Assignment 1 - Multilayer Perceptron

Group 75

Michał Jóźwik, Lukas Ostrovskis, Martynas Krupskis, Denis Tsvetkov

February 2022

1 Running the code

Open **Group_75_code** directory with PyCharm, inside you will find *main.py* file, which you can run through PyCharm. You can also run this code via command line using *python main.py* command (tested with Python 3.9.5) in the provided directory. You need to install numpy and matplotlib before running the code. The code in *main.py* will generate labels for data specified in **Group_75_code/data/unknown.txt**. The labels will be written to **Group_75_classes.txt**. Be aware that running *main.py* will overwrite **Group_75_classes.txt**.

2 Architecture

1. Graphs presenting the error over epochs of our single perceptron trying to learn the OR, AND and XOR functions.

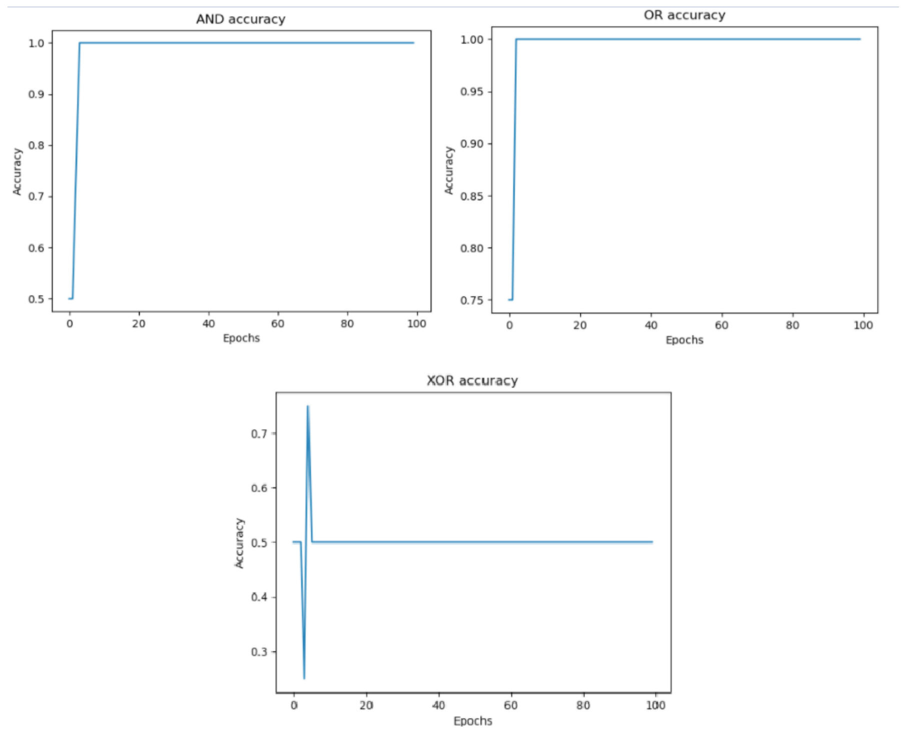

Figure 1: Single perceptron error for AND, OR and XOR functions

2. We need 10 input neurons, as we have 10 features, so the ANN takes in 10 arguments.

3. We need 7 output neurons, as we have 7 classes, so the ANN outputs probabilities for those 7 classes.

4. We will have 3 layers in total, as in the course book it is argued that most commercial applications require 3 layers with output and input, this means that we will have 1 hidden layer. For the architecture we chose to use 15 hidden neurons, this was an educated guess given the examples of ANN that we have seen in our lives, we noticed that the pattern is that the network grows from the input layer and then it starts shrinking towards the output layer. So a number above 10, but below 30 made sense.

5. We will use the *Sigmoid* function. We are aware that something like *Softmax* or *ReLU* could be used for specific layers, but as we had backpropagation figured out for the Sigmoid function in the lecture we decided to try that out.

6.

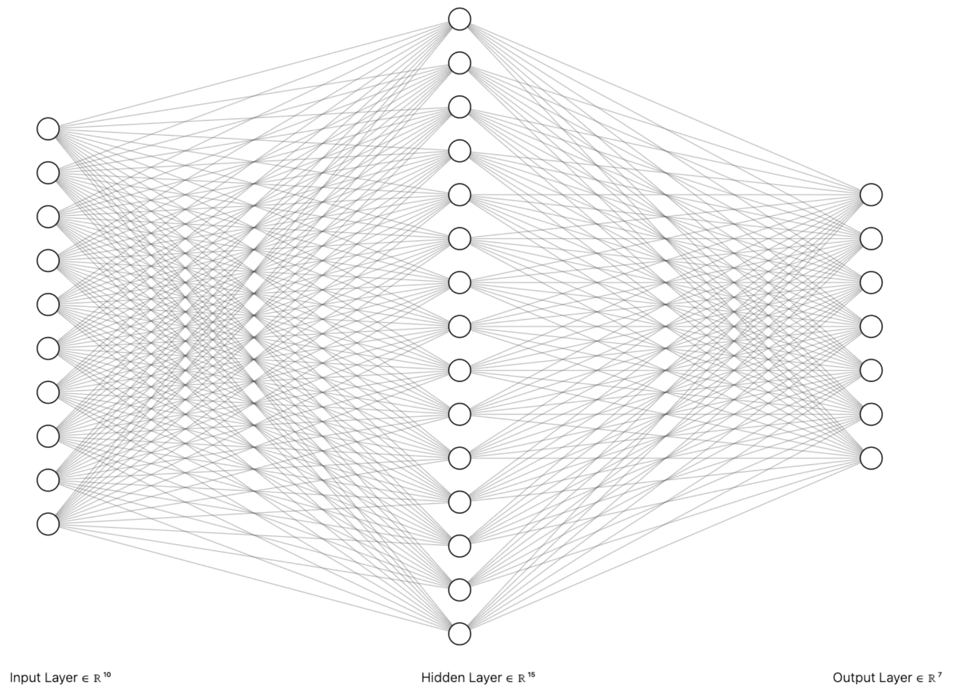

Figure 2: The architecture of our ANN model

3 Training

7. Test set - 10% — Validation set - 10% — Training set - 80%.
The reason for this split is that we don't have that much data, only 7854 entries. We do believe that 785 entries are enough to get the accuracy of the model and to tune the parameters.

8. We use a test set to evaluate the accuracy. To do that we get the labels and the predicted labels of the test set and then divide the number of correctly predicted labels by the total number of entries in the test set.

9. We have two stopping conditions, the first one is the number of **epochs**. The model will stop training after the model has been updated **epochs** times.
However, there is a problem with only using of epochs as a stopping condition, as no one is stopping you from assigning a really huge value to epochs parameter, meaning that training might not end in our lifetime. To tackle this problem we introduced two parameters **epsilon** and **stop_num**. The training will stop if accuracy does not improve over **stop_num** consecutive updates by some **epsilon** value.

10. We've trained our network 10 times, each time with different random initial weights. The difference in final accuracies is due to the random initialization of weights and biases. However we can observe that the delta between highest value and lowest value of accuracy is 0.02 meaning that our model is not that sensitive to the randomness of the weights.

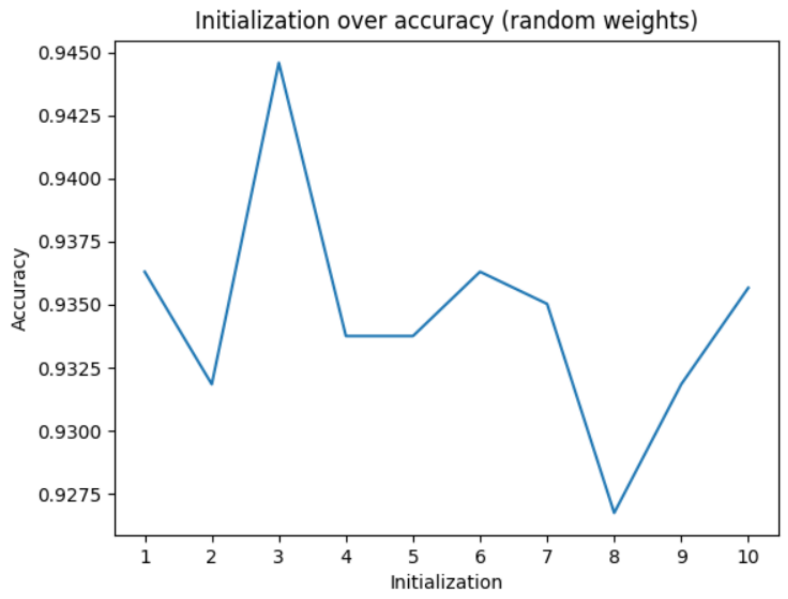

Figure 3: Initialization over accuracy

4 Optimization

11. We used cross-validation to measure the mean accuracy for our network with a different number of neurons in the hidden layer in the interval [10,20]. As we can observe in the generated plot below, the accuracy is the lowest with 10-11 neurons, seems to be optimal in the range [12,17], and starts decreasing again from 18 neurons. Lower accuracy with a lower number of neurons can be explained by underfitting - our model is not able to sufficiently fit to the given data and thus the accuracy is lower. On the other hand, lower accuracy with a high number of neurons can be explained by overfitting - our model overfits to the training data and fails to generalize to the test data.

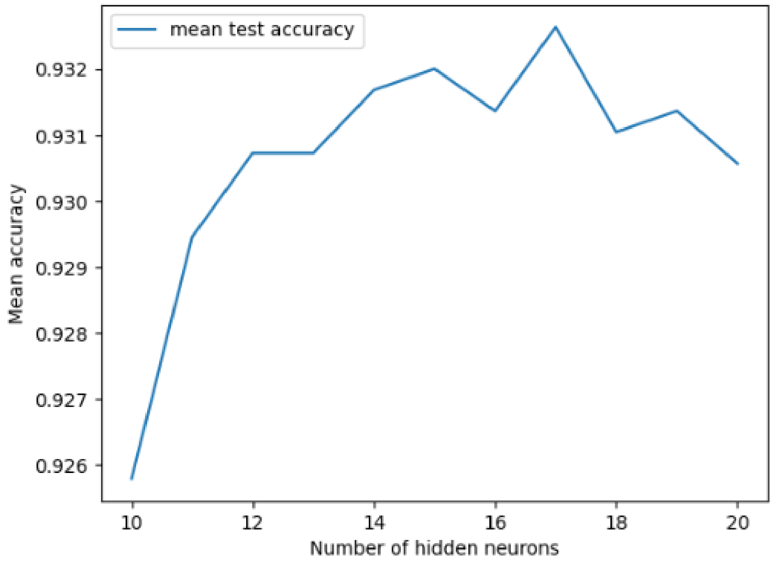

Figure 4: Mean accuracy for different number of hidden neurons

12. Here are the plots of the performance of the architecture (17 neurons) that we've chosen (both accuracy and train/test error plots). It's visible that the model accuracy is sufficiently high and the validation error is very similar to the training error, thus we can conclude that our model does not suffer from underfitting or overfitting and is an optimal choice overall, therefore this choice of architecture has the best result.

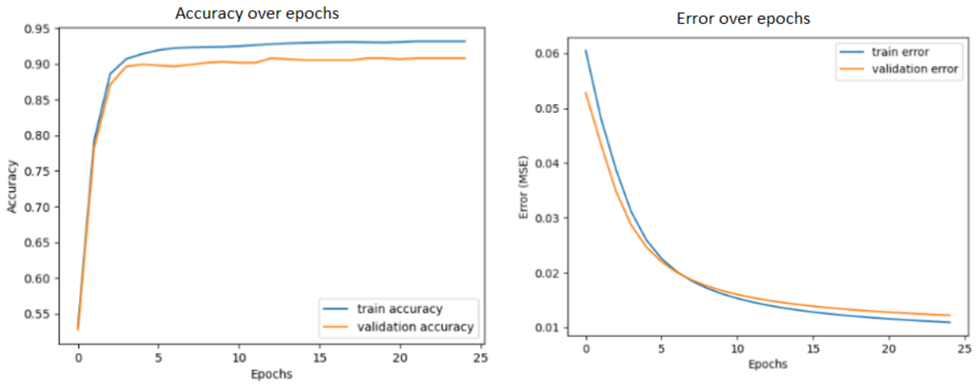

Figure 5: Performance of our architecture over epochs

5 Evaluation

13. The accuracy for the test set is on average 93.26%, but it fluctuates between 91% and 94% and the accuracy for validation is 92.5%. We consider such a small difference to be random, but the fact that the difference in the success rate is small means that our model does not overfit.

14. Confusion matrix is used to visually describe the performance of a classification model. In our case, it plots the 7 possible classes (y axis) against the 7 possible predictions (x axis). It can be plotted with just the number of times each class is predicted for every actual class but it can also be normalized and visualized in ratios,

thus making it easier to see where most of the mistakes are made. E.g. given class 1, our model predicted it correctly with 94% accuracy, but also gave some different predictions: 1% of them were class 3, 4% were class 4, 1% were class 6.
It is visible in the image above that our ANN performs worst on classes 3 and 6 - both obtaining 0.91 (91%) accuracy.

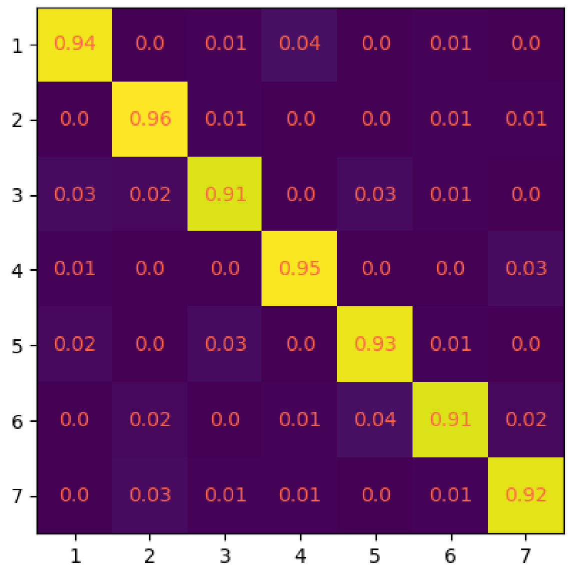

Figure 6: Confusion matrix

6 Scikit-learn

16. The values that were found by using the provided notebook were pretty close to what we have used ourselves. We were not trying all the possibilities ourselves, such as different activation functions, because we were already getting around 93.26% accuracy using the sigmoid function. The notebook suggested using around 20 hidden neurons and 0.01 learning rate. Our experiments had shown that something around 17 hidden neurons and 0.015 learning rate was optimal. The accuracy in the notebook was 93.0% meaning that the performance did not differ that much. The training behavior was also fairly similar, but the scikit-learn one had a steeper increase of accuracy reaching 90% in around 2-3 epochs, while our model took around 4-5 epochs to reach it.

17. The optimal hyper-parameters don't seem to influence the accuracy of the model by much. On average with optimal hyper-parameters we are getting 93.26% accuracy and with optimal hyper-parameters in the notebook we got 92.3%. The performance with scikit-learn optimal parameters seems to be worse and we believe that is due to the fact that the 93.26% accuracy represents the optimal hyper-parameters that our cross-validation was able to find, meaning that using any other parameters not equal to the optimal ones will have a lower performance.

7 Reflection

18. One example of misclassification that we think is very illustrative is email spam. The goal of this classification model is to determine which emails are spam and which are not. Considering that spam is a positive label, the false positive can be potentially dangerous and have harmful consequences. Missing an important email due to it ending up in spam can deprive people of work opportunities, job offers, etc. On the other hand, not marking an actual spam email as spam (false negative) can also have negative consequences, users might infect their devices with viruses or even become victims of scamming.

19. There are multiple ways to mitigate the harm done by unjust classification:

- We can try debiasing the data. E.g. if the size of one group is bigger than the size of another - try to make them equal. Removing proxies (correlated variables) can also help in mitigating unjust classification.
- Preferential sampling: From a discriminated group we could draw more examples with positive labels and from favored groups draw less examples with possible labels.
- We can try to create diversity within the actual team of developers working on the model. Developers with different backgrounds can have different data that's 'similar' to them, which in turn can help filter the data appropriately.

This, however, doesn't ensure that we'll completely solve the problem. It can be very hard or impossible to measure the actual bias and lack of fairness (e.g. how to treat similar individuals from different groups fairly and equally), thus, we can only hope to reduce it as much as possible, but not to solve it completely.