# Neural Networks

# Multilayered Perceptron trained with Backpropagation and tested on two sets from the UCI Machine Learning repository

Sonia Grzywacz

April 2023

## Contents

# 1 Problem description

The objective of this project is to implement and train a Multi-Layer Perceptron (MLP) model on the two data sets from the UCI Machine Learning Repository. Data sets that were chosen for the project are:

- The Wine dataset - it consists of 178 samples, each representing a wine with 13 attributes, such as alcohol content, color intensity, and hue. The task is to classify the wines into one of three possible classes based on these attributes.
- The Breast Cancer Wisconsin (Diagnostic) dataset - it consists of 569 samples, each representing a digitized image of a fine needle aspirate (FNA) of a breast mass. The dataset contains 30 continuous attributes (features) computed from the images, which describe the characteristics of the cell nuclei present in the images. The goal is to accurately predict whether a breast mass is malignant or benign based on the 30 computed features.

For this assignment, MLP model was developed from scratch using Python and NumPy. The model will be built using the following components:

1. Sigmoid and Hyperbolic Tangent activation functions

```
class Sigmoid class Tanh

```

2. Fully connected (dense) layers

```
class FullyConnected

```

3. Softmax output layer

```
class Softmax

```

4. Cross-entropy loss function

```
class CrossEntropyLoss

```

5. Stochastic Gradient Descent optimizer with momentum

```
class SGD

```

The model is trained on both datasets using the following parameters:

- Number of hidden layers
- Number of neurons in each hidden layer

- Activation function (either sigmoid or hyperbolic tangent)
- Learning rate
- Momentum
- Batch size
- Number of training epochs

The goal of the project is to train implemented model on chosen datasets and evaluate its performance on a separate test sets to determine the classification accuracy. This should help to understand how MLP model works, how to train it using backpropagation, and how to evaluate its performance on unseen data.

# 2 Data sets description

## 2.1 Wine Dataset

The Wine dataset is a classic **multiclass** classification dataset from the UCI Machine Learning Repository. It consists of **178 samples**, each representing a wine produced in the same region in Italy. The dataset contains 13 **continuous attributes (features)** that describe the chemical composition of the wines. These attributes include:

1. Alcohol content
2. Malic acid
3. Ash
4. Alcalinity of ash
5. Magnesium
6. Total phenols
7. Flavanoids
8. Nonflavanoid phenols
9. Proanthocyanins
10. Color intensity
11. Hue
12. OD280/OD315 of diluted wines

13. Proline

The samples in the dataset belong to one of three distinct classes, corresponding to the three different cultivars (grape varieties) used to produce the wines.

Samples per class: 59,71,48.

Labels are one-hot-encoded for the model output.

The goal is to predict the class of a wine based on its chemical composition.

This classification task can be useful for applications such as quality control, fraud detection, or understanding the relationships between different wine varieties and their chemical properties.

## 2.2 Breast Cancer Wisconsin (Diagnostic) Dataset

The Breast Cancer Wisconsin (Diagnostic) dataset is a **binary classification** dataset from the UCI Machine Learning Repository. It consists of **569 samples**, each representing a digitized image of a fine needle aspirate (FNA) of a breast mass. The dataset contains 30 **continuous** attributes (features) computed from the images, which describe the characteristics of the cell nuclei present in the images. These attributes can be grouped into three categories, with each category containing ten features:

1. Mean values of the following properties:
    - Radius (mean of distances from the center to points on the perimeter)
    - Texture (standard deviation of gray-scale values)
    - Perimeter
    - Area
    - Smoothness (local variation in radius lengths)
    - Compactness ($perimeter^2/area - 1.0$)
    - Concavity (severity of concave portions of the contour)
    - Concave points (number of concave portions of the contour)
    - Symmetry
    - Fractal dimension ("coastline approximation" - 1)
2. Standard error values of the same ten properties mentioned above.
3. Worst (largest) values of the same ten properties mentioned above.

The samples in the dataset are classified into one of **two classes: malignant (cancerous) or benign (non-cancerous)**.

Samples per class: 212 Malignant, 357 Benign.

The goal is to predict whether a breast mass is malignant or benign based on the 30 computed features.

This classification task can be useful for applications such as early diagnosis, treatment planning, and improving our understanding of the relationships between different tumor characteristics and their malignancy.

# 3 Instruction how to use the application

Project folder contains README.md file where the information about how to run the application is provided.
Python 3.6 or higher is required. Open a terminal or command prompt and navigate to the directory containing the source code files. Then execute:

```
python3 -m pip install -r requirements.txt
```

This will run the installation of the following libraries:

```
numpy
scikit-learn
matplotlib
argparse
black
isort
```

Project folder contains four source code files:

*main.py* - main file that execute all steps

*nn.py* - contains all classes that builds NN model (Sigmoid, Tanh, Softmax, FullyConnected, CrossEntropyLoss, MLP)

*optim.py* - contains class for SGD (Stochastic Graadient Decent)

*train.py* - model training function

You can print usage help message to get the information how parameters are named

```
python3 main.py --help
```

The following variables are available to be adjusted:

```
--hidden_layers # A list of integers representing the number of
    neurons in each hidden layer. For example, --#hidden_layers 16
    8 will create a model with two hidden layers, the first one
    with 16 neurons and the second one #with 8 neurons. Default is
    for no hidden layers.
```

```
--activation #The activation function to be used in the hidden
    layers. Options are 'sigmoid' or 'tanh'. Default is 'tanh'.
--learning_rate #The learning rate for the SGD optimizer. Default
    is 0.01.
--momentum #The momentum for the SGD optimizer. Default is 0.9.
--batch_size #The size of each mini-batch for training. Default is
    32.
--epochs #The number of training epochs. Default is 10.
--dataset # wine for the Wine, breast_cancer for the Breast Cancer
    dataset
```

Example usage:

```
python3 main.py --hidden_layers 16 8 --activation tanh --
    learning_rate 0.01 --momentum 0.9 --batch_size 32 --epochs 100
```

If you want to run training on UCI Breast Cancer dataset:

```
python3 main.py --dataset brest_cancer --hidden_layers 16 8 --
    activation tanh --learning_rate 0.01 --momentum 0.9 --
    batch_size 32 --epochs 100
```

The script will load the Wine dataset (by default), preprocess it, and split it into training and test sets. It will then train the MLP model with the provided hyper-parameters on the training set and print the training loss and accuracy for each epoch. After the training is complete, the script will evaluate the model's accuracy on the test set and print the result. If you want to experiment with different model configurations, you can change the hyper-parameters and run the script again to see how the model performs with different settings.

# 4 Description of the experimental setup - reproducibility of results

The experimental setup in our basic Multi-Layer Perceptron (MLP) model for classification task consists of the following components:

Data: We have used the Wine dataset and Breast Cancer Wisconsin (Wisconsin) from the UCI Machine Learning Repository. Both datasets has been preprocessed, normalized, and split into training and test sets using an 70-30 split.

Model: The MLP model itself, has been built with customizable parameters, including the number of hidden layers, number of neurons in each hidden layer, and activation function (either sigmoid or tanh).

Training: The model has been trained using Stochastic Gradient Descent (SGD) with momentum, with customizable parameters such as learning rate, momentum, batch size and number of training epochs.

Evaluation: After training the model, we have evaluated its performance on the test set by computing the classification accuracy.

To ensure the reproducibility of the results, the following measures have been taken:

Setting random seeds: We have set fixed random seeds for both NumPy and Python's built-in random module. This ensures that the random number generators used in the application produce the same sequence of random numbers for each run, making the initialization of the model's weights and the shuffling of the dataset consistent across different runs.

Consistent data preprocessing: The data preprocessing steps, such as normalization and train-test split, have been performed consistently for every run.

Consistent model architecture and training parameters: The model architecture and training parameters are specified using command-line arguments, ensuring that the same settings are used for each run when comparing results.

By following these measures, the experimental setup ensures consistent results across different runs, allowing for the comparison of model performance and the effects of various hyperparameter settings.

# 5 Presentation and discussion of the experimental results

## 5.1 Activation function - impact on accuracy

In this experiemnts, we compare two MLP activation functions for hidden layers:

1. Sigmoid
2. Tanh

We use two datasets (Wine and Breast Cancer) to evaluate the impact of activation functions on the model performance measured by accuracy on the test set. We use the same MLP architecture and training hyper-parameters as well as random seed for all runs:

- 1 hidden layer with 8 neurons
- 20 epochs
- Batch size of 32
- 0.1 learning rate
- Softmax output
- Cross-entropy loss function

| Dataset | Sigmoid | Tanh |
|---|---|---|
| The Wine | 98.1% | 98.1% |
| The Breast Cancer | 98.8% | 98.2% |

Table 1: Activation function accuracy comparison

The table below presents the results.

Based on the results in Table 1. the impact of activation function is not significant on the model performance. For the Wine dataset, the results are the same, whereas for the Breast Cancer dataset, sigmoid activation for hidden layers achieved slightly better accuracy (by 0.6%), which is probably not significant (1 more correctly classified example out of 171 in the test set).

Tanh activation function has better gradient properties (higher value range) which can be more beneficial for backpropagation in MLPs with more hidden layers and for more complex problems, but in our case, it did not give significant improvements.

## 5.2 Hidden layers - number and size impact on accuracy

We use two datasets (Wine and Breast Cancer) to evaluate the impact of activation functions on the model performance measured by accuracy on the test set. We use the same MLP training hyper-parameters as well as random seed for all runs:

- activation function tanh
- 20 epochs
- Batch size of 32
- 0.1 learning rate
- Softmax output
- Cross-entropy loss function

The tables below presents the results for both datasets.

| No. neurons | 0 | 4 | 8 | 16 | 24 |
|---|---|---|---|---|---|
| 1 layer | 100.0% | 100.0% | 98.1% | 98.1% | 98.1% |
| 2 layers | 100.0% | 96.2% | 92.5% | 98.1% | 98.1% |
| 3 layers | 100.0% | 94.4% | 98.1% | 96.2% | 98.1% |

Table 2: Model performance results measured with accuracy on the Wine dataset

Based on results in Table 2. The problem is trivial, specifically linearly separable. We observe 100% accuracy in the case where there is no hidden layer. In

our model implementation, it means that we don't use activation function and we don't introduce non linearity to our model. When we add more layers or neurons the performance on the test set decreases. It indicates model overfitting, i.e., the model is too complex (has too many parameters) for the problem complexity.

| No. neurons | 0 | 4 | 8 | 16 | 24 |
|---|---|---|---|---|---|
| 1 layer | 97.7% | 98.8% | 98.2% | 97.0% | 97.7% |
| 2 layers | 97.7% | 97.7% | 95.9% | 98.2% | 97.1% |
| 3 layers | 97.7% | 98.2% | 98.2% | 97.1% | 98.2% |

Table 3: Model performance results measured with accuracy on the Breast Cancer dataset

Results on the Breast Cancer dataset were not significant and inconclusive.

### 5.3 Learning rate - impact on accuracy

We use two datasets (Wine and Breast Cancer) to evaluate the impact of learning rate on the model performance measured by accuracy on the test set. We use the same MLP architecture and training hyper-parameters as well as random seed for all runs:

- activation function tanh
- 1 hidden layer with 4 neurons
- 20 epochs
- Batch size of 32
- Softmax output
- Cross-entropy loss function

Graphs below (1, 2, 3 ,4) shows results for different learning rates tested on the Wine dataset.

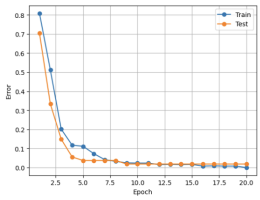


Figure 1: learning rate 0.001

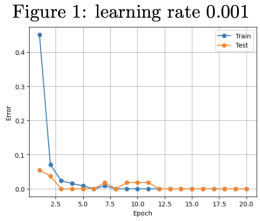


Figure 2: learning rate 0.01

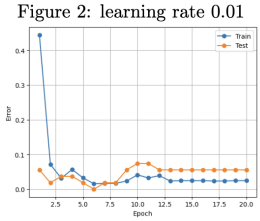


Figure 3: learning rate 0.1

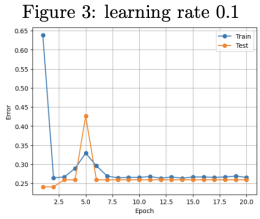


Figure 4: learning rate 1.0

Using the smallest tested learning rate of 0.001, the optimization process is very smooth. Both training and test errors gradually decrease. After increasing the learning rate by a factor of 10 to 0.01, the optimization is more sharp and the training error very quickly achieves 0. However, the test error follows the training error and quickly reaches optimal value. For the next tested value of the learning rate (0.1) the optimization does not end in local minimum because the learning rate value is too large and misses local minimum. Finally, this effect is exacerbated even more with the learning rate of 1.0. It's important to notice the error scale in figure 4, both training and test error saturate at $\sim 25\%$.

Graphs below (5, 6, 7, 8) shows results for different learning rates tested on the Breast Cancer dataset.

Similar conclusions are applicable for the cancer dataset.

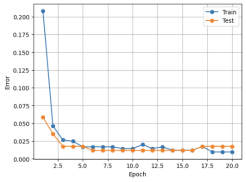


Figure 5: learning rate 0.001

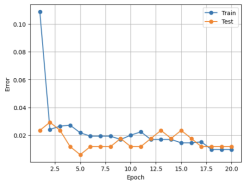


Figure 6: learning rate 0.01

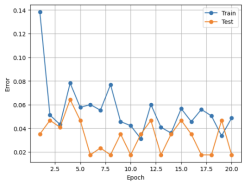


Figure 7: learning rate 0.1

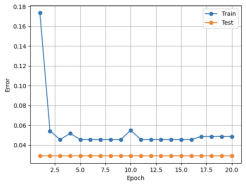


Figure 8: learning rate 1.0

# 6 Conclusions, presumed reasons for success or failure

Implemented model achieved very good results on both tested datasets. Selected datasets are relatively simple tasks for a small MLP model and are widely studied. I was able to achieve perfect (or almost perfect) test accuracy on both

datasets. In the experiments even very poor selection of hyper-parameters related to both training and model architecture yelded relatively good performance.

# 7 Directions for potential future improvement

There are multiple areas of improvement in this project. The first one is related to the input features:

preprocessing: removal of outliers or scaling of feature values insensitive to outliers

feature selection: removal or correlated features or features with very low variance

feature engineering: adding new features based on existing features by combining them

Another area of improvement is related to model selection. In this project the datasets were only split on training and test subsets. The test subset was actually used as a validation set because it was used for hyper-parameter tuning and model selection. Given additional validation set which is separate from the test set we could apply early stopping based on validation sets performance for model selection. In addition the split into training and test set could be stratified to ensure the same proportion of classes in each subset.

To account for uneven number of samples per class (especially for the Breast Cancer dataset), I could consider incorporating class weights into cross-entropy loss function used for model training. Assigning higher weight for the class with lower number of samples (benign) would help to reduce negative impact of the class imbalance or give us a way to improve sensitivity for selected classes if required by a use case.

Finally, we measured model performance using error rate and accuracy. There are other performance metrics more suitable for some applications. We could consider, for example, confusion matrix.