1 Introduction

The task discussed in this report is solving the XOR problem, using a minimum layer perceptron (MLP) trained on noisy datasets of various sizes. To achieve this, a number of experiments were conducted by varying the training set size and the number of hidden neurons used. The main aim of the experiments was to identify if there is an influence of the training set size and hidden layer size on the accuracy of the model.

The XOR problem is given one input vector $v = (x1, x2)^T$ of boolean values, predict the outputs in the table below.

$x1$	$x2$	$x1 \oplus x2$
0	0	0
0	1	1
1	0	1
1	1	0

Figure 1: Table showing the input and output combinations for XOR.

This problem can be viewed as a classification problem, in which the input vectors are assigned two classes (0 or 1). There are four input points, one for each vector, which can be plotted in a 2D plane. By doing so it can be clear that the problem is not linearly separable, which makes the problem non-trivial. The MLP needs at least 2 hidden neurons since simpler architectures can only perform correctly on linearly separable input points [1]. The values for XOR are not linearly separable. This means that, as show in figure 2, there is no way of drawing a line between the data points, which separates points of class 0 from points of class 1.

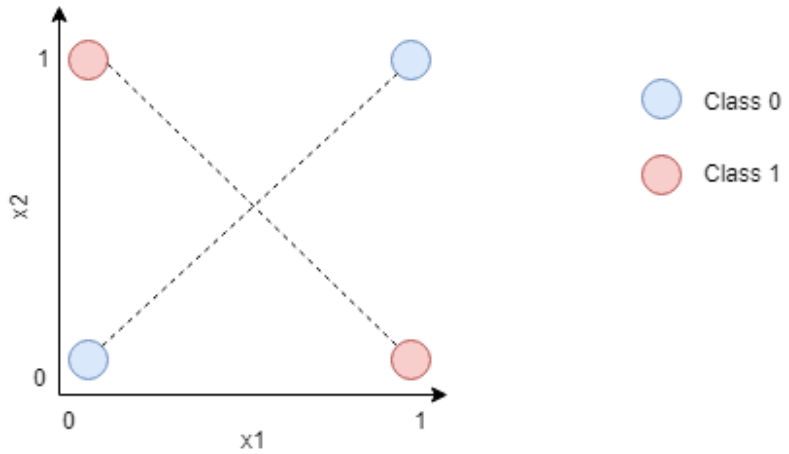


Figure 2: The data points of the XOR problem with the corresponding classes, plotted in a 2D plane, to show that the inputs are not linearly separable.

The solution proposed as part of this assignment is by using a Multi-Layer Perceptron (MLP). The MLP consists of (1) an input layer, which has all the dimensions of the input and a bias term; (2) one or more hidden layers, each one with multiple neurons which communicate through weights - mimicking the synaptic connections in the human brain; (3) an activation function; which determines the output of a neuron, by mapping it into a desired range; (4) an output layer; with a neuron for each dimension of the output problem [2].

To truly challenge this architecture and see how it performs, the inputs have been generated with noise. This is to make sure that even though there are some errors in the data given, the MLP can still figure out what the significant trends behind the data are. However, as proved in [3], a minimum configuration MLP is sufficient for solving the XOR problem.

2 Training set

The training sets used for the experiments described in this report are generated from the original 4 input vectors for XOR, by adding noise sampled randomly from a normal distribution $\mathcal{N}(0, 0.25)$. Across the generated dataset, the maximum difference reached between the actual value and the noisy value of a data point was approximately 1.45. This difference is quite large and the amount of noise added to the datasets is considerable, meaning that more complicated architectures might perform better.

The sizes of the training sets: 16, 32 and 64 are all multiples of 4, since they are generated by replicating each of the 4 input possibilities by 4, 8 and 16 times each, respectively. This was done to ensure the uniform distribution of input possibilities across the training set and therefore avoiding any biases towards either of the respective output values.

The training sets are always re-generated, not extended upon each other. In some cases it might be useful to have the smaller training sets be a subset of the others, but this could also have the disadvantage that if there are particularities of the smallest training set that confuse the model, these will be in every other training set. For example it might sometimes happen, by chance, that there is a pattern in the noisy training set that is not at all connected to the actual problem pattern. In this case, the model might pick up on the false pattern and thus overfit to this training set. I considered it more relevant for the experiments to generate the training sets independently, form the same noise distribution. Since they have the same noise distribution, the results should not be influenced by the fact that the training sets were independently generated.

3 Testing set

The testing set was generated in the same manner as the training set: from the 4 possible inputs with added noise from a normal distribution $\mathcal{N}(0, 0.25)$.

All the experiments were done on the same test set. This is such that results are meaningful across experiments. If there were multiple testing sets and one happened to contain easily predictable samples, then the experiment corresponding to it would not be comparable to the others. Of course there is still a chance that the testing set generated itself has some easily predictable samples, but since the same one is used for all experiments, it will be easy to predict for all. In this case the differences would be in just how easy.

The testing set was generated independently from the training sets, to test on previously unseen data. This represents a better way of telling how good the models are at predicting and therefore at understanding the pattern behind the data.

4 Models

For the experiments described in this project, there were 3 models used, each of them trained on datasets of sizes 16, 32 an 64. All models had a single hidden layer, one input and one output layer. The differences between them consisted from the number of neurons on the hidden layer:

- Model A (2 hidden neurons)
- Model B (4 hidden neurons)
- Model C (8 hidden neurons)

The intuition behind the 2 hidden neurons is that one will act as an OR gate and the other one will act as a NAND gate. [4] These operations are enough to define XOR, since an input vector $v = (x1, x2)^T$ should only activate the output neuron if and only if $x1 \vee x2$ and $\neg(x1 \wedge x2)$ are True. The truth table for XOR cannot be expressed in terms of a single other logical operation. Leaving the intuition behind, a model for successfully solving the XOR problem needs at least two hidden neurons, since (as explained in the Introduction) the input is not linearly separable.

All three models should be able to solve the XOR problem, since they have more than 2 hidden neurons. However, there is a chance that model A is too simple and it will not understand enough of the data. Model C might be too complicated, since the number of weights to be adjusted ($2 \times 8 + 8 \times 1 = 24$) is much larger than the problem complexity (4 possible input combinations). In this case the model might start overfitting. These are simple speculations and will be verified or contradicted by the experiments below. One problem

is that there was a considerable amount of noise added to the input vectors and this will likely have a big impact on how the networks behave.

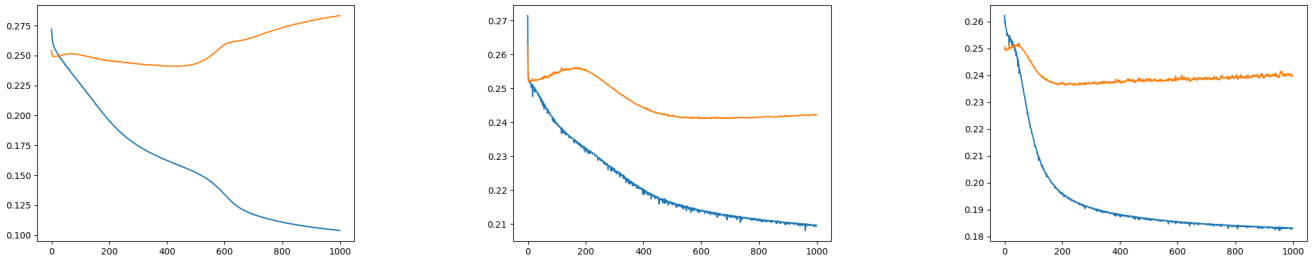
The sigmoid function is used as an activation function for all models, to "squash" the sum of input values closer to either 0 or 1. The sigmoid function is also called the logistic function and has the value below:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

5 Experiments - MSE against Epoch

The first experiment to be conducted as part of this assignment was to train the models A,B and C (described above) on three different sets of noisy input vectors, with sizes 16, 32 and 64 respectively. By observing the convergence of the mean squared error with increasingly larger epoch numbers for training and validating sets, it was possible to observe different patterns and analyse how *well* each model performs on unseen data.

Model A (2 hidden neurons)



(a) Training set size 16 (b) Training set size 32 (c) Training set size 64

Figure 3: Mean Squared Error against Epoch for Model A (2 hidden neurons). Fitting on training set of different sizes (blue) and validating on the same set of 64 samples (orange)

Figure 3(a) shows how model A, the MLP with 2 hidden neurons, behaved on a training set of size 16 as compared to the test set of size 64. The line representing the proportion between epochs and mean squared error - MSE - for the training set (blue) is converging to 0 as the number of epochs increases. This is because the model learns the optimal parameters for the training set, fitting more and more to various patterns in the data. The MSE for the test set (orange) decreases in a more shallow manner. This is to be expected since it is unseen data, so the parameters that are precisely tailored to the training set will not have that much of an impact on the test set. At approximately 550 epochs, there is a local minimum for the testing set MSE, which corresponds to an inflection point in the training set. After 550 epochs the model starts overfitting to the training set, the weights not being relevant to the general problem any more and thus increasing the validation error.

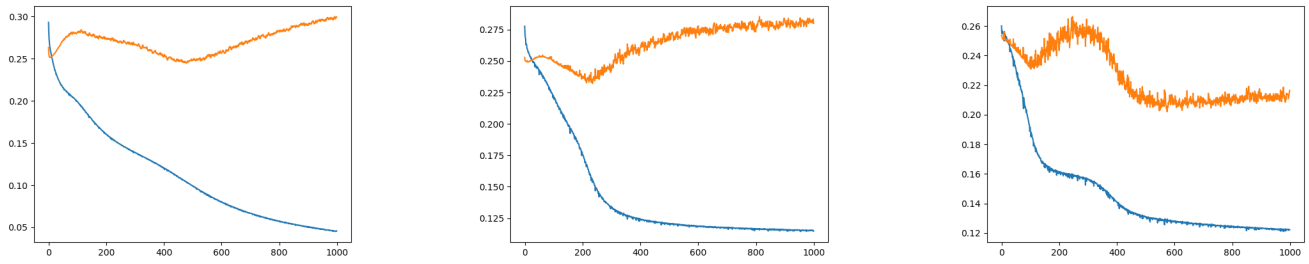
Figure 3(b) shows how model A, the MLP with 2 hidden neurons, behaved on a training set of size 32 as compared to the test set of size 64. The blue line (epoch against MSE for training) is converging towards a lower error as the epoch number increases. In the early stages of training, the MSE decreases almost instantly for both the training and the test set. As expected, the decrease in validation (testing) error is much smaller than the training error. The training set MSE is decreasing gradient, but the testing set MSE has a local maximum point around 200 epochs, before it converges towards 0.24. Both lines are oscillating between epoch numbers, instead of decreasing smoothly. This could be because of the step in the gradient descent, which falls on different sides on the minimum point at each iteration.

Figure 3(c) shows how model A, the MLP with 2 hidden neurons, behaved on a training set of size 64 as compared to a test set of the same size. The model MSE is decreasing on both the training set and the validation set, with a much steeper fall for the training set. The validation set MSE seems to be growing in the larger number of epochs, but slower than the initial decrease, around 200 epochs. As in figure 3(b),

there is an oscillation from the general trend (noise). This can be explained by the same phenomenon of the parameters going on either side of the minimum at each iteration of the gradient descent. To improve this, the learning rate could be decreased.

Model A obtained a mean squared error of 28.34%, having been trained on a set of size 16; 24.22%, on a set of 32; 23.96%, on a set of size 64. This shows a decrease in MSE with the increase of the training set size, which is expected because more information helps the network learn more. It can also be observed that in the models which were trained on more data, the MSE "increase" which comes in the later stages of the training is flattened out.

Model B (4 hidden neurons)



(a) Training set size 16 (b) Training set size 32 (c) Training set size 64

Figure 4: Mean Squared Error against Epoch for Model B (4 hidden neurons). Fitting on training set of different sizes (blue) and validating on the same set of 64 samples (orange)

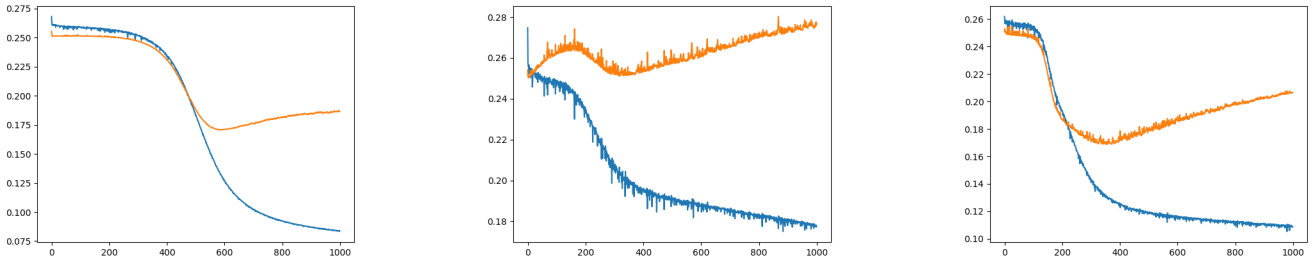
Figure 4(a) shows how model B, the MLP with 4 hidden neurons, behaved on a training set of size 16 as compared to the test set of size 64. The figure shows the mean squared error -MSE- against the epoch number, for the learning (blue line) and validating (orange line). The gradient of the training set MSE is quadratic and with a smooth line as the model fits the weights to the training data. The testing set MSE shows an increase in the early stages (low epoch), because the model has not yet had to opportunity to learn significant trends in the data. However, after approximately 100 epochs, the validation MSE begins decreasing, reaching a local minimum around 500 epochs. After this point the model starts fitting to noise in the training dataset, therefore performing worse on the unseen data.

Figure 4(b) shows how model B performed on a training set of size 32 and on a validation set of size 64. The training set MSE decreases in a parabola, with a turning point at around 300 epoch. The validation set reaches a minimum of MSE at approximately 250 epoch. Afterwards the model is overfitting to the training data. The increase in MSE for the training set is becoming less and less steep as the number of epoch increases. This could be because the model is close to the optimal parameters for the training data, therefore not having much impact on the validation error either.

Figure 4(c) shows how model B performed on a training set of size 64 and validated on a set of the same size. The blue line shows how the mean squared error decreases over epochs in training. A first observation is that the model is more unsure of the parameters, falling in and out of the optimal minimum, oscillating around the general trend. The model shows how this oscillation has a much bigger impact on the validation set (larger range) than on the training set. The testing MSE is increasing in the early stages of training, reaching a maximum at approximately 300 epochs. The model reaches a minimum MSE for validation at 500 epochs, point after which it begins overfitting to the noise in the training set, but with very little steps at each increment. This is giving a shallow increasing slope for the testing and decreasing for the training.

Model B gave a performance of 29.96% MSE for a training set of size 16; 28.00% MSE for a training set of size 32; and 21.65% MSE for a training set of size 64. This shows a decrease in MSE as the training set size increases, showing that there is a need for more data to gain a better understanding of the patter. One reason why this happens could be that the number of weights in the model ($2 \times 4 + 4 \times 1 = 12$) is still much smaller than the amount of data in the testing set (64 samples).

Model C (8 hidden neurons)



(a) Training set size 16 (b) Training set size 32 (c) Training set size 64

Figure 5: Mean Squared Error against Epoch for Model C (8 hidden neurons). Fitting on training set of different sizes (blue) and validating on the same set of 64 samples (orange)

Figure 5(a) shows how model C, the MLP with 8 hidden neurons, performs, in terms of mean squared error over epoch increase, for a training set of size 16 and a validation set of size 64. In the early stages of training, both the training set MSE and validation set MSE are decreasing slowly with the increase of epochs. A considerable amount of epochs (approx. 250) is needed to understand the data and fit to the appropriate weights. After these are found both MSE, training and validation, drop considerably in a much steeper manner. The model begins to overfit after 600 epochs, when the parameters are fixed to the training set noise rather than to any significant trends in the data.

Figure 5(b) shows the mean squared error against epoch number of model C on a training set of size 32, as compared to the validation set of size 64. The testing set MSE begins with a lower value than the training set MSE, which could simply be by chance - the initial parameters are a good tell of the data. The model is oscillating between either side of the local minimum, gradient descent not finding the optimal parameters. Because the number of hidden neurons (8) is greater than the needed number (2), there are extra degrees of freedom when choosing weights. This is why some neurons begin to "share" features, when these features are noisy, the covariance is higher. Having multiple neurons focus on the same noise features leads to having the model be more likely to predict based on them, thus giving a higher MSE for unseen data. This could explain why the model shows an initial increase in the training set MSE.

Figure 5(c) displays the convergence of mean squared error over epochs in both training set and validation set of equal size, 64 samples. The orange line, corresponding to the mean squared error of the testing data, shows a steep decrease until approximately 375 epochs. This point loosely corresponds to where the gradient of the testing set becomes more shallow. One reason why this might be is that the model has already learnt the patterns begind the data and is fitting to the noise, but since the noise is less important than the general patterns, the gradient is more shallow.

Model C performed to an MSE of 18.68% for a training set of size 16; 27.72% for a training set of size 32; and 20.66% for a training set of size 64. The smallest error at 1000 epoch is for the smallest training set. However, at the minimum point, both the training set of size 16 and of size 64 gave an MSE of about 17.5%. The difference in MSE for 3(b) could be down to simple differences in input noise that the model picked as relevant.

Summary

The table below (figure 6) shows the mean squared error for the 9 experiments described above. Out of the three models with the 3 training set sizes, the best MSE performance was obtained by model C. Model C had the most neurons and therefore the most weights to be adjusted, giving it more power to capture more trends in the data and accentuate important features. For models A and B it is noticeable that the MSE improves as the size of the training set increases. However, for model C this seems to not be the case. It is likely that this comes down to the learning rate being too small (as explained above) and causing the oscillation in and out of optimal parameters. At the minimum point in each of the experiments done on model C, it performs better than the other models.

The model which seems to perform worst is model B for the training sets of sizes 16 and 32; and model A for the training set of size 64. Since model B has more neurons, it can capture more features, but if these

Model	16	32	64
A	28.34	24.22	23.96
B	29.96	28.00	21.65
C	18.68	27.72	20.66

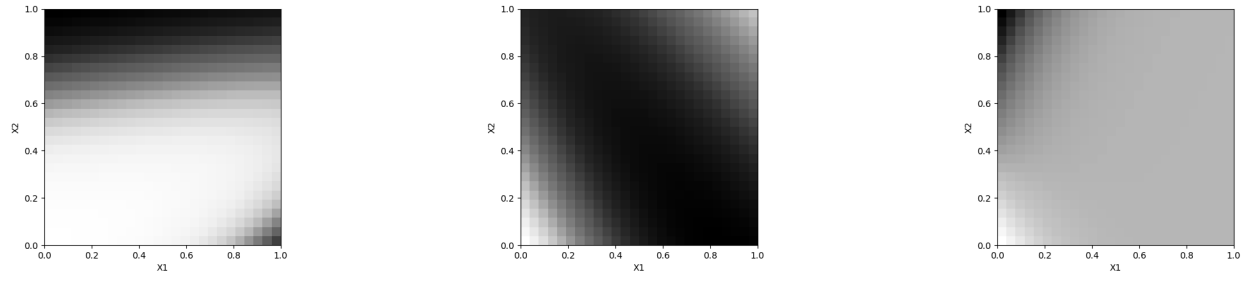
Figure 6: Table showing the Mean Squared Error for 1000 epochs for the three models (A, B and C) after being trained on datasets of sizes 16, 32 and 64 samples.

features are noisy then it is likely to overfit. However, if it is given a larger training set, such as the one of size 64, the model begins to put more importance on the correct features, instead of fitting to noise. This would justify the fact that it performs worse than a simpler model (A) on the smaller training sets, but better on the larger training sets.

6 Experiments - Output function

The second experiment to be conducted as part of this assignment was to test the three models - having been trained on sets of sizes 16, 32 and 64 - on a matrix of input data between 0 and 1. Testing it on pairs of values between 0 and 1 and plotting the predictions as white-black values, would allow to see areas in which the model predicts 0 values (white) or 1 values (black). The ideal model would learn the actual rules behind the XOR function and therefore predict values of 1 around corners (0,1) and (1,0), values of 0 around corners (0,0) and (1,1), leaving a grey area in the middle, with smooth transitions to white or black.

Model A - 2 hidden neurons



(a) Training set size 16 (b) Training set size 32 (c) Training set size 64

Figure 7: Output from model A (2 hidden neurons). Shows difference between different training set sizes. The colour scale of black to white represents 1 to 0 values.

Figure 7(a) shows model A predictions inside the unit square, after training on a set of size 16. The model predicts 1 correctly for (0,1) and (1,0), but also for (1,1), which is incorrect. The model seems to always predict 1 when x2 is 1. The fact that there is not much black around the corner (1,0) could imply that the model is more unsure of this pattern. One reason behind this could be that there was more noise in the samples generated from (1,0).

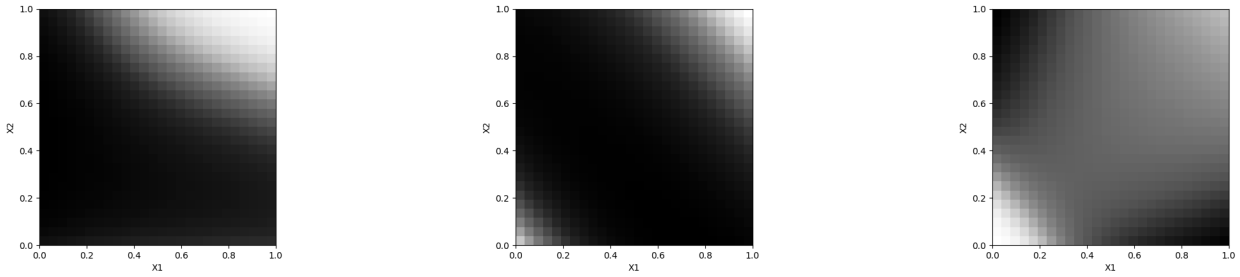
Figure 7(b) shows a considerable improvement from figure 7(a). Model A is now predicting on unit square values, after training on a set of size 32. It correctly determines the values for all four corners: 1 for (0,1), (1,0) and 0 for (0,0),(1,1). However it is noticeable that the output matrix is predominantly black. This shows a bias of the model towards predicting 1. Because the sigmoid function "sqashes" the values, the model predicts with very high certainty (there are few grey areas). Some symmetry can be noticed, meaning that the model has gathered approximately the same amount of trends from all four possible input vectors.

Figure 7(c) shows that in the case of training on 64 samples, model A is very unsure of most of the unit square. The only significant trends it can identify are the values when x1 is 0, i.e. corners (0,0) and (0,1). This could be because the model parameters got stuck in a local minimum, early on in the training, fitting

to noise rather than actual trends.

Overall, model A picks up on at least two corners of the ideal output matrix. The best experiment on model A was on the training set of size 32, where the model picked up on all four corners.

Model B - 4 hidden neurons



(a) Training set size 16 (b) Training set size 32 (c) Training set size 64

Figure 8: Output from model B (4 hidden neurons). Shows difference between different training set sizes. The colour scale of black to white represents 1 to 0 values.

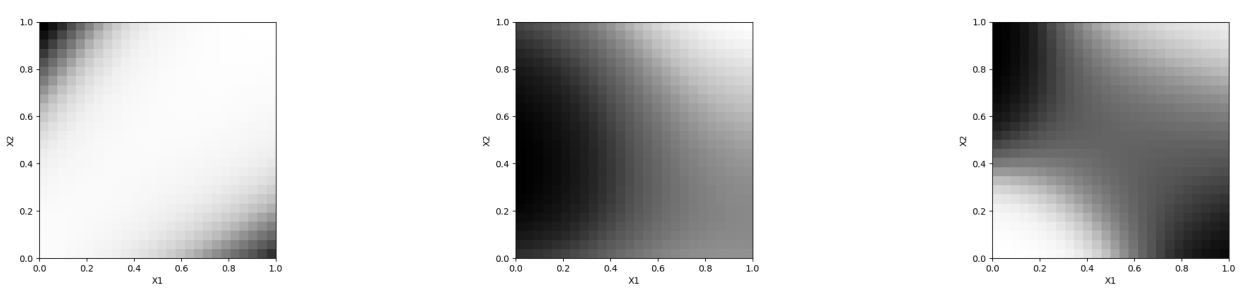
Figure 8(a) shows the predictions made by model B on the unit square, after training on 16 samples. The model pretty confidently predicts 1 (black) for most of the unit square, but it picks up on the fact that corner (1,0) should be 0. The transition from corner (1,0) towards the center of the square is smooth. The fact that the model predicts corner (0,0) to be 1 could be due to the noise that prevents it from understanding the 4 samples coming from the input vector (0,0).

Figure 8(b) shows an improvement from figure 2(a). Adding more samples to the training set has helped the model begin to understand all four corners. However, the model still does not show as much 0-coverage for corner (0,0) as it does for corner (1,1). This could be because the training sets had more values towards 1 than towards 0, after adding noise.

Figure 8(c) shows how model B predicted outcomes on the unit square values, after training on a set of size 64. The model has a much bigger uncertainty in the middle of the square, which is the correct expected behaviour, but the model also fails to predict value 0 (white) in the (1,1) corner. The model therefore only partly learnt the true feature space, leaving mostly noise where the (1,1) samples should have been.

Overall, model B seems to be learning more and gaining more 0-coverage for corner (0,0): starting from predicting 1 to predicting 0 for most corner values. At the same time the model is loosing 0-coverage for the (1,1) corner, becoming more and more uncertain. The model only starts to predict with uncertainty (grey areas) after training on a set of size 64. This could be due to the variance term, which means that because of the more extensive training the model "understands" that the values in the middle are more likely to interchange between 0 and 1, than those in the corners.

Model C - 8 hidden neurons



(a) Training set size 16 (b) Training set size 32 (c) Training set size 64

Figure 9: Output from model C (8 hidden neurons). Shows difference between different training set sizes. The colour scale of black to white represents 1 to 0 values.

Figure 9(a) shows model C predictions on the unit square, after training on a set of 16 samples. Since the model has more weights, it picks up on relevant feature trends early, without needing many samples to train on. Even with only 16 samples, the model has identified the correct values for all 4 corners. The only problem is that, because of the lack of samples, it will not "learn" when to be undecided. For this reason the middle area of the unit square is 0 (white).

Figure 9(b) shows the predictions made on the unit square values by model C, after training on a set of size 32. The model seems to think that values around (0,0) and (0,1) are 1 and is uncertain of anything in the corner of (1,0). This could be since the number of weights to be adjusted is much larger than the number of possible features, meaning that the model might "learn" the noise in the data, overfitting to it. Some neurons will be activating on patterns that only represent noise.

Figure 9(c) shows the greatest resemblance to the "ideal" output function. Model C was trained on a set of size 64, from which it learnt all the 4 corners of the unit matrix, corresponding to the 4 input vectors and correctly represented uncertainty in the middle. The model has enough neurons to understand all the patterns behind the data and enough samples to start distinguishing noise from real trends.

Overall, model C showed the best performance when trained on a dataset of 64 samples. It correctly displays the values for all 4 input vectors and shows a smooth difference in certainty when moving from the corners towards the center of the square, adding noise.

Summary

The best output function was obtained by model C, after training on a set of size 64. The experiments show that both models A and B can achieve an output map which clearly identifies all 4 input possibilities correctly, but only when trained on 32 samples. If models A or B are given more data (64 samples), they become more uncertain of most areas in the matrix and lose sight of relevant trends in the data. This could be because they begin to notice that the "middle area" is not always either 1 or 0, but has overlapping values. Noticing this, makes up for some of the neurons in the architecture, not leaving enough to capture the trends corresponding to the 4 corners of the unit matrix. It can be noticed that model B outputs better predictions for the unit square, even when trained on 64 samples. Overall, when the models are given more data, they begin to notice more significant trends in the data, but they need enough neurons to capture all (2 white corners, 2 black corners and the grey middle area). The only experiment which satisfied all of these was after training model C (8 hidden neurons) on a dataset of size 64.

7 Conclusions

The purpose of this project was to experiment and understand the influence of training set size and number of hidden neurons of a Multi-Layer Perceptron (MLP) architecture with one hidden layer. The experiments have shown that despite the noise added, the network can still pick up on most relevant trends in the data, all experiments resulting in a mean squared error of below 30%. The architecture that performed best was the model with 8 hidden neurons (Model C), obtaining a mean squared error of 18.68%. Having been trained on a set of size 64, model C successfully creates an output function for the unit square which correctly captures all features of the data. The impact that the training set size has on the model performance is that in most cases, more data helps the model "understand" the significant trends behind the problem. If the model architecture is too complex and there is not enough data given, sometimes it will overfit, the extra degrees of freedom focusing on noisy features, instead of meaningful trends in the data.

References

[1] L. Araujo, "Solving xor with a single perceptron. advocating for polynomial transformations as a way to increase the representational power of artificial neurons.." https://medium.com/@lucaspereira0612/solving-xor-with-a-single-perceptron-34539f395182, 2018. Accessed: 2019-05-01.

[2] M. Riedmiller, "Advanced supervised learning in multi-layer perceptrons from backpropagation to adaptive learning algorithms," *Computer Standards Interfaces*, vol. 16, no. 3, pp. 265 – 278, 1994.

[3] V. K. Singh, "Proposing solution to xor problem using minimum configuration mlp," *Procedia Computer Science*, vol. 85, pp. 263 – 270, 2016. International Conference on Computational Modelling and Security (CMS 2016).

[4] S. O. Dukor, "Neural representation of and, or, not, xor and xnor logic gates (perceptron algorithm)." https://medium.com/@stanleydukor/neural-representation-of-and-or-not-xor-and-xnor-logic-gates-perceptron-algorithm-b0275375fea1, 2018. Accessed:2019-05-01.

Code available: https://github.com/avaspataru/CS311-Neural-Computing/blob/master/MLP.py
(Uses Python 3 and Keras)