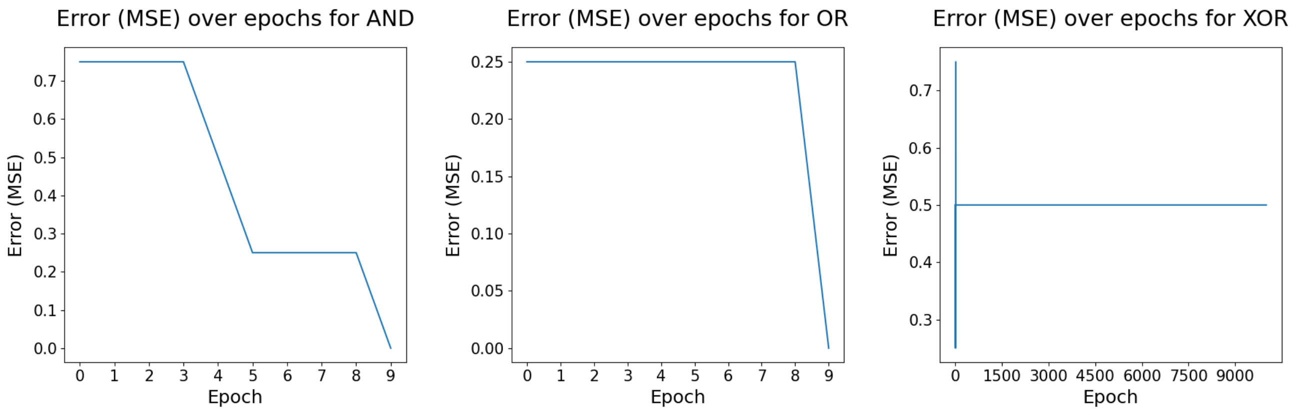
CI ANN Report - Group 66

Alto Delia, Amar Mesic, Mikkel Makela, Pandelis Symeonidis

March 2021

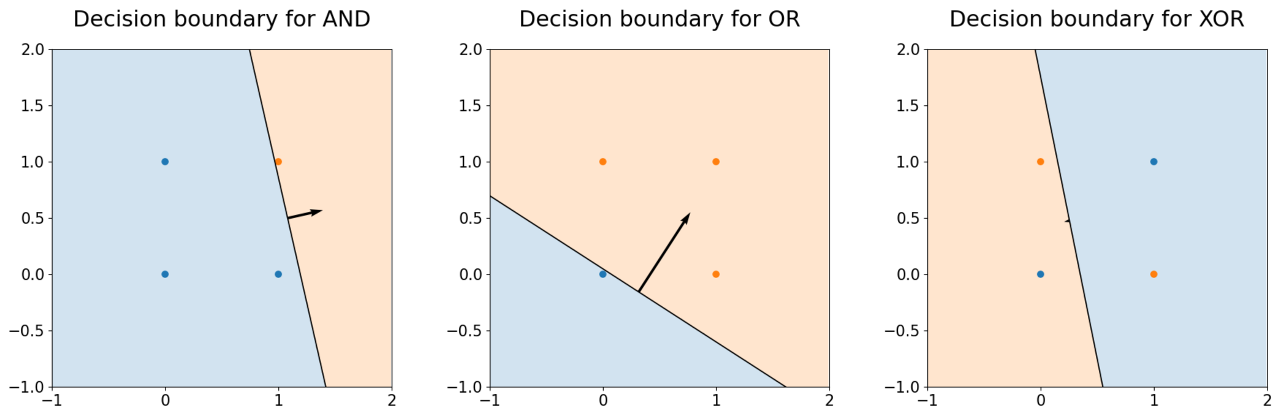
Architecture

1. The error vs. epoch graphs for training a perceptron on the AND, OR and XOR logic operations can be seen in figure 1 below. As you can see, the perceptron is able to quickly learn the AND and the XOR but does not converge for the XOR (reaches the max threshold of 10,000 epoch set). The error oscillates between 0.25 and 0.75 and then gets stuck in an infinite loop with an error 0.5. This is because the perceptron models a linear decision boundary whereas the XOR function would require a non-linear boundary.



(a) AND (b) OR (c) XOR

Figure 1: Perceptron learning. The error (MSE) is plotted against the epochs during training. As seen it doesn't converge for XOR



(a) AND (b) OR (c) XOR

Figure 2: Perceptron decision boundaries

2. The amount of input neurons depends on the input data. Since in this case our data comprises of 10 features, the network should also consist of 10 input neurons.

3. The amount of output neurons depends on the task at hand. In this case, we are doing multi class classification with 7 classes. Hence we are using 7 output neurons where neuron 1 represents class 1, neuron 2 represents class 2 etc. Each neuron will then represent the probability of the input fed through the network belonging to that class.

4. The amount of hidden layers and neurons is hard to determine. As far as the amount of layers is concerned, after doing research we concluded that using only one could be enough for our task. One hidden layer with enough neurons is enough to model any function due to the Universal Approximation Theorem [2]. Moreover, we found that for most problems it seems to be agreed that one hidden layer is enough. However, we will design our neural network in such a way to allow easy configuration of the number of hidden layers and neurons so that we can try out the performance of our network using multiple hidden layers as well.

   As far as the number of hidden neurons is concerned, this is also difficult to estimate and will be easier to determining using k-fold validation to see what amount performs best. However, as a starting value we decided for 8 neurons since it seems that a common heuristic is to take the mean of the amount of inputs and amount of outputs but this is subject to change as we train our network [1].

5. The networks activation functions will be as follows. The output layer neurons will have a SoftMax activation in order to convert the output into a probability distribution. This makes sense as each output neuron will model the posterior class probability. For the hidden layer we will start with the Sigmoid function to squish the values between 0 and 1 but we will also experiment with the hyperbolic tangent and the leaky RELU and pick the one that gives the best performance.

6. The network can be visualized with figure 3 below. This is the diagram of our initial architecture which might change throughout evaluating and optimizing our network.

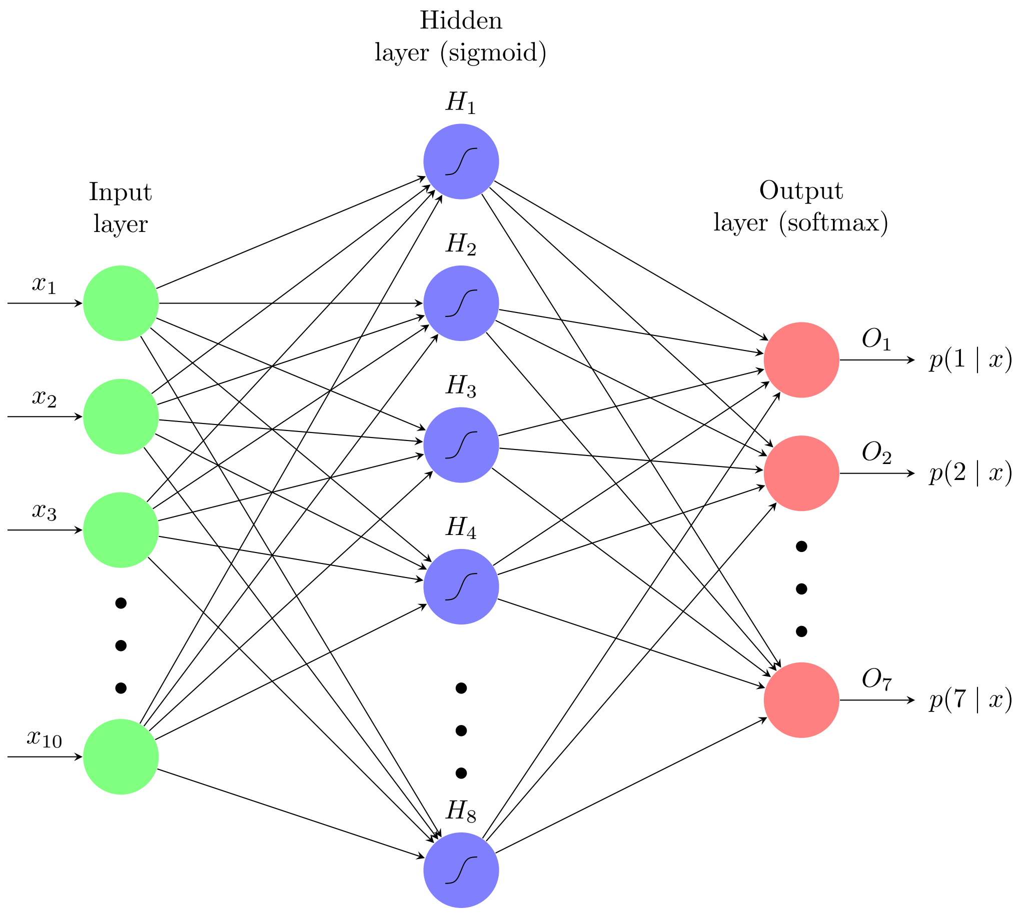
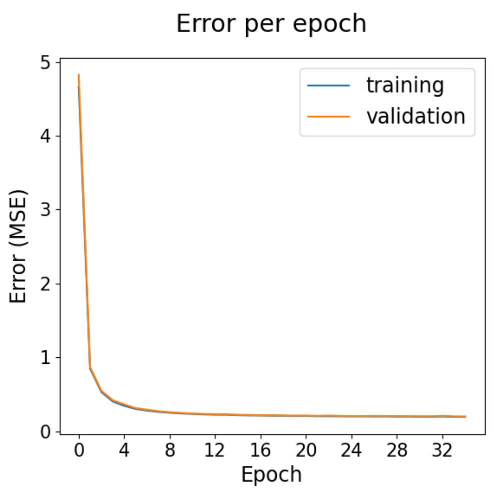


Figure 3: Neural network architecture

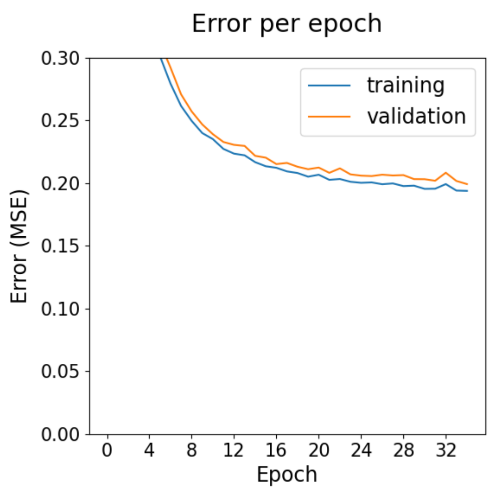
Training

7. The data is split into 3 parts - 15% for testing, 15% for validation, and 70% for training. Training data gets shuffled and put into mini-batches which are then used in the back-propagation algorithm to tune each weight and bias. The validation data is used to check for the cost on each training cycle. Once the training is complete, a graph that displaying cost over time is plotted. Observing this graph for different hyper-parameters is useful for tuning them. Finally, the test data is used to measure the performance of our network.

8. The performance is evaluated by observing multiple metrics on the result gathered from the test data. One of the metrics is overall accuracy - correct classifications / total classifications. This shows how well the network performs in general. A confusion matrix is also generated, providing correct/incorrect classification data on each class and making it easier to identify classes that the network often misclassifies. In the case of big discrepancies, the network could be inspected for bugs and the data for biases.

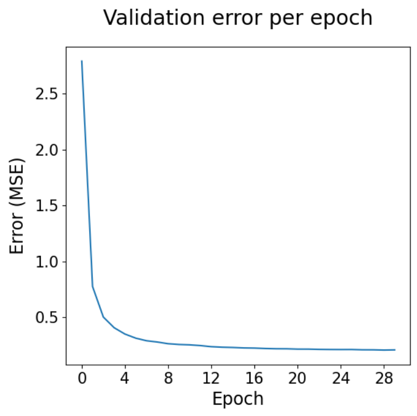


(a)

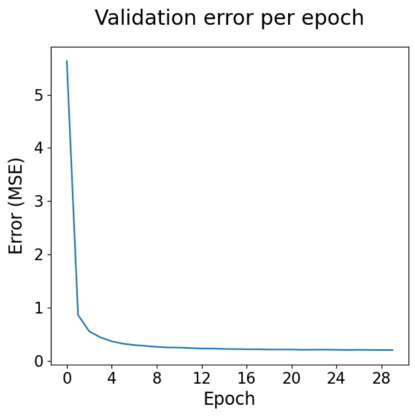


(b) Zoomed in

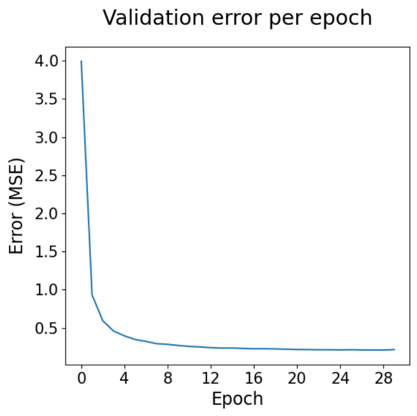
Figure 4: Cost function evaluated on the validation set and the training set per epoch.



(a)



(b)



(c)

Figure 5: Cost function over time for different initial weights and biases

9. A decision of when to stop training is made by monitoring the cost function and accuracy over time. Once accuracy stops meaningfully increasing and the cost function stops decreasing, training should be stopped.

Also, we plot the error on the training set and the error on the validation set as seen in figure 4 and see when the error on validation stops decreasing and starts increasing in order to avoid overfitting.

10. When testing different initializations we find that different initial weights and biases cause gradient descent to start at a different point in the loss space as seen in figure 8. However, the actual performance is dependent on the number of training cycles. With few epochs, gradient descent may begin at very different locations for different values but for more epochs are added, these values converge to the same local minima and the initialization values stop being relevant.

Optimization

11. The ANN was tested with six different hidden layers as seen in figure 6. It went from five all the way up to 45 to capture a wide range, and we found that the more neurons were placed, the better it performed in the cross validation. The trend overall was expected, as more neurons correlates with more features being retained and learnt. However, it peaked at 30, but we expected it to go down afterwards because the initial assumption was that more layers would lead to more over-fitting. With more neurons this would possibly be the case.

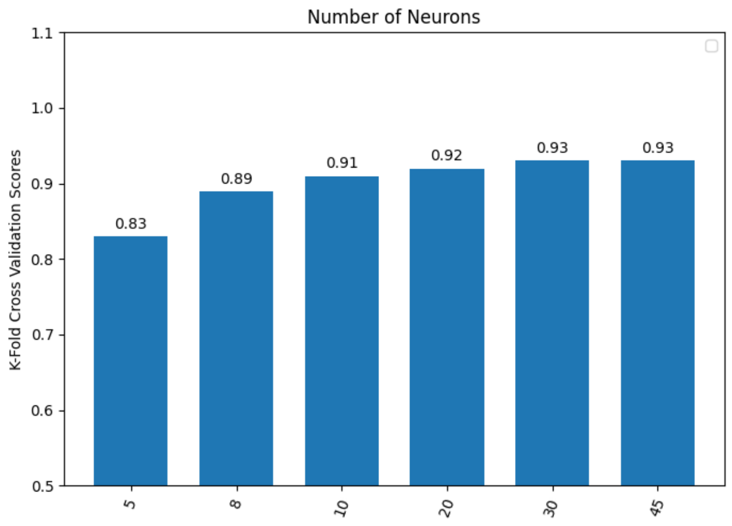


Figure 6: Performance of the network in relation to the size of the hidden layer

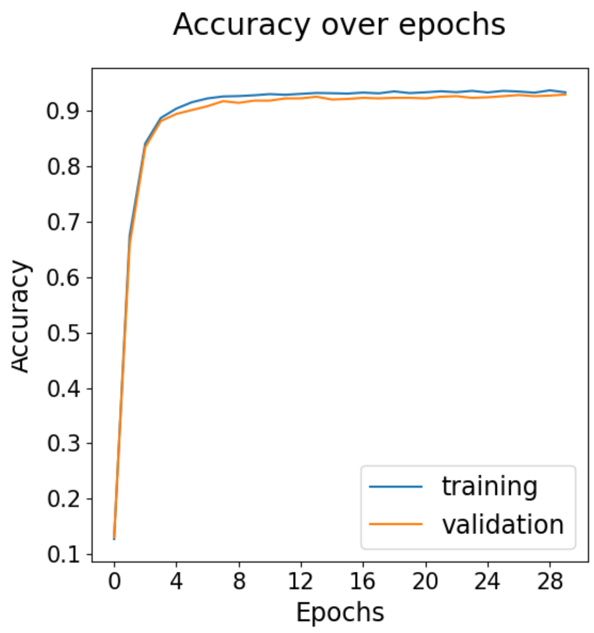


Figure 7: Accuracy of the network over epochs in training for 30 neurons in the middle layer

12. As 30 neurons in the middle layer provided the best performance, we further analyzed how it performed during training over epochs. Results are shown in figure 7. We chose this architecture because it provided the best performance in cross validation with the lowest risk of over-fitting and is less computationally taxing.

Evaluation

13. The success rate which is measured by the ratio of successful predictions over all predictions made across a test set is 0.93. The system achieves approximately this accuracy with all the functions that were used throughout the training algorithm, with the softmax used on the output layer and max log-likelihood used as a cost function proving to give a very slight improvement.

    When tested against the validation set the accuracy of the predictions remain in the same levels. This is a reason to believe that the Neural Network does not over-fit with training.

14. In order to have a better idea of how the Neural Network performed on each class the confusion matrix was produced and plotted in figure 8.

    The confusion matrix shows the count of the cases that were encountered when making the prediction. The percentages are computed for each actual class to demonstrate how each class is interpreted by the Neural Network. The x-axis is the class that was predicted and the y-axis the actual class of the data point. The top left square for example, shows the total times that the network predicted the data point to be of class 1 and the actual class was 1. The percentage given in the square shows that data points that correspond to class 1 were predicted correctly 92.26% of the time, were predicted as class 3 2.38% of the time and so on.

    Thus, from this matrix we see that mostly the highest number of cases reside in the diagonal, meaning that they are correctly identified by the algorithm. Among them the most successfully predicted class seems to be classes 2 and 7 given that they have the highest percentage and number of successful prediction as opposed to the incorrect predictions. Another point of discussion would be that from the matrix we can see that the neural network finds some similarities between pairs of classes such as the case with class 6 being predicted as class 5, 3.5% of the time or class 1 being wrongly classified as class 3 and vice versa a significant amount of times. These frequent miss-classifications indicate that there are differences between classes that the network is not trained enough to identify.

15. The unknown set of items are fed into the neural network and the results are then printed through a script in the a text file attached to the report.

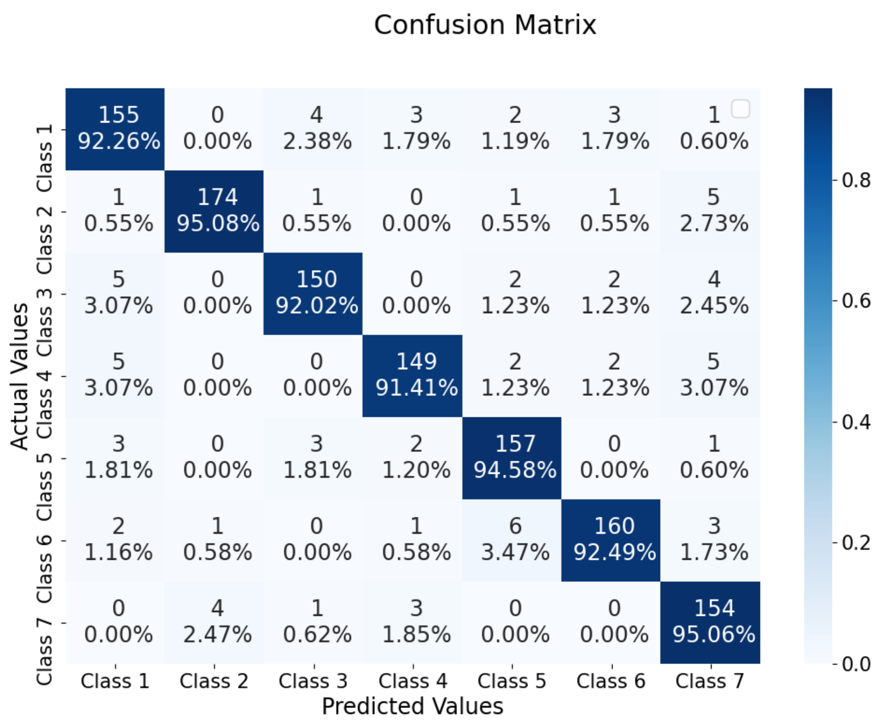


Figure 8: Confusion Matrix of the Neural Network Predictions

Scikit-learn

16. After tuning the hyper-parameters on the grid search we achieved a maximum accuracy similar to that of our own program. Interestingly, the grid search showed great variability, even when the hyperparameter choices were the same. When it came to activation functions, *all* of them were chosen to be the optimal function. This clearly depended on the starting positions, and showed that the activation function would not greatly impact the effectiveness of the ANN. We chose the logistic function hand-in-hand with the *softmax* function as it performed well for us. In terms of hidden layers, the grid search generally preferred one hidden layer of 25-35 neurons as opposed to multiple hidden layers. This actually aligns with our original hypothesis that one hidden layer would be enough. With regards to the learning rate, the grid search suggested that values between 0.005 and 0.01 achieve the most accurate training. Our learning rate was significantly above that, at 0.07.

17. After tuning our ANN with the hyper-parameters chosen by Scikit's grid search, we saw no significant improvement. Regarding the activation function, *softmax* and *ReLu* performed best, but depending on initial conditions, either could edge above the other, and this indicated neither was superior. The same story held true with the middle layer. While we were now certain one hidden layer was best, anything between 25 & 35 performed similarly. The learning rate, on the other hand, definitely impacted the performance. Negatively, in fact. This stems from the differences in our implementations, where our choice is better suited to our program. All in all, implementing grid search was not very helpful, but was a useful tool in comparing vastly different architectures.

References

[1] Jeff Heaton. *Introduction to Neural Networks for Java, 2nd Edition.* 2nd. Heaton Research, Inc., 2008. ISBN: 1604390085.

[2] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators". In: *Neural Networks* 2.5 (1989), pp. 359–366. ISSN: 0893-6080. DOI: `https://doi.org/10.1016/0893-6080(89)90020-8`. URL: `https://www.sciencedirect.com/science/article/pii/0893608089900208`.