MULTI-LAYER PERCEPTRON

Lisanne Wallaard | s2865459

Leiden University, LIACS, Cognitive Modelling

January 4, 2023

Abstract

This paper is about a multi-layer perceptron (MLP), a specific type of artificial neural network. First, a general introduction to artificial neural networks and this specific model will be given. Then the model and the corresponding process, which can be found in the notebook (see Appendix), are extensively explained. Initially, a neural network, consisting of a single perceptron, is built and trained on generated data, *the OR data* and *the XOR data*. After concluding that this single-layer network can not separate non-linear data like *the XOR data*, an MLP with one hidden layer is built. This network, by contrast, works for both *the OR data* as *the XOR data*. The MLP is eventually trained on EEG recordings of a motor imagery task, after some preprocessing of this EEG data. The paper continues by sharing several results, explanations and conclusions obtained in the notebook using multiple figures and a table. It is found that the MLP can predict based on EEG data with approximately 85% accuracy whether a participant is imagining either a hand or a feet movement. However, it is important to keep in mind that this model can not really read minds. The participants were imagining only two movements, which is different from real mind reading, where someone can have any thought. Furthermore, ideas to improve the accuracy and apply the MLP to different problems are mentioned. In the end, the process is summarised and it is concluded that the MLP can be a powerful tool for a binary classification task.

Introduction

Artificial neural networks are a hot topic of artificial intelligence as they are able to learn from experience. These brain-inspired networks can make sense of complex data, which is impossible using our traditional way of analyses (Simoneau & Price, 1998). Figure 1 nicely shows how the different types of neural networks relate to each other.

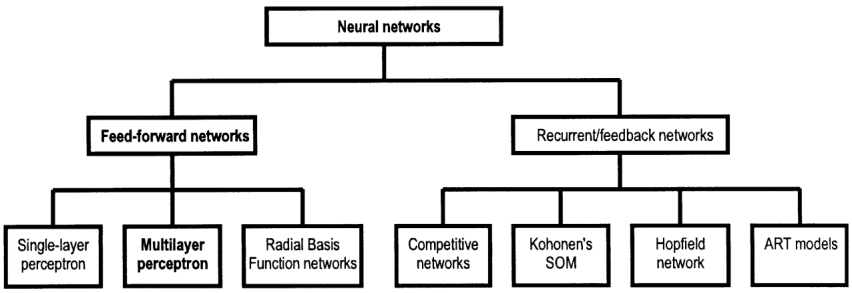


Figure 1. Different types of neural networks (Jain *et al.*, 1996).

The multi-layer perceptron (MLP) is a specific type of artificial neural network, it falls within the feed-forward networks. A perceptron is a node that represents a simplified neuron. It gets a vector of real numbers as input and gives a single real number as output, given a certain activation function. In an MLP, these perceptrons are connected by weights and output signals to perceptrons in other layers. In Figure 2 an example of an MLP is shown.

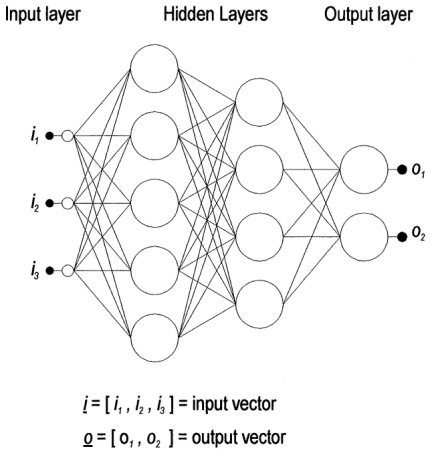


Figure 2. An MLP with two hidden layers, containing respectively 5 and 4 perceptrons (Gardner & Dorling, 1998).

As mentioned before, artificial neural networks learn from experience and are therefore not preprogrammed, but need to be trained. This can be done by using labelled datasets, which is called supervised learning. During training, a certain error signal is used, which is the difference between the predicted and actual label. The size of the error is used to adjust the weights of a network, for which different types of algorithms can be used. A trained MLP is also able to approximate non-linear functions due to their multiple layers with perceptrons. This solves the problem of a single-layer network with one perceptron, which can only do a linear mapping between an input and output vector.

It is interesting to investigate for which applications such an MLP can be used and see how accurate those predictions are. In this paper, it will be explored whether an MLP can predict, based on an EEG signal, which limbs a person imagines to move. Therefore, an MLP will be built and trained on EEG data containing a motor imagery task, where subjects were instructed to imagine moving their hands or feet, given a visual clue. So, this is a binary classification task where the signal belongs either to an imagined hand or feet movement. If the MLP is able to predict the imaginary movement of the correct limb, the MLP in a way is able to read someone's mind with limitations.

Methods

The model is implemented in a notebook, which can be found in the GitHub repository (see Appendix). This notebook can be, for example, opened in Jupyter Notebook or Google Colab. By following the steps in the introduction of the notebook, you should be all set to complete the rest of the notebook. Some libraries, functions and classes will be installed by only running the first code cell (see Listing 1). For this, you will need to press "y" and "Enter" when asked. There could pop up an error when pip or Wget are not installed on your device, in that case you should follow the given tutorials in the notebook.

```
# Install missing package
!pip install mne

# Import packages
import numpy as np                      # algebra
import pandas as pd                     # data manipulation
import matplotlib.pyplot as plt         # plotting
import mne                              # EEG dataset
import scipy                            # scientific computing & signal processing
import pywt                             # wavelet transform
```

```
import random                                   # shuffling lists
import sklearn                                  # machine learning tools (feature selection)

# Download code from the FNCM github repository
!wget --no-cache {"https://raw.githubusercontent.com/clclab/FNCM/main/book/Lab4-materials/EEG.py"}
!wget --no-cache {"https://raw.githubusercontent.com/clclab/FNCM/main/book/Lab4-materials/MLP.py"}

# Import said code
import MLP
from MLP import MLP, plot_errors
import EEG
from EEG import full_epochs, cropped_epochs, EEG_imagery
```

Listing 1. Introduction cell.

The notebook starts by defining two different activation functions, the *Sigmoid* function and the *Binary Threshold* function, of which the formulas and functions are shown in the Figures 3 & 4. The *Sigmoid* function maps every real value into a value between 0 and 1. The *Binary Threshold* function uses a threshold to output the input to either 0 or 1.

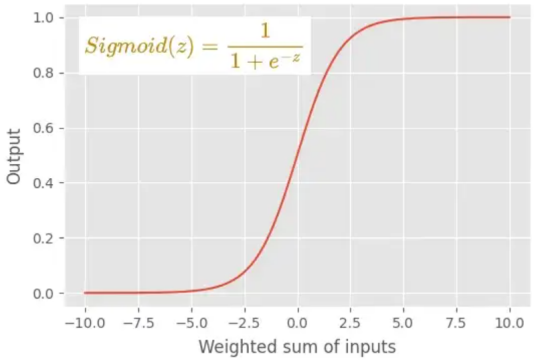


Figure 3. The *Sigmoid* function (Pramoditha, 2022).

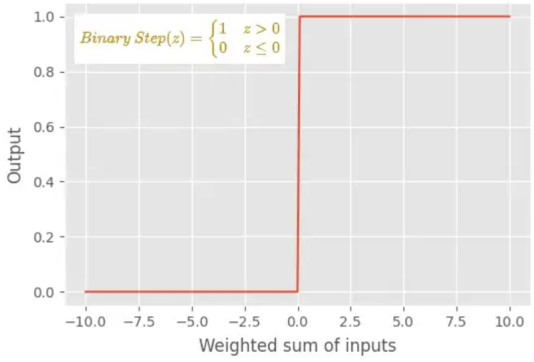


Figure 4. The *Binary Threshold* function (Pramoditha, 2022).

Then the notebook builds a single-layer perceptron using the *Binary Threshold* function. During training, the weights (**w**) and bias (*b*) of this single perceptron are updated according to the following

formulas:

$$\hat{y} = f(\mathbf{w}^T\mathbf{x} + b)$$
$$\mathbf{w}_{\text{new}} \leftarrow \mathbf{w}_{\text{old}} + \eta(y - \hat{y}_{\text{old}})\mathbf{x}$$
$$b_{\text{new}} \leftarrow b_{\text{old}} + \eta(y - \hat{y}_{\text{old}})$$

$\hat{y}$ is the predicted label of a new pattern $\mathbf{x}$ given a certain activation function f. $\hat{y}_{\text{old}}$ is such a prediction based on the old weights and bias. As we see in the formulas the weights and bias are only updated if this prediction is different from the true label y. η is the learning rate and is a value between 0 and 1. The weights are updated, proportionally to the pattern. In order to compute the error of this network, the *mean squared error* (MSE) is calculated given a sample $D = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)\}$:

$$\text{MSE} = \frac{1}{|D|} \sum_{\langle x,y \in D \rangle} (y - \hat{y})^2$$

This single-layer perceptron works for linearly separate data such as *the OR data*, but not for non-linearly separable data like *the XOR data*. Therefore the notebook continues with building a multi-layer perceptron with one hidden layer. In this network, the output of the perceptrons in the previous layer is the input to a perceptron in the next layer. The sum of these weighted inputs activates the node non-linearly by the *Sigmoid* activation function. This enables the MLP to also approximate non-linear data.
The training of an MLP works as follows. The objective function is the *weighted sum of squared errors* (weighted SSE):

$$\text{weighted SSE} = \sum_{\langle x,y \rangle \in D} \frac{1}{\left|\{\langle x', y' \rangle \in D : y' = y\}\right|} (y - \hat{y})^2 = \frac{1}{n} \sum_{(\mathbf{x},y) \in D} (y - \hat{y})^2$$

The prediction of y, $\hat{y}$, is calculated using the *feedforward* algorithm (see Figure 5). Weighted SSE differs from MSE, as it is the sum, not the mean. Thereby it also takes the size of the different classes into consideration such that a model that always predicts 0 (a feet movement) is just as good as one that always predicts 1 (a hand movement). During training, this function is minimalised using the *gradient descent* method:

$$\text{Gradient} = \frac{dE}{dw}$$

The negative of the calculated Gradient shows how the weights should be updated (Roy, 2020). The only problem with this technique is that you can be stuck at a local minimum. The training method backpropagation (see Figure 5) uses the negative of the calculated Gradient to proportionally adjust the weights of all the perceptrons.

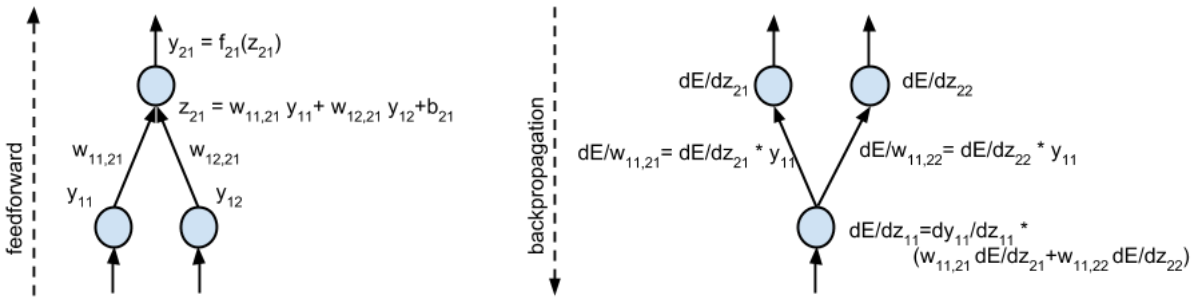


Figure 5. The *feedforward* (left) & *backpropagation* (right) algorithm from the notebook.

First, the MLP will be trained on data generated in the notebook, *the OR data* & *the XOR data*. Finally, the

EEG data, containing the earlier described motor imagery task, will be used. The signals are recorded from 64 electrodes across the scalp. Only the EEG recordings from one participant, consisting of 45 trials (21 for hand, 24 for feet), are used. However, before the training, the raw EEG signals, starting from the point where the participant imagines the movement, need to be preprocessed. In the notebook not all datapoints are used, two features that capture the important information are computed, the temporal and spectral properties.
The temporal properties involve the change of the weight over time. For this property, the y-axis range of the EEG signal is divided into 3 quantiles, which results in an encoding between 0 and 2 for this feature. These values roughly tell you whether there is relative upward (2) or downward movement (0) (see an example in Figure 6). To make sure every quantile approximately contains an equal number of points, the quantiles are calculated individually for each of the 64 channels of every epoch (a specific time frame of an EEG signal).

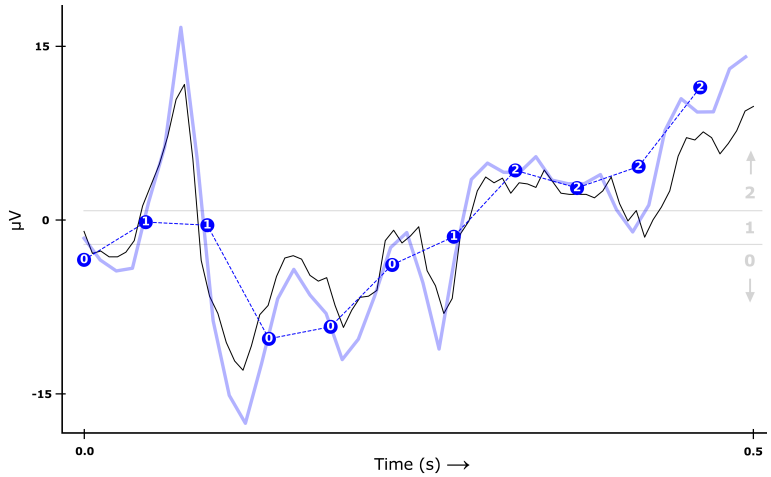


Figure 6. An example of encoding temporal properties of an EEG signal in the notebook. The black line is the raw EEG wave, the blue line is the EEG wave without noise and the blue dots are the average of the blue signal at some evenly distributed points. This results in the following list: [0, 1, 1, 0, 0, 0, 1, 2, 2, 2, 2]. [a]

0 I was not able to add a legend to this plot, because it is in the notebook as a PNG already.

The temporal properties involve the intensity of the signal in different frequency bands: thèta (3.5 - 7.5 Hz), alpha (7.5 - 13 Hz) and bèta (14+ Hz). The relative spectral power density is encoded for each frequency using 0, 1, 2, where 0 represents the frequency band with the least movement and 2 with the most movement (see an example in Figure 7). Across the different trials, the best spectral and temporal features are selected for encoding the signals.

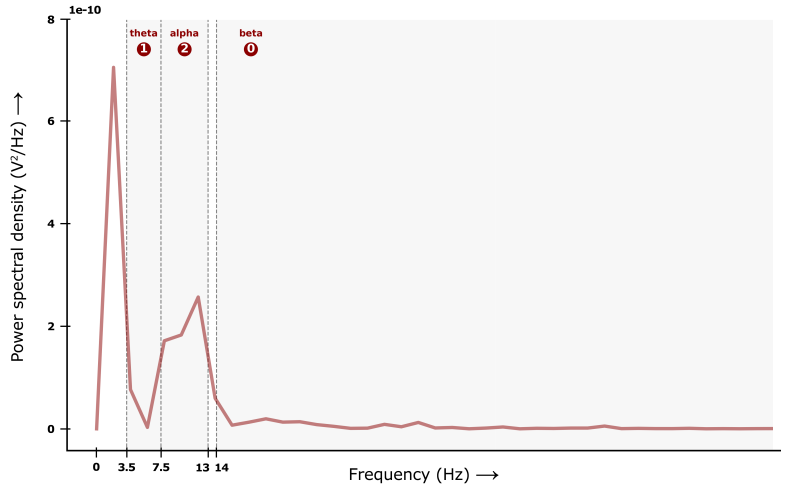


Figure 7. An example of encoding spatial properties of an EEG signal in the notebook resulting in the following list: [1, 2, 0].

This preprocessed dataset is split into a training (2/3) and test (1/3) set. The training set is used to train the network and the test set is used to calculate the prediction accuracy of the trained network on unseen data. This method is called cross-validation.

Results

As mentioned in the section Methods, the MSE formula is used to calculate the error of the single-layer perceptron. In order to calculate the accuracy of this network, there are two possible formulas:

$$\text{accuracy} = \frac{1}{n}\sum_{i=1}^{n} \mathcal{I}_{y_i}(\hat{y}_i), \qquad 1 - \text{MSE}$$

Both formulas iterate over all the elements in the sample. In the MSE formula every true value is subtracted by the predicted value. The outcome of this subtraction is squared and all the outcomes are summed up. So, the results are zero when the prediction is correct, $(0-0)^2$ or $(1-1)^2$. The results are one when both values are different and thus when the prediction is not correct, $(0-1)^2$ or $(1-0)^2$. The accuracy formula, on the other hand, gives one when the predicted and actual value of the sample match, $(0=0)$ or $(1=1)$. The function gives zero when they are not the same and when the prediction, therefore, is wrong, $(0 \neq 0)$ or $(1 \neq 1)$. All these results are also summed up and are the exact opposite of what is calculated in MSE. As $\frac{1}{|D|} = \frac{1}{n}$, accuracy is the same as 1 – MSE on a binary classification task.

Training this single-layer network with *the OR data* results in convergence. The number of iterations differs from only 2 iterations to 9 iterations. These varying numbers of iterations have a different order of input examples before resulting in the correct decision boundary. However, the answer, the correct decision boundary with weights of [0.1 0.1] and a bias of 0, is every time the same. Therefore the answer does not depend on the order of the examples.
On the other hand, training the single perceptron with *the XOR data* does not converge at all. It keeps adjusting the linear plane as it does not find the correct one. That is because the data is not linearly separable as we see in Figure 8. Therefore, the single-layer perceptron model is not able to create a linear plane that can separate the targets of *the XOR data*.

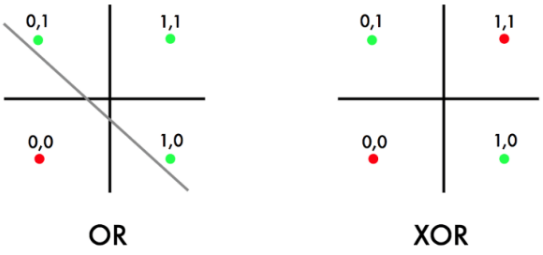


Figure 8. Visualisation of the linear separability of *the OR data* and *the XOR data* (Ahire, 2020).

When training the MLP on *the OR data* using around 400 iterations, the accuracy stabilises at 1. However, when setting a seed, the first accuracy of 1 is found when having a maximum of 206 iterations (see Figure 9). A seed of 1000 is used to be able to reproduce the plots.

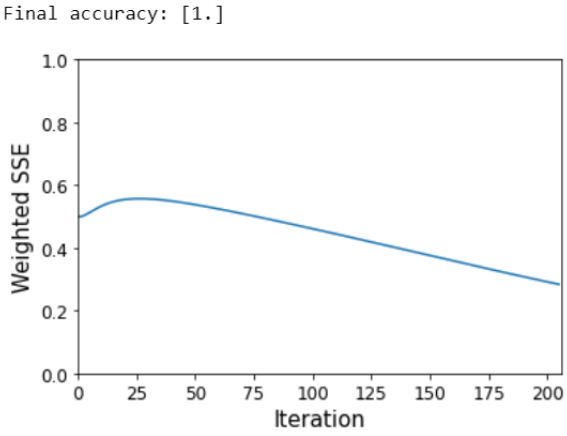
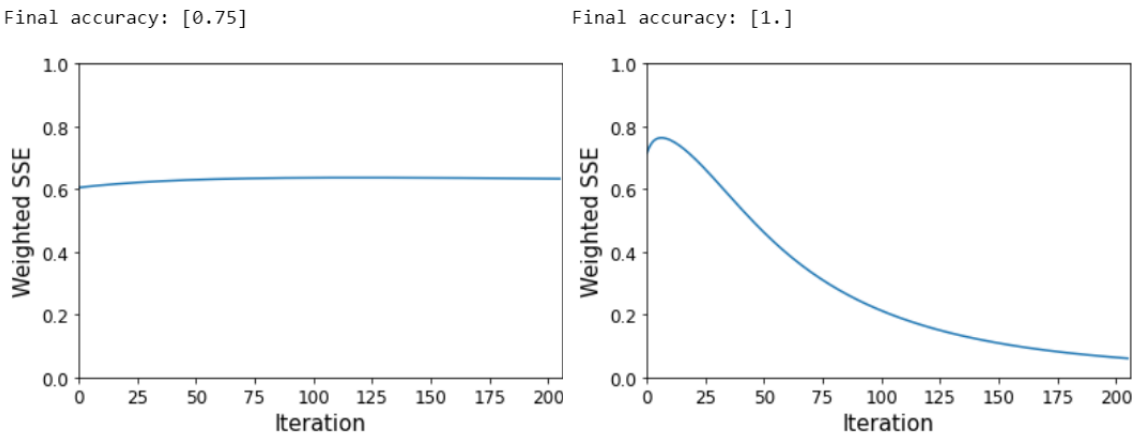


Figure 9. The weighted SSE against the number of iterations (206), when the number of hidden nodes is 5 and the learning rate is 0.2.

In Figure 10a both the learning rate and the number of hidden nodes are decreased. This results in a lower accuracy and a flatter line. In Figure 10b both the learning rate and the number of hidden nodes are increased. This results in an accuracy of 1 again and a steeper line.



(a) The number of hidden nodes is 2 and the learning rate is 0.1. (b) The number of hidden nodes is 25 and the learning rate is 0.3.

Figure 10. The weighted SSE against the number of iterations (206), when changing the number of hidden nodes and the learning rate.

So, after increasing and decreasing the learning rate and the number of hidden layers, it seems the higher the learning rate and the number of hidden nodes, the faster the training converges towards a weighted SSE of 0 and reaches an accuracy of 1.
When the MLP is trained on *the XOR data*, theoretically, just two hidden nodes should be enough (see Figure 11). One node of the hidden layer will operate as an OR gate and the other as a NOT AND gate. If both of these are true, then the output is also true. Indeed, after training an MLP with two hidden nodes a couple of times without seed, an outcome with an accuracy of 1 was found.

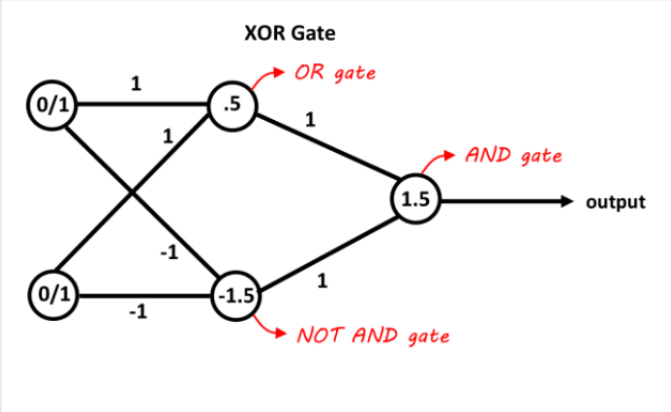


Figure 11. An MLP with one hidden layer containing 2 nodes mimicking a XOR gate (Neel, 2015).

Finally, the MLP is trained on the preprocessed EEG data. When running the training a couple of times with the same settings, different results were found. In the MLP.py file the weights and biases have been initialised using *np.random.uniform()*. As there is no seed, these values are different every time. Therefore, the outcome is different as well every training session.
In Figure 12 the accuracy is shown for different numbers of hidden nodes. The accuracy is an average of ten training sessions. The number of hidden nodes is increased from 0 to 19 with steps of 1. In Table 1 the precise values of the first ten are illustrated. The average accuracy quickly goes up until two nodes and then stabilises around 85%. Sometimes the average accuracy is a bit higher, but it does not exceed 90%. This is quite high considering the model only has one hidden layer. Also, keep in mind that you

do not want to make the network unnecessarily complicated by adding too many nodes without much improvement in accuracy.

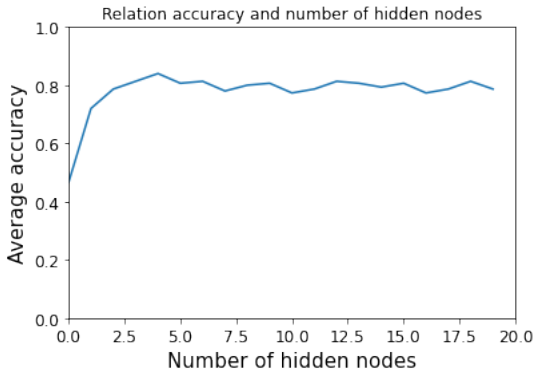
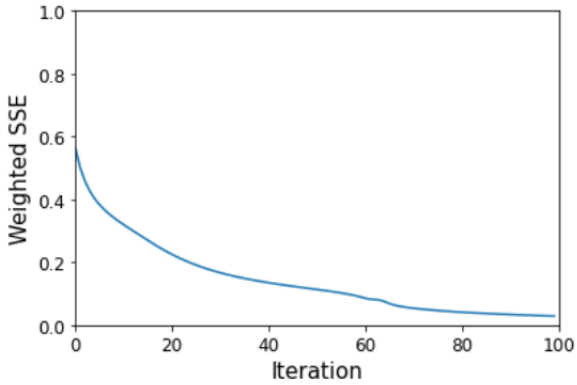


Figure 12. Average accuracy for 20 different numbers of hidden nodes with a learning rate of 0.1 and a maximum of iterations of 100.

Number of nodes	Average accuracy
0	0.39999999999999997
1	0.7933333333333334
2	0.8533333333333335
3	0.8600000000000001
4	0.8866666666666667
5	0.8600000000000001
6	0.8933333333333333
7	0.86
8	0.8333333333333334
9	0.8800000000000001

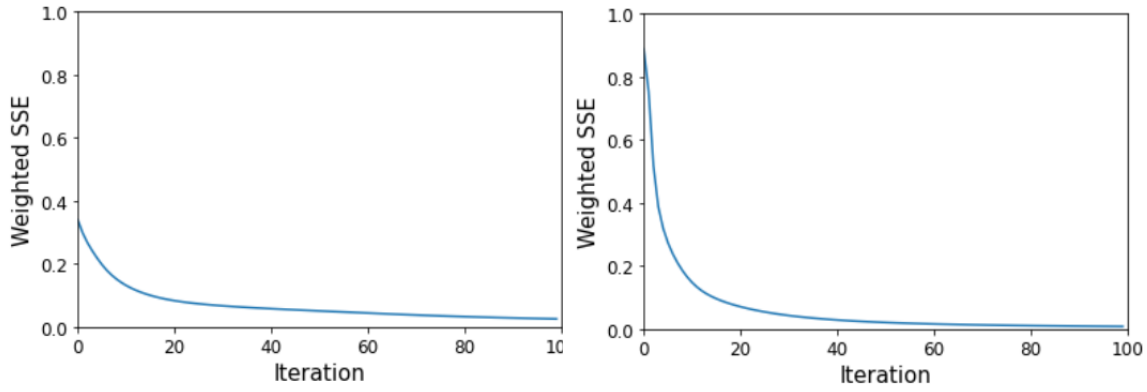
Table 1. Average accuracy for 10 different numbers of hidden nodes with a learning rate of 0.1 and a maximum of iterations of 100.

So, the MLP is trained on the EEG data and in Figure 13 the weighted SSE of this trained network is plotted against the number of iterations. When increasing the learning rate, the accuracy goes up and the weighted SSE decreases faster (see Figure 14a). The same happens when the number of hidden layers is increased separately (see Figure 14b). It seems like the higher both variables, the higher the convergence speed as the weighted SSE drops more quickly.




```
Final accuracy: 0.7333333333333333
```

Figure 13. The weighted SSE against the number of iterations (100), when the number of hidden nodes is 5 and the learning rate is 0.1.



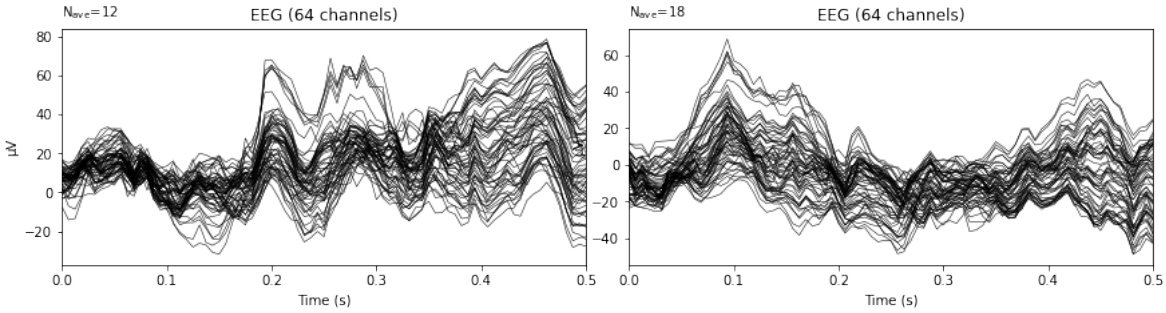

```
Final accuracy: 0.8666666666666667
```

```
Final accuracy: 0.8666666666666667
```

(a) The number of hidden nodes is 5 and the learning rate is 0.2. (b) The number of hidden nodes is 25 and the learning rate is 0.1.

Figure 14. The weighted SSE against the number of iterations (100), when changing the number of hidden nodes and the learning rate.

Although the model works relatively well considering the preprocessing omitted quite some information, it does not achieve 100% accuracy. Therefore, there are still a few misclassified epochs (trials in the test set) when running the MLP on the test data. To investigate why the trained MLP made these mistakes, it is interesting to have a look at the specific elements from the test set that caused the incorrect prediction. For this, it is needed to have an averaged training set epochs for the imagined movements as comparable data (see Figure 15).

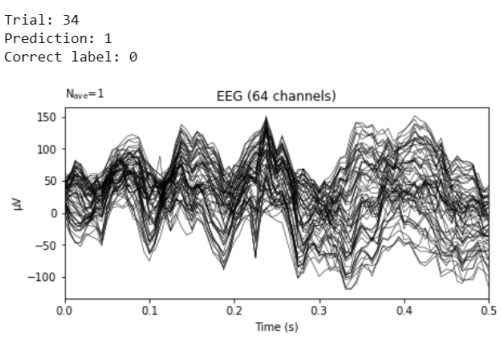


(a) Averaged training set epochs for an imagined feet movement (0).

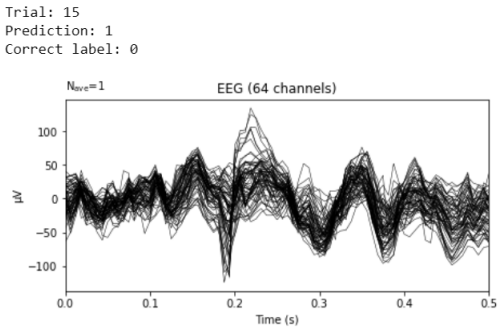
(b) Averaged training set epochs for an imagined hand movement (1).

Figure 15. Averaged training set epochs for imagined limb movements with label 0 or 1.

In Figure 16 two misclassified epochs are shown. Figure 16a, in the end, looks like Figure 15b as the value of the electrical current drops significantly, which correlates more with the imagined hand movement than the correct imagined feet movement in Figure 15a. The same applies to the other misclassified trial in Figure 16b. Also, around 0.3 seconds both the electrical current values in Figure 16 are relatively lower, which matches again the imagined hand movement better than the imagined feet movement as we see in Figure 15. Therefore, it is not surprising that it is difficult for MLP to classify this imagined feet movement correctly.



(a) Trial 34 is falsely predicted to be a hand movement (1).



(b) Trial 15 falsely predicted to be a hand movement (1).

Figure 16. The EEG data of two misclassified trials in the test set from an MLP trained with 5 hidden nodes, 100 iterations and a learning rate of 0.1.

Discussion

Thus, the MLP trained with the preprocessed EEG data is able to predict the correct imagined movement with an accuracy of around 85%, regarding either the hand or the feet of the participants. It may be interesting to explore if this accuracy can be increased when preprocessing the EEG data in a different way. Now, only two features are extracted and roughly encoded. Extracting more features or encoding the features in more detail maybe results in a higher accuracy. Other improvements in the accuracy of the model might be adding more layers, experimenting with different objective functions and/or using the EEG recordings of more than one participant. These things can be areas for future research.

The results may give the impression that humans are able to read minds, but actually, it is very limited. During this experiment, it was known that the subjects were thinking about moving their hand or their feet. Therefore, classifying the EEG data was just choosing between two options (a binary classification task). This is totally different from real mind reading where the thought of the participant can be anything. Training an MLP on every existing thought is impossible, but it could be interesting to see whether the MLP can distinguish more than two classes (multiclass classification). This can be done by recording brain activity of more imagined movements of the human body.

Obviously, the MLP can be trained on different datasets as well and solve other problems. Another direction of future research would therefore be to explore other possible applications for the MLP.

Conclusion

First, in this paper, a network is built containing a single perceptron and trained on generated *OR data* and *XOR data*. After concluding that this single-layer network is not able to separate non-linear data like *the XOR data*, a multi-layer perceptron is built and works for both *the OR data* as *the XOR data*. Then EEG data of a motor imagery task, where participants imagine moving their hands or feet given a visual clue, is preprocessed by extracting roughly the temporal and spatial properties. Although some information of the data is omitted, the MLP is able to reach an accuracy of around 85% on the unseen test data with just one hidden layer. It is pretty exciting that this model can predict which limb movement someone is thinking of without the use of many layers and many hidden nodes. However, we need to keep in mind that it is quite different from real mind reading where the subjects have the freedom to have any thought. Maybe, that is even for the better, because we can ask ourselves if we really want to be able to read someone's mind. Anyway, this paper has shown that the MLP can be a powerful tool to classify binary data. It is exciting to explore what more this model can be used for.

Appendix

Link to the GitHub repository

References

Ahire, J. B. (2020). Demystifying the xor problem. https://dev.to/jbahire/demystifying-the-xor-problem-1blk

Gardner, M., & Dorling, S. (1998). Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. *Atmospheric Environment*, *32*(14), 2627–2636. https://doi.org/https://doi.org/10.1016/S1352-2310(97)00447-0

Jain, A., Mao, J., & Mohiuddin, K. (1996). Artificial neural networks: A tutorial. *Computer*, *29*(3), 31–44. https://doi.org/10.1109/2.485891

Neel. (2015). Training neural networks with genetic algorithms. https://blog.abhranil.net/2015/03/03/training-neural-networks-with-genetic-algorithms/

Pramoditha, R. (2022). How to choose the right activation function for neural networks. https://towardsdatascience.com/how-to-choose-the-right-activation-function-for-neural-networks-3941ff0e6f9c

Roy, A. (2020). An introduction to gradient descent and backpropagation. https://towardsdatascience.com/an-introduction-to-gradient-descent-and-backpropagation-81648bdb19b2

Simoneau, M., & Price, J. (1998). Neural networks provide solutions to real-world problems: Powerful new algorithms to explore, classify, and identify patterns in data. https://www.mathworks.com/company/newsletters/articles/neural-networks-provide-solutions-to-real-world-problems-powerful-new-algorithms-to-explore-classify-and-identify-patterns-in-data.html