# COMP 30230 Connectionist Computing

Programming / Research Assignment

*December 2025*

## TABLE OF CONTENTS

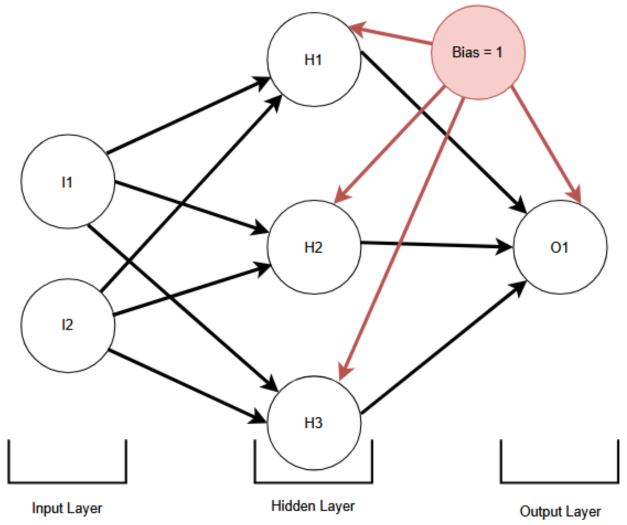
## 1 INTRODUCTION

Artificial neural networks are often made using high-level libraries which prevent users from fully understanding how they work. In this project, I was required to make a basic neural network, a Multi-Layer Perceptron (MLP), from scratch, without using any machine learning or neural networks library. The purpose of this assignment is therefore to understand how learning happens, by observing the concrete effects of forward propagation, backpropagation and weight updates.
This report will discuss the effects of different hyperparameters such as the learning rate, the number of epochs or the number of hidden units, on learning, by analysing data obtained from training the model. Two tasks were investigated: solving the XOR problem and approximating the function $\sin(x_1 - x_2 + x_3 - x_4)$.

## 2 METHODS

### 2.1 NETWORK ARCHITECTURE

The network is composed of three layers: An input layer that receives the data, a hidden layer whose number of units varies, and an output layer that produces the final prediction. Each neuron is connected to every neuron in the next layer, and each connection has an associated weight that determines its influence on the network. In addition, there is a bias unit with a constant value of 1. It is connected to all neurons in both the hidden and output layers, each connection having its own weight.



*MLP with bias and 3 hidden units*

### 2.2 FORMULAS

#### 2.2.1 Forward Propagation

To compute the input of each neuron during forward propagation, the linear combination of its inputs is first evaluated using:

$$z_j = \sum_i w_{ji} x_i + b$$

Where $j$ represents the current neuron, $i$ the indexes of previous neurons, $w_{ji}$ is the weight connecting neuron $i$ to neuron $j$, $x_i$is the value of the input coming from neuron $i$ and b is the bias

The value $z_j$ is then passed through one of the following squashing functions to produce the neuron's activation $y_j$:

- Sigmoid: $y_j = \frac{1}{1+e^{-z}}$
- Tanh: $y_j = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
- Linear: $y_j = z$

Sigmoid is not used for the sine problem because it produces a number in [0,1] while a number in [-1,1] is needed.

#### 2.2.2 Weights Initialisation

To initialise the weights, the formula $W = Uniform(-1, 1)$ was first used. Then, as indicated in the assignment presentation video, the weights were initialised using $W = Uniform(-1, 1)/\sqrt{n}$ where n is the number of input units. This scaling helps control of the linear combination computed during the first step of forward propagation.

#### 2.2.3 Backpropagation and weight update

After each example processed during forward propagation, I compute the corresponding value of δ for the gradient, using the simplified derivative of the activation function. Then I store the quantity $\delta_i * x_i$ where $x_i$ is the input of the weight being updated, so that the weights can be modified with the learning rate at the end of the epoch, once all examples have been processed. This update originally consisted of adding the gradient multiplied by the learning rate to the current weight, but this caused an overflow in the SIN task. Consequently, the option to use the mean of the gradient instead of their sum was added.

## 2.3 CODING CHOICES

Python and NumPy were used to simplify operations on the neural network as weights and activations can naturally be represented as matrices. The MLP was implemented as a class with separate methods for forward propagation, backpropagation, weight updates and gradient accumulation. After testing the network with test examples, an accuracy or test error is calculated for each example so that it is possible to compare configurations and hyperparameters.

## 2.4 HYPERPARAMETERS TESTED

Three hyperparameters

| | Learning rate | Hidden units | Epochs |
|---|---|---|---|
| XOR | 0.001, 0.01, 0.1, 1, 2, 5 | 3,4,5,7,10 | 1,10,1000,2000,10000 |
| SIN | 0.05, 0.1, 0.3, 0.5, 1 | 5,6,7,10,12 | 100, 1000, 3000, 10000 |

Different coding configurations were also tested:

- XOR: with weight initialisation $W = Uniform(-1, 1)/\sqrt{n}$ or $W = Uniform(-1, 1)$
- XOR: with gradient update using the sum or the mean of all gradients over the epoch
- XOR: With or without bias
- SIN: with weight initialisation $W = Uniform(-1, 1)/\sqrt{n}$ or $W = Uniform(-1, 1)$
- SIN: with gradient update using the sum or the mean of all gradients over the epoch
- SIN: with tanh activation function for hidden units and linear activation function for output units or tanh activation function for hidden units and output units.

# 3 RESULTS

## 3.1 EXPERIMENT 1: XOR PROBLEM

Several configurations of the XOR task were tested in order to evaluate the influence of bias weight initialisation and gradient update rule. The goal is to identify which components contribute the most to make the learning stable and fast on the XOR problem by computing every configuration with every hyperparameters.

### 3.1.1 Configurations

#### *3.1.1.1 Effect of bias*

Files: XOR.txt and XOR_wo_bias.txt -> weight initialisations / sqrt(n), gradient update using the mean of all gradients over the epoch

Summary of training at the end of txt files.

| Bias? | Frequency of Accuracy=1 |
|---|---|
| Yes | 24% |
| No | 19.33% |

Files: XOR_updates_no_mean.txt and XOR_no_bias_no_mean -> weight initialisations / sqrt(n), gradient update using the sum of all gradients over the epoch

| Bias? | Frequency of Accuracy=1 |
|---|---|
| Yes | 33% |
| No | 30% |

Files: XOR_init_no_sqrt.txt and XOR_no_bias_no_sqrt -> gradient update using the sum of all gradients over the epoch

| Bias? | Frequency of Accuracy=1 |
|---|---|
| Yes | 30% |
| No | 1.33% |

The configurations show no significant differences, except when initialisation is not divided by $\sqrt{n}$

#### *3.1.1.2 Effect of initialisation*

Here, the goal is to determine whether there is a difference between randomly initialising the weights in [-1,1] with or without 1/$\sqrt{n}$ scaling.

Files: XOR.txt and XOR_init_no_sqrt.txt

| $\frac{1}{\sqrt{n}}$? | Frequency of Accuracy=1 |
|---|---|
| Yes | 24% |
| No | 30% |

There are no significant differences.

#### *3.1.1.3 Effect of the mean*

Here, the goal is to determine whether there is a difference between whether there is a difference between updating the weights using the mean of all gradients and using their sum

Files: XOR.txt and XOR_updates_no_mean.txt

| Mean? | Frequency of Accuracy=1 |
|---|---|
| Yes | 24% |
| No | 33% |

No significant differences are observed.

### 3.1.2 Hyperparameters

To analyse the effects of hyperparameters, only the standard XOR configuration with bias, $1/\sqrt{n}$ weight initialisation and gradient updates based on the mean over the epochs (XOR.txt file) is used

#### *3.1.2.1 Number of hidden units*

| Number of hidden units | Number of runs achieving 100% test accuracy |
|---|---|
| 3 | 7 |
| 4 | 8 |
| 5 | 7 |
| 7 | 6 |
| 10 | 8 |

There are no significant differences.

#### *3.1.2.2 Learning rate*

| Learning rate | Number of runs achieving 100% test accuracy |
|---|---|
| 0.001 | 0 |
| 0.01 | 0 |
| 0.1 | 0 |
| 1 | 8 |
| 2 | 13 |
| 5 | 15 |

The results shows that very small learning rate (0.001 -> 0.1) never achieved perfect accuracy, whereas larger values progressively improve performance.

#### *3.1.2.3 Epochs*

| Number of epochs | Number of runs achieving 100% test accuracy |
|---|---|
| 1 | 0 |
| 10 | 0 |
| 1000 | 8 |
| 2000 | 13 |
| 10000 | 15 |

The number of epochs has a clear influence on training: no run succeeds with only one or 10 epochs, while 1000 epochs already allow 8 successful runs. The performance continues to improve with more epochs.

## 3.2 Experiment 2: SIN problem

Several configurations were tested to evaluate the behaviour of the network on the sine function:

- Different activations function:
    - Tanh for hidden units and Linear for output (tanh-linear)
    - Tanh for hidden and output units (tanh-tanh)
- Different gradient update rules (with or without mean), scaled or unscaled weight initialisation and variations in learning rate, hidden units and number of epochs

For this problem, a test error calculated by $\frac{1}{2} * (output - target)^2$ is used

### 3.2.1 Activation functions

SIN_tanh_linear.txt and SIN_tanh_tanh.txt

Bias, mean gradient update and $1/\sqrt{n}$ weight updates are used here.

| Activation functions | Mean test errors | Best test error | Worst test error |
|---|---|---|---|
| Tanh-linear | 0.0127 | 0.00067 | 0.06354 |
| Tanh-tanh | 0.0125 | 0.00037 | 0.12992 |

Overall, the two activation functions show very similar mean test errors. The tanh–tanh configuration shows a slightly better best test error and a worse worst test error, but the difference is not significant.

### 3.2.2 Gradient update: mean vs no mean

When the gradient update is performed without averaging (i.e. using the sum of all gradients over the epoch), the model crashes due to numerical overflow (SIN_updates_no_mean.txt).

No information can be obtained in this configuration.

### 3.2.3 Effect of Initialisation (with and without $/\sqrt{n}$)

With SIN_tanh_linear.txt and SIN_init_no_square.txt

| $/\sqrt{n}$? | Best test error | Test error for HU=5, LR=1, Epochs=10000 |
|---|---|---|
| Yes | 0.00067 | 0.06354 |
| No | 0.00070 | 0.01952 |

The difference is not significant.

### 3.2.4 Hyperparameters

For these analyses, the SIN_tanh_linear.txt will be used

#### *3.2.4.1 Hidden units*

The SIN_tanh_linear.txt file is used.

| Number of hidden units | Mean of test errors | Lowest test error |
|---|---|---|
| 5 | 0.01208 | 0.00127 |
| 6 | 0.01390 | 0.00115 |
| 7 | 0.01292 | 0.00115 |
| 10 | 0.01257 | 0.00135 |
| 12 | 0.01203 | 0.00066 |

The difference is not significant for hidden units between 5 and 10, but the lowest test error obtained with 12 hidden units stands out.

##### *3.2.4.2 Learning rate*

| Learning rate | Mean of test errors | Lowest test error |
|---|---|---|
| 0.05 | 0.02214 | 0.00208 |
| 0.1 | 0.01593 | 0.00105 |
| 0.3 | 0.0115 | 0.00066 |
| 0.5 | 0.00748 | 0.00135 |
| 1 | 0.00644 | 0.00307 |

The mean test errors decrease as the learning rate increase. However, the lowest test error is obtained with a learning rate of 0.3. This learning rate (0.3) also gives the best results for several others hyperparameter combinations (SIN_tanh_linear.txt, Global summary at the end of the file)

##### *3.2.4.3 Epochs*

| Number of epochs | Mean of test errors | Lowest test error |
|---|---|---|
| 100 | 0.03534 | 0.00675 |
| 1000 | 0.00874 | 0.00192 |
| 3000 | 0.00383 | 0.00133 |
| 10000 | 0.00289 | 0.00067 |

The number of epochs has a clear impact on performance even though the improvement slows down beyond 1000 epochs.

### 3.2.5 Overfitting

A simple overfitting check was implemented by monitoring both the training and test errors at specific epochs. In some configurations with a high learning rate (e.g. 5 hidden units, 3000 epochs, learning rate = 1), a slight form of overfitting can be observed: the test error decreases initially but increases again later, while the training error remains low or oscillates.

# 4 DISCUSSION

## 4.1 THE XOR TASK

### 4.1.1 Configuration

Overall, the different configurations tested on the XOR problem do not show significant differences except for one configuration: XOR problem without a bias and the weight initialisation $* 1/\sqrt{n}$. This is unexpected because removing only bias or only weight initialisation $* \frac{1}{\sqrt{n}}$ doesn't show any significant differences. This result could be attributed to the combined effect of lacking both elements at the same time.

### 4.1.2 Hyperparameters

The analysis of the hyperparameters shows that the number of hidden units has little influence on the performance in the XOR task, as all configurations between 3 and 10 units give similar number of perfect test results. Nonetheless, the learning rate plays a much more important role: very small values (0.001-0.01) never achieve perfect accuracy, while larger learning rates progressively improve results. The number of epochs also has a significant effect: one or ten epochs are insufficient for learning, whereas 1000 epochs already show several successful tests and it continues to improve up to 10000 epochs with 15 successful tests. Overall, the XOR task is highly sensitive to learning rate and training duration, but insensitive to the exact number of hidden units.

## 4.2 THE SIN TASK

#### 4.2.1 Configuration

The SIN problem show that most configuration choices lead to similar performance. The tanh-linear and tanh-tanh activation function settings produce comparable means and lowest test errors. Likewise, scaling the initialisation by $1/\sqrt{n}$ does not strongly affect the results.

The only configuration that completely fails is the one without gradient averaging. In this case, the model updates the weights using the sum of all gradients over the epoch, which in the SIN problem corresponds to 400 training examples. This accumulation produces extremely large weight updates, causing an overflow in python. This behaviour does not occur in the XOR task because it contains only four examples, so the gradient sum remains small.

#### 4.2.2 Hyperparameters

Hyperparameter analysis shows that the number of hidden units has only a limited impact on performance, with similar mean test errors between 5 and 10 units and a slight improvement for 12 units. The learning rate plays a much larger role: higher values lead to lower mean test errors, although the best performance is obtained with a learning rate of 0.3. The number of epochs also has a strong effect, with significant improvements between 100 and 1000 epochs, and light but noticeable gains up to 10 000 epochs.

#### 4.2.3 Overfitting

To monitor overfitting, training and test errors were recorded at predefined checkpoints throughout training. For most configurations, especially with low learning rates, both errors decreased together. However, in some cases involving a high learning rate and a high number of epochs, the test error begins to rise again after an initial decrease, while the training error remains low or oscillates. This indicates overfitting, although it is not strong enough to be noticeable while performing tests.

# 5 CONCLUSION

To conclude, the purpose of this project was to build a MLP entirely from scratch in order to gain a concrete understanding of how forward propagation, backpropagation and weight updates work at low level. A second goal was to study the role of hyperparameters and configuration choices through experimentation. Two tasks were used for that: the XOR problem and another problem based on the function $\sin(x_1 - x_2 + x_3 - x_4)$.

Technically, this project offered a deep understanding of how a MLP works internally. Implementing forward propagation, backpropagation and gradient updates by hand clarified how they are computed. It also highlighted how hyperparameters such as learning rate, initialisation and number of epochs directly affect training dynamics and stability.

With the two tasks, several lessons were learned. The bias and the $1/\sqrt{n}$ weight scaling do not individually have a strong impact on testing, but removing both can severely reduce learning, as seen in the XOR experiments. Moreover, gradient averaging is essential for larger datasets, because summing 400 in the sin task produces excessively large weight updates and an overflow. Finally, the learning rate plays a key role in both problems: very small values prevent learning, while larger ones improve it but may introduce oscillations or overfitting. Overall, the comparison shows that the simple problems like XOR are robust to most configurations choices, whereas larger tasks require more careful tuning to remain stable and reach good performances.