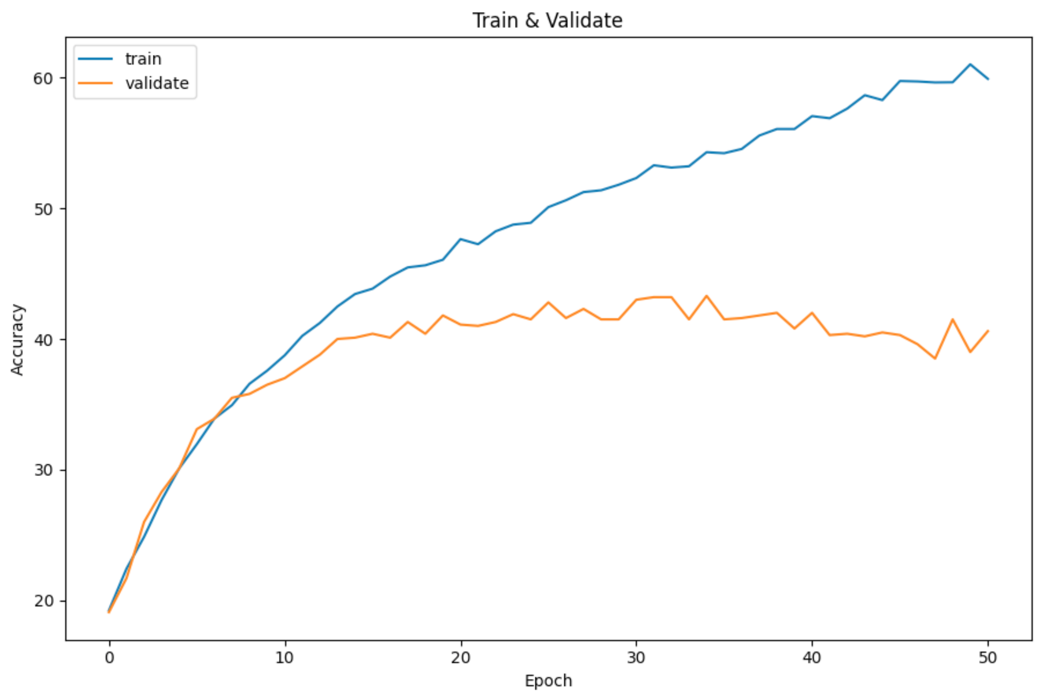
Our implementation consists of the ANN class, which exports an API for adding fully connected layers (given number of neuros and activation function), train using back propagation with a given classified data set, prediction of set of records and evaluation function to calculate the accuracy of the model given data set with its labels.

After we’ve finished implementing our ANN class, we started conducting experiments to calibrate and choose hyper-parameters and the network architecture.
We tried different architectures, some of them with more layer than others, we tried changing the number of neurons on each layer, and at the end the network architecture that gave us the best results is as follows:

Input 3072 -> Hidden 256 -> Hidden 128 -> Output 10.

We also had to face an early overfitting, which made our model to be stuck around 27% on validation set. To solve this issue, we implemented an input noise mechanism and conducted other experiments to determine the noise factor (we use 80% noise factor in our assignment).
Using input noise made us to lower our learning rate, to avoid “jumpy” training and to allow our model to converge. The learning rate we use in this assignment is 0.0005.
To make our results even better, we tried to use zero mean normalization on the data, and it indeed helped, so we’re using it with the validate and test data sets.

Throughout our work, we fixed the random seed of NumPy, so our experiments will be reproducible, and we saved the model after every epoch along with its accuracy score on both the train and validation sets. By doing so, we could look at the models after we finished the train process and decide which of them to use to avoid overfitting (by looking at the accuracies graph).



<u>Using the Code</u>

In order to train the model from scratch run main.py from src direcory:

```
python3 ./main.py
```

In order to generate a file with prediction given a saved model and a dataset, use the following command:

```
python3  ./predict.py <path_to_model_file> <dataset_path>
```

In order to evaluate the accuracy score of a saved model with a tagged dataset, use the following command.

```
python3 ./evaluate.py <path_to_model> <dataset_path>
```