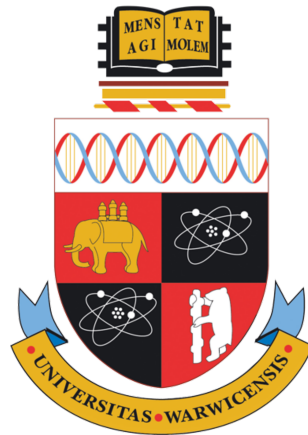


CS331: NEURAL NETWORKS



Modelling the XOR Problem by Implementing a Multilayer Perceptron

May 2, 2019

Marek Topolewski
University ID: 1633084

Module organiser: Prof Roland Wilson

Abstract

Neural networks are currently one of the fastest growing areas of computer intelligence. They are also arguably our best attempt to model human brain thus far. This project attempts to provide a theoretical and empirical analysis of a problem that early neural networks were incapable of overcoming - simulation of the XOR logic gate. A common solution to this issue is the multilayer perceptron. This project implements and evaluates this neural network under a variety of parameters to determine their influence over the model's performance.

Contents

1	Introduction	4
2	Perceptron	5
3	Problem of XOR	5
4	Multilayer Perceptron	6
5	Project Design	8
6	Implementation	9
7	Testing	10
7.1	Output maps	11
7.2	Convergence plots	11
7.3	Accuracy on the test set	11
8	Evaluation	12
8.1	Output maps	12
8.2	Convergence Plots	12
8.3	Accuracy on the test set	13
9	Conclusions	14
A	Output maps for batch mode and selected parameters.	16
B	Output maps for online mode and selected parameters.	17
C	Learning error plots for batch mode and selected parameters.	18
D	Learning error plots for online mode and selected parameters.	19
E	MSE on the test set for MLPs in batch and online mode.	20
F	Average MSE on a random dataset for all model configurations.	20
G	Average MSE for all model configurations.	21

1 Introduction

Artificial neural networks (ANN) are the result of the pursuit to simulate the biological neural network of a human brain. These models are an interconnected system of artificial neurons which imitate the basic computational capabilities of the biological ones. The ANNs owe their performance edge over many other machine learning (ML) models primarily to the connectivity as opposed to their building units.

A neural network can be represented as a graph (see Figure 1) where each node illustrates a neuron or an input unit. Another important component is the set of directed edges between these vertices which mimic the neurological connections between biological neurons. To model the **synaptic plasticity** of a human brain, each edge has a weight associated with it. Increasing this value is analogous to the alteration of signalling strength in synapses which is the outcome of either structural or chemical changes in the brain's connections upon learning. The aim of an ANN is to assign weights to these connections such that the error function is minimised.

The earliest ANNs are the **feedforward networks** of which a prominent example is the **linear associator** (LA) proposed by James Anderson [1]. Models of this class are static, i.e. facilitate no feedback or cycles in their connections. A simple artificial neuron is described as follows:

$$\mathbf{u} = \mathbf{w}^T \mathbf{v} \quad (1)$$

where \mathbf{u} is the output vector, \mathbf{v} is the input vector, \mathbf{w} is the weight vector of connections between each input node and the neuron.

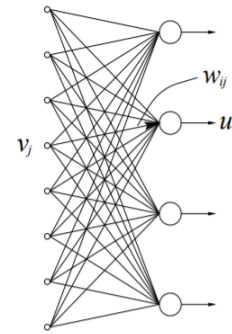


Figure 1: Visualisation of a linear associator where v_j represents the j^{th} input, u_i the i^{th} output, and w_{ij} the weight of the edge between them.

Source: [CS331 Neural Computing: Lecture Notes](#)

The learning is facilitated in a supervised learning fashion by employing the **delta rule** to gradually converge to their optimum. The connection weights \mathbf{W} are altered proportionally to $\Delta \mathbf{W}$ after being presented with each training feature-target pair. This policy is known as the **gradient descent** and is given by the following formula [2]:

$$\Delta \mathbf{W} = -\alpha \frac{\partial(\mathbf{u} - \mathbf{W} \mathbf{v})}{\partial \mathbf{W}} \quad (2)$$

where α is the learning rate and the numerator of the fraction is the prediction error. The delta rule simulates the human **positive and negative reinforcement** process in response to reward and punishment correspondingly.

Due to the linearity of the LA, the model is solely able to project an input vector onto a linear subspace spanned by the training target vectors, hence, lacking adaptiveness to input noise [14]. This implies that the model is prone to **overfitting**, i.e. given a sufficiently high degree of freedom, a perfect fit on the training data is obtained, however, it results in poor performance on unobserved inputs.

2 Perceptron

In order to improve the robustness of a linear associator, a degree of non-linearity ought to be introduced. In 1957 Frank Rosenblatt invented the perceptron [11] which suggests the use of an **activation function** to achieve this.

The proposed neuron model is defined by:

$$u = f(\mathbf{w}^T \mathbf{v} - \theta) \quad (3)$$

where f is the monotonically non-decreasing non-linear activation function and θ is an optional constant threshold. The activation function employed for the purpose of this project is the **sigmoid function** described as follows:

$$f(x) = \frac{1}{1 + \exp[-x]} \quad (4)$$

An example of a single layer and single neuron perceptron is presented in Figure 2. The model is comprised of D input nodes each of which has an outgoing edge to the neuron node with a corresponding weight w_d . The computational unit aggregates the inputs into a weighted average and passes the resulting scalar through the activation function f which produces the final output.

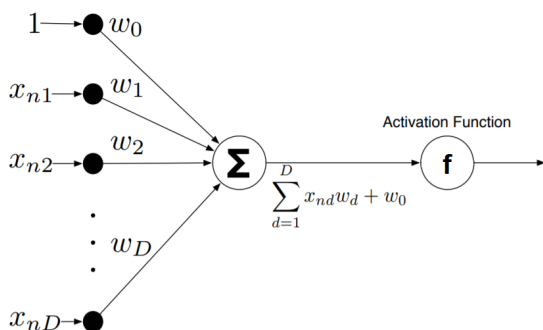


Figure 2: Illustration of a single neuron perceptron.
Source: [CS342 Machine Learning: Lecture 8](#)

3 Problem of XOR

Despite the perceptron's success in modelling classification problems such as the AND and OR logical functions with ease, the network remains unsuited to correctly implemented the XOR gate [6]. The issue was discovered by Minsky and Papert in 1969 [8]. In fact, only a subset of problems is possible for a simple perceptron to simulate, this class is known as **linearly separable**.

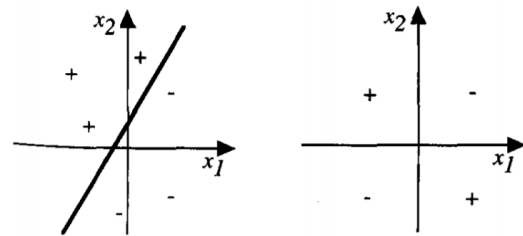


Figure 3: On the left, a linearly separable problem. On the right, a problem for which a line cannot be drawn to separate its classes, hence, not linearly separable.

Source: [CS342 Machine Learning: Lecture 8](#)

Classification problems in this group exhibit the following property - target classes cannot be separated by a single **decision boundary** [14] which is a hyperplane in the feature space defined by:

$$\mathbf{w}^T \mathbf{v} - \theta = 0 \quad (5)$$

As an example, Figure 3 displays two subplots representing different classification problems. The first is linearly separable since a line (hyperplane in 2D space) illustrating the decision boundary can be found. On the other hand, the latter subplot depicts the XOR feature space for which a hyperplane cannot be drawn to partition the classes.

The XOR problem is particularly adequate for this project, as it is sufficiently complex (not linearly separable) but one can also easily visualise its classification results and analyse the underlying model.

4 Multilayer Perceptron

It was not until 1975 when P. Werbos devised the **backpropagation algorithm** for neural networks [12] which enabled the construction of a perceptron with many layers of neurons. Such a neural network is referred to as a **multilayer perceptron** (MLP).

Not only does an MLP consist of an input and an output layer but also one or more **hidden layers**. Alike an output layer, a hidden is also a computational layer, i.e. it consists of neurons, known as **hidden neurons**, though, their number need not be equal to the size of the output vector. The input layer is connected to the first hidden layer. Adjacent hidden layers are mapped in an identical fashion and the last hidden layer is the input to the output layer (see Figure 4).

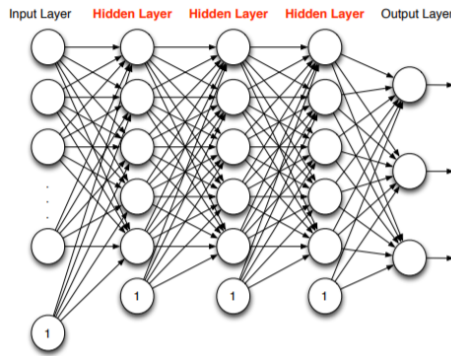


Figure 4: An example of a 5-layer MLP with 4 computational layers and a bias at each computational layer. Source: [CS342 Machine Learning: Lecture 9](#)

An MLP is able to model more complex problems than the perceptron, namely, it produces a **convex** decision boundary (see Figure 5), i.e. a conjunction of hyperplanes. The adaptiveness of the model makes the MLP a **universal function approximator**, hence, it can simulate any *smooth* feature to target mapping [7].

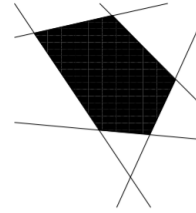


Figure 5: Graphical depiction of a convex decision boundary - there is no hyperplane between two points within it having an area outside of the convex space. Source: [CS331 Neural Computing: Lecture Notes](#)

The **forward pass** (also forwardpropagation) is the procedure of sequentially propagating the predictions \mathbf{u}_i of a layer to its successor. A recursive formula for obtaining the target prediction $\hat{\mathbf{t}}$ can be derived as follows [14]:

$$\begin{aligned}\hat{\mathbf{t}} &= f^l(\mathbf{u}^{l-1} \mathbf{W}^l) \\ &= f^l(f^{l-1}(\mathbf{u}^{l-2} \mathbf{W}^{l-1}) \mathbf{W}^l) \\ &= f^l(f^{l-1}(f^{l-2}(\mathbf{u}^{l-3} \mathbf{W}^{l-2}) \mathbf{W}^{l-1}) \mathbf{W}^l) \\ &= \dots\end{aligned}\quad (6)$$

where l is the number of layers, f^i is the activation function of the i^{th} layer, \mathbf{W}^i is the weight matrix of incoming edges and \mathbf{u}^{i-1} is the output vector from the previous layer.

Additionally, the inclusion of **bias** (also offset) generally proves beneficial [2] and often critical in ANN learning as it enables for the activation function output to be shifted on the X-axis. The offset is a constant value b that can be incorporated into the neuron's model as follows:

$$\mathbf{u} = f(\mathbf{w}^T \mathbf{v} + w_{n+1} b) \quad (7)$$

Having defined the operation of the forward pass, the MLP's convex decision boundary can be proved by solving the XOR problem. It suffices to construct a simple multilayer model with a single hid-

den layer comprised of only two hidden neurons utilising the **threshold function**:

$$f(x) = \begin{cases} 1, & \text{if } t \geq 0.5 \\ 0, & \text{otherwise} \end{cases} \quad (8)$$

By performing a single forward pass on a static ANN (see Figure 6) it is easy to calculate the outputs for the possible input vectors and assert its correctness. Further analysis of the intermediate results reveals that two distinct decision boundaries are generated by each of the hidden neurons (see Figure 7). Consecutively, the results are combined with corresponding weights into a single decision boundary at the output layer.

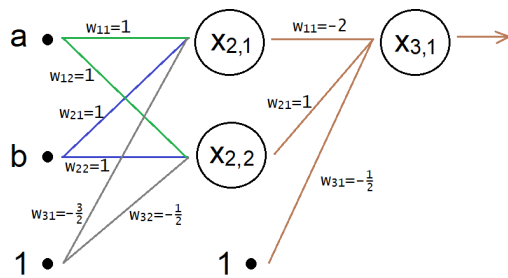


Figure 6: A 3-layer MLP with 2 hidden neurons and assigned weights as suggested by S. Haykin [6].

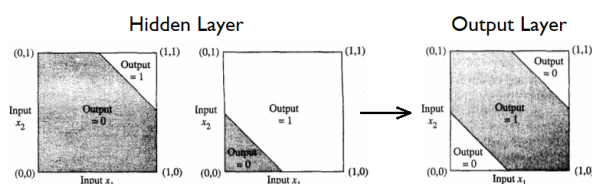


Figure 7: Decision boundaries of the MLP presented in Figure 6. Source: S. Haykin [6].

In the above example the weight matrix contains already precomputed values, however, in reality they ought to be established through training. To efficiently tune the MLP layers, the previously mentioned backpropagation algorithm is applied during the **backward pass** (see Figure 8). The weight update between last two layers is analogous to gra-

dient descent in an LA, however, to modify previous layers, the **chain rule** for derivatives is applied recursively [14]:

$$\frac{\partial E}{\partial w_{ij}^L} = \frac{\partial E}{\partial u_i^L} \frac{\partial u_i^L}{\partial w_{ij}^L} \quad (9)$$

where E is the prediction error (see equation 2). From this rule the weight update can be derived:

$$\delta_i^k = g^k(y_i^k) \sum_m (w_{mi}^{k+1} \delta_m^{k+1}) \quad (10)$$

$$\Delta w_{ij}^k = \alpha \delta_i^k u_j^{k-1} \quad (11)$$

where k is the considered layer, g^k is the derivative of the f^k activation function, δ_m^{k+1} is the error signal of m^{th} neuron from the next layer, w_m^{k+1} is its associated weight, and u_j^{k-1} is the actual output from the preceding layer.

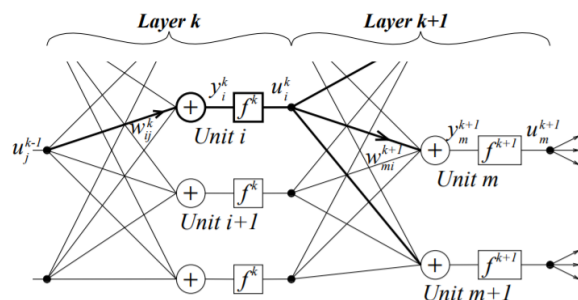


Figure 8: A path of the backpropagation algorithm. Source: CS331 Neural Computing: Lecture Notes

The above formula is utilised in **online** training of an MLP, also known as single-step. Another approach is known as the **batch** mode which, instead of updating model's weights after each sample, adjusts them only once prediction errors for the entire training set are generated:

$$\Delta \mathbf{W}_k = \alpha \mathbf{U}_{k-1}^T \boldsymbol{\delta}_k \quad (12)$$

where \mathbf{U}_i is a $(N \times J_{i-1})$ output matrix from the i^{th} layer (J being the number of neurons in the layer) and $\boldsymbol{\delta}_i$ is a $(N \times J_i)$ prediction error matrix.

A method excluded from consideration in this project is the **mini-batch** which provides an intermediate between online and batch training. The gradient descent algorithm is modified to update edge weights after predicting a sample subset of size N' s.t. $1 < N' < N$.

This type of training was believed to have been theoretically advantageous to online training, however, recent studies [13] have shown the contrary for MLPs employing the backpropagation algorithms. In this project we attempt both approaches in order to ensure an exhaustive exploration of the configuration space.

5 Project Design

The aim of the project is to model the XOR problem using a 3-layer perceptron with **fully connected** layers, i.e. at each layer (except the output layer) all neurons have edges to every neuron in the consecutive layer. To find the best performing configuration, the following parameters of the neural network are tuned:

- M_i - number of neurons in the hidden layer
- epoch_i - number of training iterations
- N_i - size of the training set

Not only is the goal to achieve a high score on the training set but also analyse the performance on the test (unseen) dataset. Moreover, the analysis outcomes ought to be conceptualised in relation to the model's configuration.

To construct the training and test set, first a 2D integer input space is defined where each coordinate can be either 0 (false) or 1 (true). Consecutively, the target (actual output) is appended to each point according to the XOR function

along with the bias (offset) of -1. The dataset is augmented by uniformly sampling from the input space $2N_{\max}$ times. Set's cardinality is defined as above to always select the last N_{\max} records as the test set. The reason for a separation of the dataset into the training and testing subsets is to emulate the process of **leave-p out cross-validation** which allows to evaluate the model's behaviour on unseen data [2].

Finally, zero-mean Gaussian noise with a variance of 0.25 (standard deviation 0.5) is added to the coordinates of every sample. The aim of this process is to improve the model's robustness to outliers and avert overfitting [10]. For a set of 128 observations, an average of 35 samples include a sufficiently high distortion that it results in a divergence between the expected and actual target scalar. In other words, the input coordinates may exceed the 0.5 threshold as a consequence of noise addition with an approximate probability of 0.27.

a	b
0	1
1	0
1	1
0	0

→

a	b	offset	c
0	1	-1	1
1	0	-1	1
1	1	-1	0
0	0	-1	0

→

a	b	offset	c
-0.309	0.898471	-1	1
1.25774	-0.23174	-1	1
1.31959	0.82952	-1	0
-0.08971	-1.02045	-1	0

Figure 9: Three steps of the dataset (here: $N = 4$) generation for the XOR problem.

To assert the correctness of the developed models, the example observations provided in the project's specification can be incorporated as the first 16 vectors. Consistent input for all models and reduction of the execution time is achieved by exporting the data into a CSV which prevents dynamic sample generation.

The project is based in Python 3 and leverages the standard data processing libraries including numpy and pandas. However, none

of the ML toolboxes like `sklearn` or `keras` is adopted due to the inability to sufficiently modify their models, most notably, the bias which value is bound to 1. Moreover, the difficulty in extracting the information necessary for model's evaluation hinders the advantage of utilising a readily available model.

6 Implementation

The model's implementation of the model is based on a multilayer neural network developed by Milo Spencer-Harper. Original source code available at <https://github.com/miloharper/multi-layer-neural-network/>. Despite the base model already incorporating elements of the required functionality such as the sigmoid activation function or backpropagation algorithm, it lacks the inclusion of the bias or the evaluation methods.

For the MLP to take into account the offset, the model is extended to accommodate 3 input neurons instead of the default 2 as well as an additional node in the hidden layer (see Figure 10).

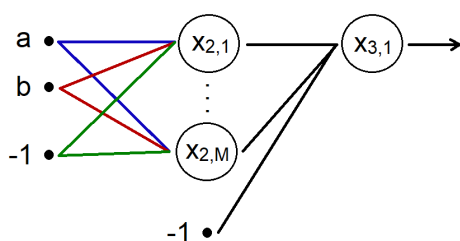


Figure 10: An illustration of the enhanced ANN implemented in this project.

The input layer X_1 consists of 3 nodes (2 neurons + bias) and is bound to the hidden layer of $M + 1$ nodes (M neurons + bias) X_2 with connections represented by a $(M \times 3)$ matrix of weights W_1 . Subsequently, the hidden layer is connected to

the output layer X_3 by a weight matrix W_2 of size $(1 \times M + 1)$.

Because the offset vertex in the second layer is also defined as a hidden neuron, the model must avoid connecting it to the previous input layer but maintain the edge to the output layer. This requires an adjustment to the original backpropagation method, which omits the propagation of the offset error. For the j^{th} sample and final output:

$$y_j = f(\mathbf{w}_2^T \mathbf{v}_{2,j}) \quad (13)$$

The update of W_2 remains the same:

$$\Delta_j^2 = (u_j - y_j) f'(y_j) \quad (14)$$

But the delta on W_1 is defined as:

$$\Delta_{jk}^1 = \mathbf{W}_k^2 \Delta_j^2 f'(\mathbf{V}_{kj}^2) \quad (15)$$

where k range is reduced $[1, M]$, hence, ignores the neuron corresponding to the offset and results in W_2 having dimensions $(M \times N)$ instead of previously incorrect $(M + 1 \times N)$.

To maintain the simplicity of the model, no momentum or regularisation terms are adopted, hence, the MLP lacks the ability to avoid local optima and overfitting. Moreover, a non-adaptive learning rate α is used further improving the comparison stability across model's configurations.

The number of training iterations is variable depending on the training set size N_i . A constant integer K is defined to calculate the number of epochs:

$$\text{epoch}_i = \frac{K}{N_i} \quad (16)$$

Such solution ensures an equivalent number of training samples regardless of the training set size.

Moreover, the online gradient descent learning rule was implemented on top of the batch mode training. The new approach was created simply by invoking the batch training procedure for each training vector, hence, updating the network's weights after considering each sample.

Finally, the target prediction of the MLP is a single real value $t \in [0, 1]$ as defined in equation 13. Although the bounds are not imposed (because the MLP ought to learn the target range), it generally suffices to choose an adequately high learning rate to ensure that model's predictions span the simple domain of XOR, however, the number of iterations is dependant on the weight initialisation. Due to the output resemblance to that of a regressor [10], a question of its best interpretation is introduced. A more informative approach in terms of model evaluation is to use its raw form by treating it as a probability $P_c(t)$ of belonging to the nearest class C :

$$C(t) = \underset{c \in \{0,1\}}{\operatorname{argmin}} (|c - t|) \quad (17)$$

$$P_c(t) = |c - t| \quad (18)$$

On the other hand, the value may be thresholded which produces a binary classification:

$$C(t) = \begin{cases} 1, & \text{if } t \geq \eta \\ 0, & \text{otherwise} \end{cases} \quad (19)$$

where η is the classification threshold which a real constant within the same range as t . In this project the former solution is adopted.

The complete source code is available at: <https://github.com/marektopolewski/mlp-xor>.

7 Testing

Firstly, the testing parameters are defined:

Parameter	Symbol	Value
Epoch constant	K	16,000
Training set sizes	$N = \{N_i\}$	{16, 32, 64}
Hidden layer sizes	$M = \{M_i\}$	{2, 4, 8}
Learning rate	α	1.0

Subsequently, a nested loop is initialised with number of hidden neurons as the outer loop and the training set size as the inner loop. At each iteration a new MLP is constructed with a distinct parameter pair.

The values of both weight matrices between network's layers are chosen randomly but, due to the high sensitivity of the initial assignment [9], the random number generator is seeded to guarantee reproducible results and equivalent starting structure for all models.

As proposed in section 6, models share identical test set and each selects a subset of the same training set according to the N value. Once trained, every model is evaluated in terms of:

- Output map
- Convergence plots
- Accuracy on the test set

For better readability of the analysis, the models will be referred to using the following notation:

learning_mode – hidden_neurons – data_size

For instance, a model with 4 hidden neurons trained using the online mode on 64 training samples has this signature: online-4-64.

7.1 Output maps

The first method is applicable only to models that have at most two features and one target. The XOR function satisfies this condition since it has 2 input arguments (bias can be disregarded since it is constant) and a single real output. To construct the map, square matrix M of size $D + 1$ is filled using the following formula:

$$\forall_{i,j \in [0,D]} M_{i,j} = \text{PREDICT}\left(\left\{\frac{i}{D}, \frac{j}{D}, -1\right\}\right) \quad (20)$$

When plotted as a 2D grayscale image, the dark pixels indicate values close to 0, bright pixels those approaching 1 and grey squares correspond to uncertain classifications with results nearing 0.5.

An acceptable classification will result in an output map having dark pixels forming a triangle in the (0,0) and (1,1) corners but bright triangles at (0,1) and (1,0). However, for an ideal output map, a more strict outcome is expected. Namely, a perfectly black or white pixels can be present only in the very corners of the matrix, remainder of which ought to be calculated on the basis of the four nearest neighbour (4-NN) average. Such definition results in a grey vertical and a grey horizontal line intersecting at (0.5,0.5) from which the pixels gradually approach the values of the closest corners (see Figure 11).

7.2 Convergence plots

Despite being able to illustrate overfitting, the output map evaluation does not provide an objective measure of the model's performance. This, as well as the learning process, can be analysed by examining the temporal changes in the last layer's error. A formula for calculating the mean squared error

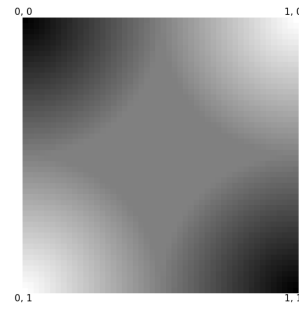


Figure 11: An illustration of an ideal output map for the XOR problem.

(MSE) for each epoch e consisting of n training samples is employed for this purpose:

$$\epsilon_{\text{MSE}}^e = \frac{\sum_{i=0}^{n-1} (\text{PREDICT}(\mathbf{x}_i) - t_i)^2}{n} \quad (21)$$

Having obtained the errors, the data is plotted as a function of the training iteration. Such measure not only allows to visualise the learning efficiency of the neural networks but also whether a given model converges.

7.3 Accuracy on the test set

The final method enables to observe the accuracy on unseen data and provide an alternative performance measure to the visual overfitting analysis offered by output maps. The aggregate error on the test set ϵ is calculated by adopting the mean error (ME) function - a derivation of the previously proposed MSE measure (see equation 21):

$$\epsilon_{\text{ME}} = \frac{\sum_{i=0}^{N_{\text{max}}-1} \sqrt{(\text{PREDICT}(\mathbf{x}_i) - t_i)^2}}{N_{\text{max}}} \quad (22)$$

To further ensure that the obtained MSE is representative of model's performance, the prior placed on the configuration is removed by averaging. Each MLP is evaluated with 50 random initialisation parameters and subsequently a mean of the obtained test errors is returned.

Such process allows to objectively analyse model's efficiency not only on the specific dataset but also for the entire XOR problem.

8 Evaluation

In this section, the empirical results of the tests outlined in the preceding passage will be presented. The models are firstly evaluated on a single dataset, however, later the average performance is discussed in subsection 8.3.

8.1 Output maps

The analysis of output maps produced by the tested models (see Appendix A and B) reveals that none of the networks match the ideal result. Nonetheless, the MLPs with 8 hidden neurons approximate the function fairly well for higher of the dataset sizes but overfit for models batch-16, batch-32 and online-8 by generating complex outcome shapes (see Figure 12). Although outliers are an issue for both learning methods, the single-step mode tends to generalise better for equivalent configurations as demonstrated by more uniform output maps.

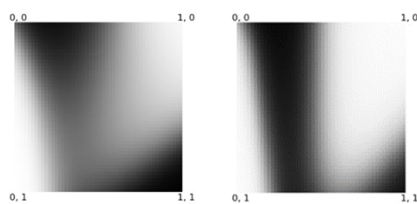


Figure 12: Examples of output maps indicating overfitting. Shape complexity is visible for online-8-16 (left) and even more clearly in batch-8-16 (right).

A significant discrepancy in the output maps between the two training approaches is visible for models with a hidden layer of size 2 and 4. While in the single-step mode the MLPs approximate the underlying function similarly for both training

sets, the batch mode exhibits an opposite property. Namely, networks batch-2-32 and batch-2-64 did not converge which is illustrated by the uniformly grey areas on the left-hand side of the maps. This failure to successfully train the models arises from the problem of fitting a large amount of data onto a simple model, hence, the incapability of finding decision boundaries that satisfy all observations.

Overall, the maps suggest that the ANNs with 4 hidden neurons perform best in terms of noise robustness while model's of size 2 are unable to fit the model onto the underlying data in batch training. Finally, online learning appears to yield more consistent result and output maps resembling the ideal pattern closer compared to equivalent configurations in the other update technique.

8.2 Convergence Plots

The most notable conclusion that can be derived from these graphs is the improved overall stability of the online approach illustrated by the smooth functions in Appendix D, as opposed to great variance between consecutive errors for the batch mode in Appendix C. This is especially prominent for a greater number of training samples (see Figure 13), however, also persistent for smaller datasets. Another empirical property exhibited in the convergence plots is the faster learning for the online training implied by the steep decrease of error between epochs.

The downside of the improved stability is the inefficient training illustrated by nearly flat learning curves for online models with 2 or 4 hidden neurons evaluated on large datasets of 32 or 64 samples (see Figure 14). In general, while it

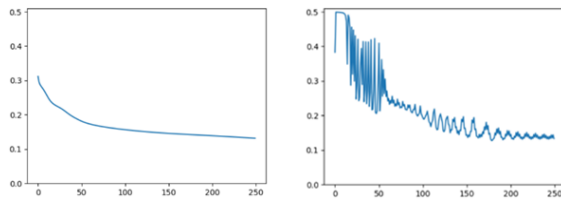


Figure 13: Difference in learning stability between online-8-64 (left) and batch-8-64 (right).

has little influence over batch mode, the learning rate of the single-step approach decreases with the increase of the training samples regardless of the number of neurons. Despite considering a larger set of samples, the models learn for fewer epochs which results in training time insufficient for the underlying dataset.

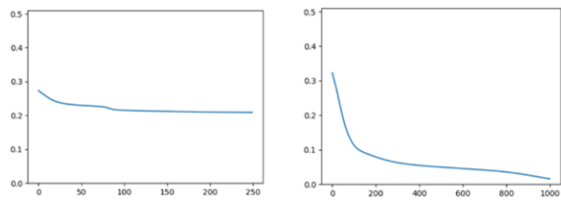


Figure 14: Difference in the rate of learning between online-2-64 (left) and online-8-16 (right).

This evaluation method suggests that the learning process is the more efficient the higher the degree of freedom in the ANN is. Not only are their plots steeper, but they also converge to lower error values. Although this observation is fairly intuitive, the ability to tailor the model accommodates a perfect training set fit introduces the *curse of dimensionality*, i.e. overfitting which was discussed in the preceding testing method. The improved learning in more complex MLPs is characteristic for both training modes but, most notably, in the single-step and 16-sample training for which the prediction MSE is reduced by 60% between the online-2-16 and online-8-16 models.

Another conclusion drawn from the convergence plots is that the final training error is increasing as the dataset size grows. The observation also conforms to the inability to converge during training for batch-2-32 and batch-2-64 presented in subsection 8.1. The correlation is firstly a result of the decline in training iterations and secondly the sample count exceeding the model's generalisation capacity while providing no further insight into the problem [3].

8.3 Accuracy on the test set

The test MSE outcomes are interpreted on a plot presented in Appendix E for the MLP with seeded weight assignment and taught on the dataset considered in previous sections. The numerical results for a series of randomly initialised models trained on fifty newly generated inputs are presented in a table in Appendix G and the corresponding graph in Appendix F.

Firstly, the seeded model with a predefined dataset is discussed. The initial observations imply the unpredictability of the batch mode learning as no obvious correlation between the number of neurons and training set size can be derived for a particular MLP. On the other hand, the test error in online learning is reduced with the increase of training samples for online-2 and online-4 models. For either training technique, the best predictions were made by the ANN with 8 hidden neurons and 64 samples. Even though the model with a hidden layer of size 2 was the most inefficient in both cases, the worst result for batch mode had model batch-2-64 and for single step online-2-16. Moreover, in contradiction to the experimental knowledge [13], the batch learning yielded better results compared to the online training.

The divergence in observed MSE in the specific case and the average one is an indicator of the high correlation between the initialisation parameters, such as the MLP's weights or underlying dataset, and the model's performance. The analysis of the average test error suggests that the batch-2 networks are better than any other MLP for each training sample count while the contrary was true for the predefined ANNs. Another discrepancy in the test suites is the higher efficiency of the online approach than the batch mode which further highlights the importance of this testing scheme.

On the other hand, the correlations between MLPs of a given hidden neuron count persist for online training as opposed to the alternative update approach. Nonetheless, their magnitudes are shifted resulting in the online-2 MLP to have the worst accuracy. For input datasets of size 16 and 64, the models with 8 neurons are favourable, however, for 32 training samples, ANNs with a hidden layer of size 4 were the most efficient.

The test set accuracy proves that it is insufficient to solely rely on the training error when evaluating the suitability of a neural network. Despite the lowest error at the learning phase, the models with 2 hidden neurons underperformed matched against many models with 4 and nearly all with 8 hidden units, however, in none of the scenarios were they the most effective. Such an observation is a prominent example of overfitting and proves how advantageous cross-validation techniques are in model selection.

Overall, the best accuracy across single-step MLPs is achieved by the online-8-64 model ($\text{MSE} \approx 0.17$) and provides an improvement over the worst model online-2-16 ($\text{MSE} \approx 0.29$) of 42% in terms

of MSE. For the batch mode the most effective ANN is the batch-2-64 ($\text{MSE} \approx 0.23$) and the least accurate is the batch-8-16 model ($\text{MSE} \approx 0.36$) yielding a performance boost of 36%.

It is worth noting that despite its superior predicting performance, the online learning models are far slower than their alternative. The mean elapsed time required to train and test a batch MLP is 787ms and 13,709ms for the single-step approach. Although the results heavily depend on the machine hardware equivalent equipment was utilised in for the measurements.

9 Conclusions

The XOR problem is an easily attainable task for today's neural networks, however, should inappropriate architecture be utilised or erroneous dataset is constructed, even a simple function like this can pose a challenge. After conducting an exhaustive analysis, the multilayer perceptron with 8 hidden neurons emerged as the optimal solution as it was able to overcome the overfitting issue by adopting a set of 64 noisy training samples, therefore, achieving sufficient generalisation.

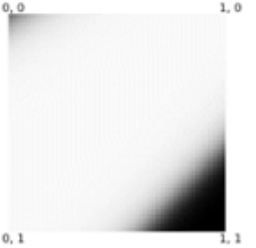
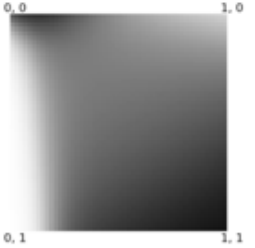
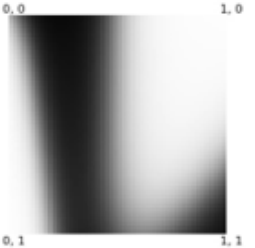
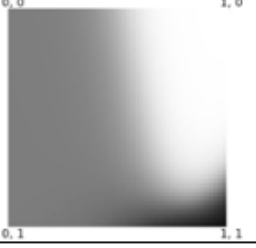
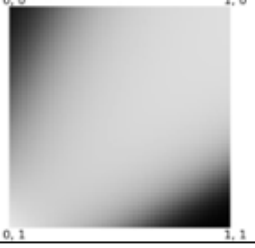
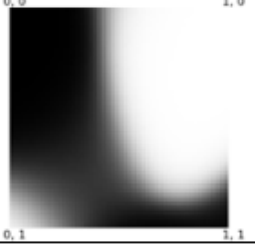
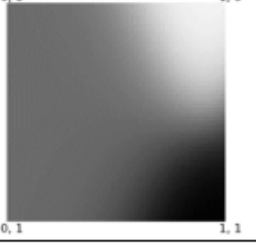
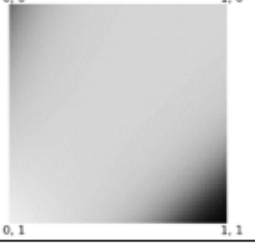
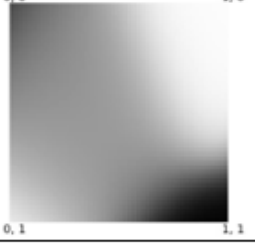
In spite of considering a large variety of parameter configurations, other machine learning models are beyond the scope of the project. To find the best predictor other artificial neural networks or structure such as Random Forest or Ridge Regression ought to be tested. Further work may also involve the investigation of other overfitting prevention as K-Fold cross-validation or early stopping algorithms [10]. Finally, the learning process may leverage the use of adaptive learning rate or different variants of the learning rule, e.g. Adam or RMSProp [5].

References

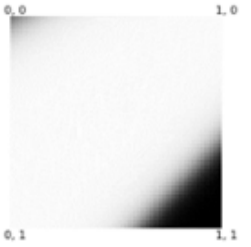
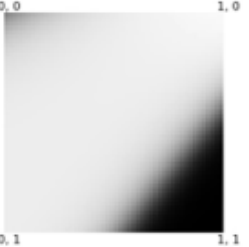
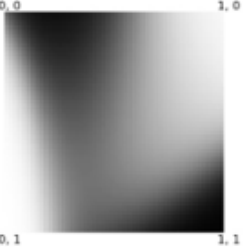
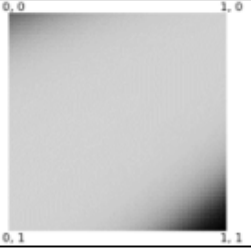
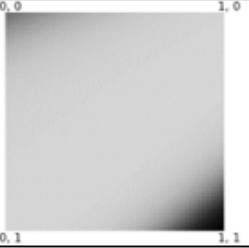
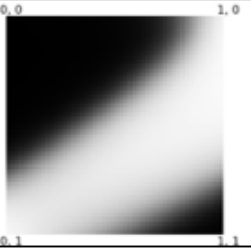
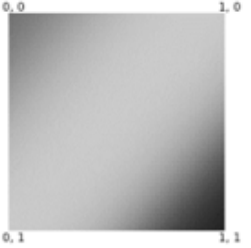
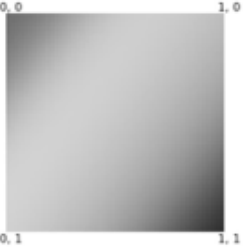
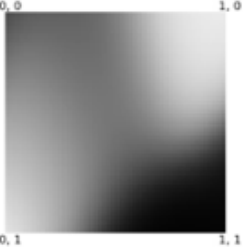
- [1] J. A. Anderson, "A Simple Neural Network Generating a Interactive Memory," *Mathematical Biosciences*, pp. 197–220, 1972.
- [2] C. M. Bishop, "Pattern Recognition and Machine Learning," *Springer*, pp. 240–241, 2007.
- [3] J. Brownlee. "Impact of Dataset Size on Deep Learning Model Skill And Performance Estimates". [Online]. Available: <https://machinelearningmastery.com/impact-of-dataset-size-on-deep-learning-model-skill-and-performance-estimates/>
- [4] ——. "What is the Difference Between a Batch and an Epoch in a Neural Network?". [Online]. Available: <https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/>
- [5] K. Diederik and J. Ba, "Adam: A method for stochastic optimization," *arXiv*, 2014.
- [6] S. Haykin, "Neural Networks - A Comprehensive Foundation," *Macmillan, New York*, p. 197, 1994.
- [7] A. N. Kolmogorov, "On the Representation of Continuous Functions of Several Variables by Superposition of Continuous Functions of one Variable and Addition," *Doklady Akademii. Nauk USSR*, pp. 679–681, 1975.
- [8] M. Minsky and S. Papert, "Perceptrons: An Introduction to Computational Geometry," *MIT Press*, 1969.
- [9] P. E. H. R. O. Duda and D. G. Stork, "Pattern Classification," *Wiley-Interscience*, pp. 282–293, 2000.
- [10] S. Rogers and M. Girolami, "A first course in Machine Learning," *CRC Press*, pp. 2–6, 2011.
- [11] F. Rosenblatt, "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain," *Cornell Aeronautical Laboratory*, p. 386–408, 1958.
- [12] P. J. Werbos, "Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences," *Harvard University*, 1975.
- [13] D. R. Wilson and T. R. Martinez, "The general inefficiency of batch training for gradient descent learning," *Fonix Corporation and Brigham Young University*, 2001.
- [14] R. Wilson, S. Clippingdale, and T. Atherton, "Neural Computing Notes," *Department of Computer Science, University of Warwick*, p. 54, 1996.

Appendices

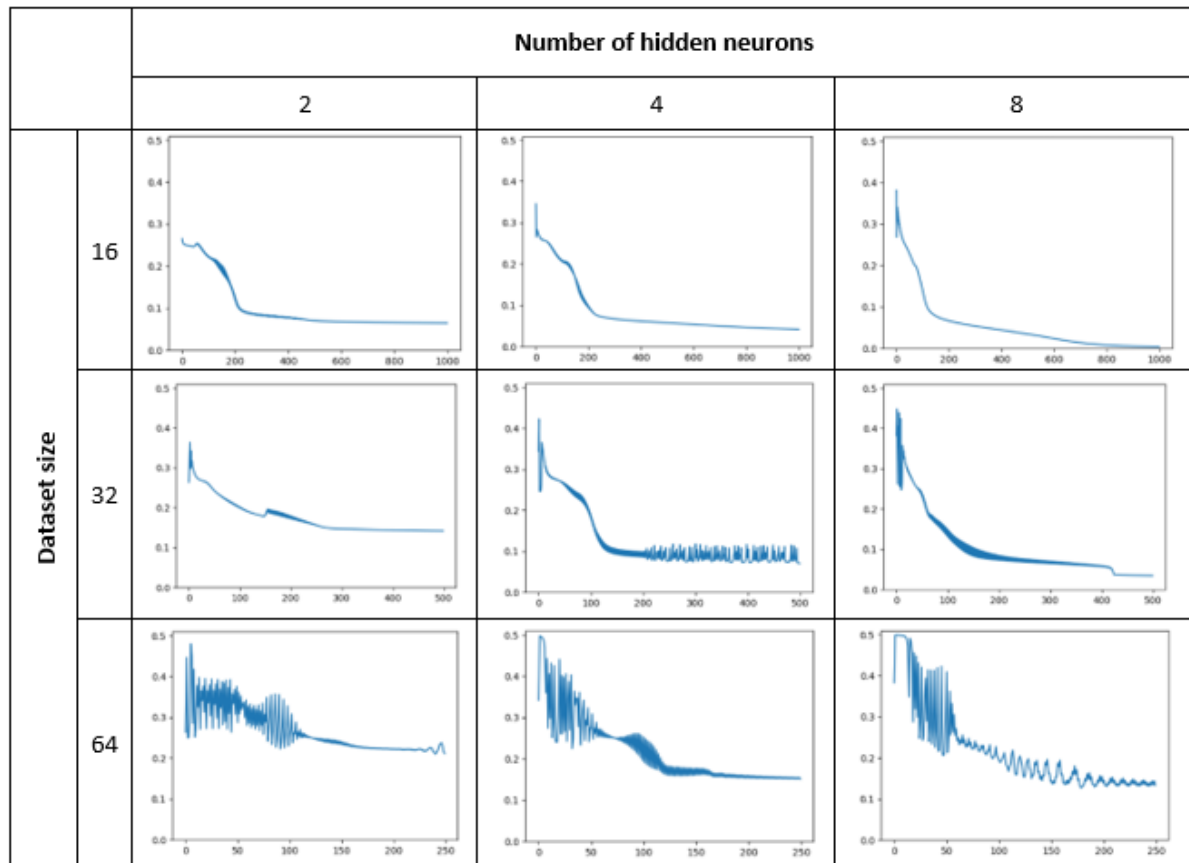
A Output maps for batch mode and selected parameters.

		Number of hidden neurons		
		2	4	8
Dataset size	16			
	32			
	64			

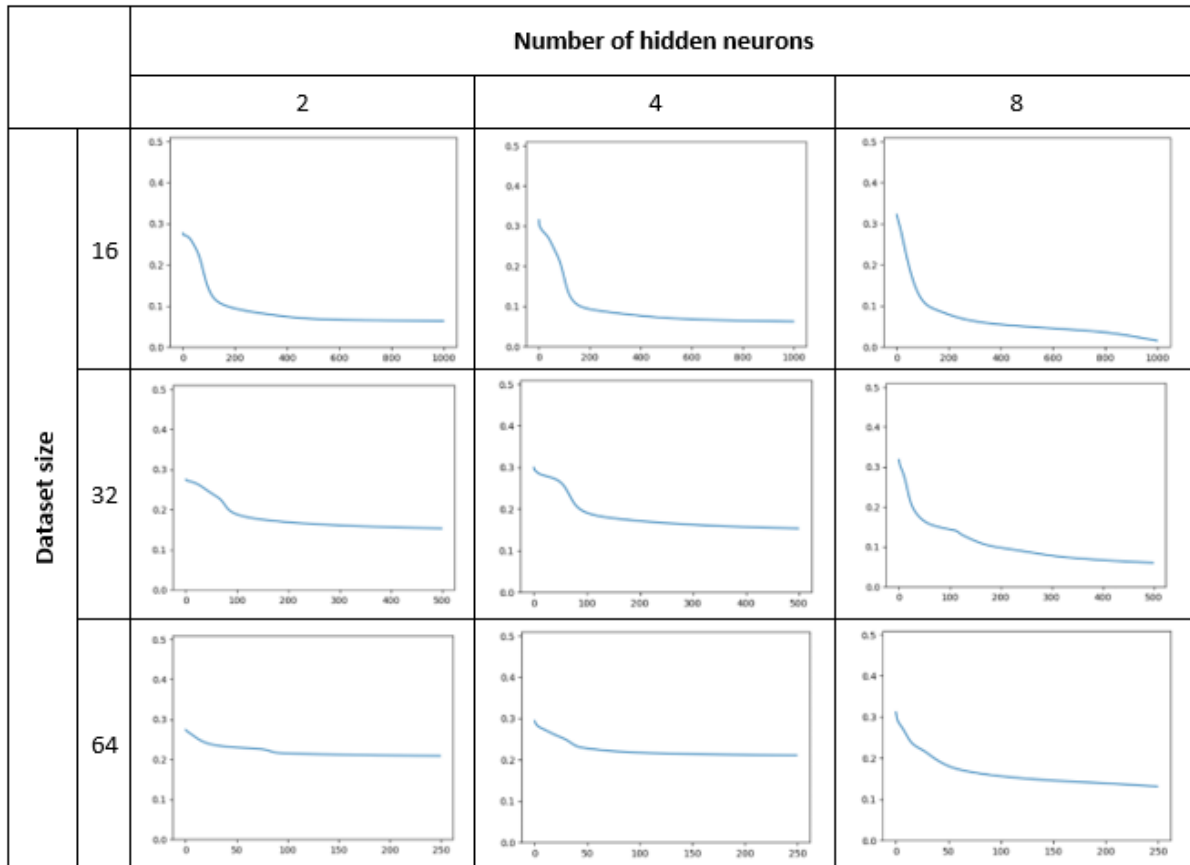
B Output maps for online mode and selected parameters.

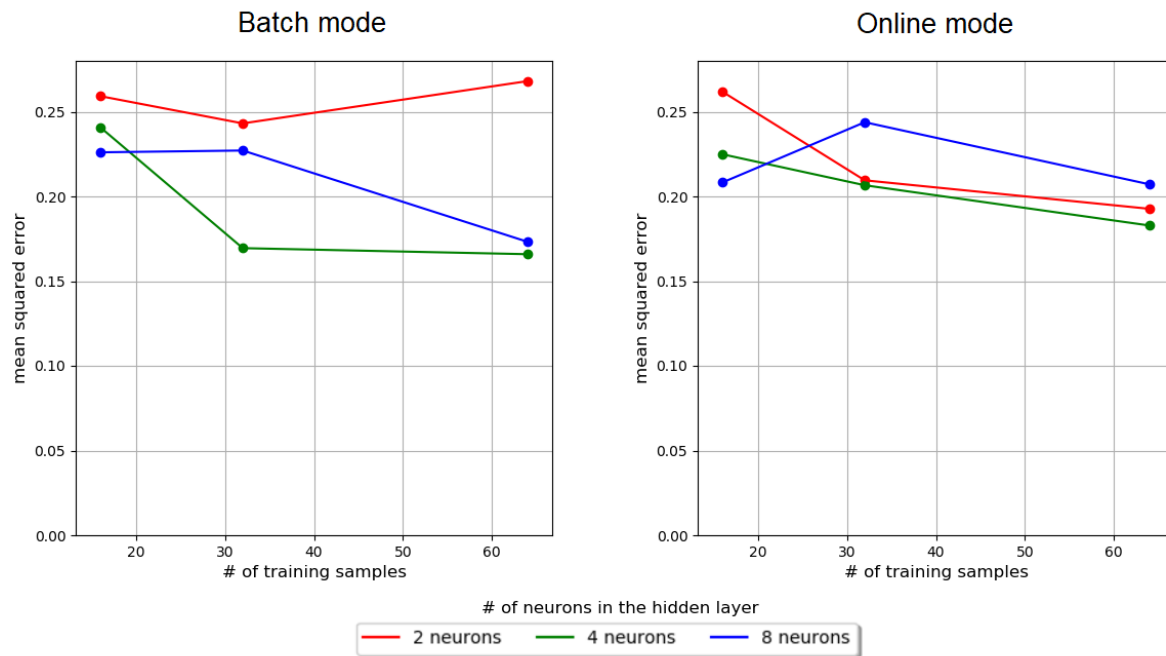
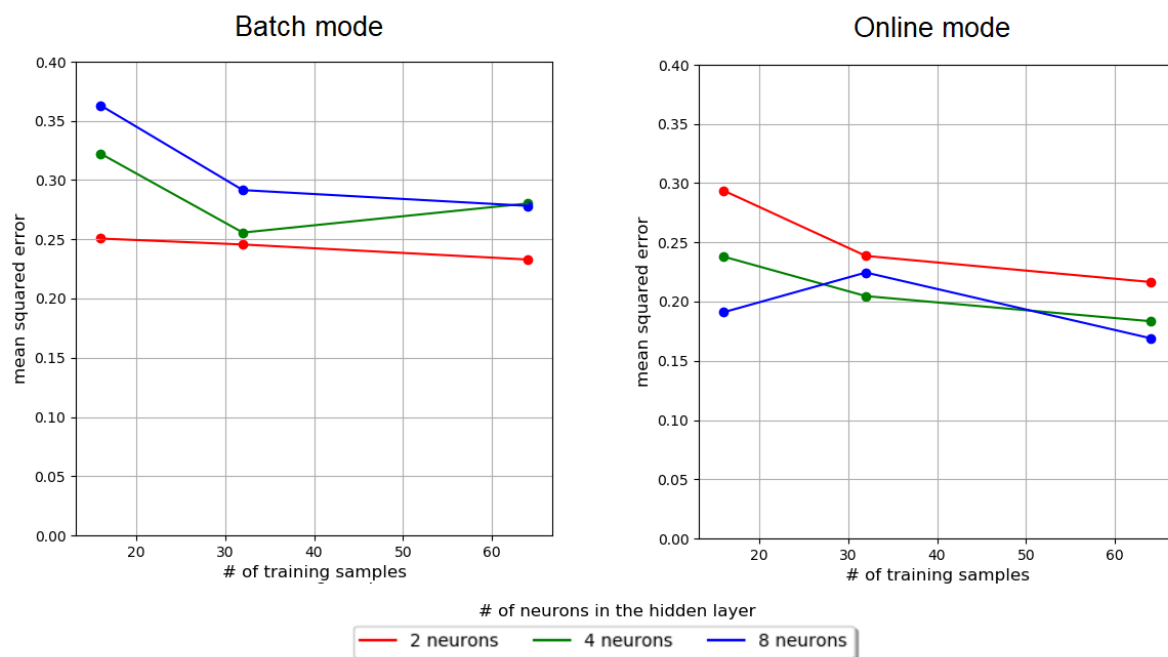
		Number of hidden neurons		
		2	4	8
Dataset size	16			
	32			
	64			

C Learning error plots for batch mode and selected parameters.



D Learning error plots for online mode and selected parameters.



E MSE on the test set for MLPs in batch and online mode.**F Average MSE on a random dataset for all model configurations.**

G Average MSE for all model configurations.

Hidden Neurons	Data Size	Learning Mode	Average MSE
2	16	Batch	0.250660
2	32	Batch	0.245627
2	64	Batch	0.232955
4	16	Batch	0.322387
4	32	Batch	0.255693
4	64	Batch	0.280266
8	16	Batch	0.363117
8	32	Batch	0.291580
8	64	Batch	0.278272
2	16	Online	0.293812
2	32	Online	0.238578
2	64	Online	0.216558
4	16	Online	0.238012
4	32	Online	0.204649
4	64	Online	0.183437
8	16	Online	0.191007
8	32	Online	0.224559
8	64	Online	0.169069