AI Project

Bostan Mihai-Tudor

431F

1. Multi Layer Perceptron(MLP) neural network model for predicting the age of an abalone.

In this project I tried to predict a desired dataset in the domain of regression. The goal is to project and develop a model in order to predict the values for a target variable,which in our case is the number of rings found on the abalone mollusk, based on different input variables that describe various physical characteristics of the mollusk.

The data set I used in this project comes from a public dataset which contains informations about the marine mollusk and includes a total of 4177 examples and 8 features: sex, length, diameter, height, whole weight, shucked weight, viscera weight, shell weight and the rings which they are used to estimate its age.

The goal of the project is to use a neural network in order to predict the age of an abalone from this set of characteristics. This neural network is trained with a training dataset and validated with a test dataset. I also used hyperparameter optimization for the MLP model such as:

-1 or 2 hidden layers chose based on the optimization of the system

-the number of neurons in each hidden layer equal to the half of the previous layer (for example 2 hidden layers with first with 200 neurons and seconds with 100 neurons)

-learning rate of 0.1 and 0.01

The network is implemented using Scikit-Learn library and the learning algorithm used is MLPRegressor, the regression neural network. Also, the GridSearchCV is searching for the best parameters for the neural network while the data is normalized with StandardScaler for model performance. For this performane I calculated the Mean Squarred Error, R-squared, Mean Absolute Error, Explained Variance Score.

In my code, the dataset is divided in 75% of train set used for training the MLP and 25% of test set used for the performance evaluation of the model for new data which was not seen or presented during the training. I chose the 75%-25% because it is a slightly standard practice in ML for relatively big data sets and it ensures a balance between training and validation.

2. Results

In order to categorize the project with having good measures we should look for the following:

-Mean Squared Error(MSE) and Mean Absolute Error(MAE) to be as low as possible

-R^2 and Explained Variance Score to be as close to 1.

Now, for this problem I calculated the minimum, maximum, mean and standard deviation for the number of rings and I got:

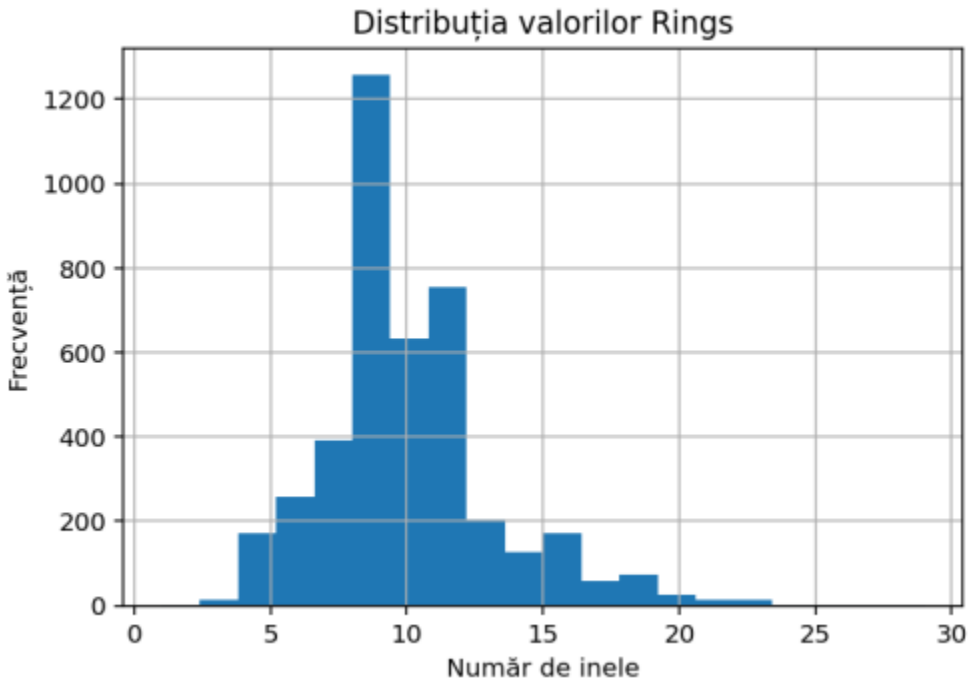
-min = 1

-max = 29

-mean = 9.933(the average number of rings across all abalones in the dataset. On average, an abalone shell has about 9.93 rings)

-std = 3.224(the standard deviation, which measures the spread or variability of the number of rings. A higher standard deviation means that the number of rings varies widely from the mean, while a lower value means it is more consistent)

min	1.000000
max	29.000000
mean	9.933684
std	3.224169



To identify the best configuration for the MLPRegressor model, I used GridSearchCV to explore multiple combinations of hyperparameters. For each combination, it was evaluated using 5-fold cross-validation and performance was measured by the mean squared error (MSE). The goal I want is to find the configuration that minimizes prediction error.

All the parameters tested:

- Hidden Layer Sizes:
  - (100,)
  - (200,)
  - (200, 100)
  - (100, 50)
  - (50, 25)
- Learning Rates:
  - 0.1
  - 0.01

This results in a total of 10 combinations tested. For each combination, the average test MSE was recorded. Below is a summarized version of the top results:

	Hidden Layer Sizes	Learning Rate	Mean Test MSE	Std MSE	Rank
0	(200,)	0.01	4.6465	0.2636	1
1	(100,)	0.01	4.8969	0.6147	2
2	(100, 50)	0.01	4.9631	0.3023	3
3	(50, 25)	0.01	5.0091	0.5956	4
4	(200,)	0.10	5.0490	0.5015	5
5	(100, 50)	0.10	5.2246	0.5218	6
6	(200, 100)	0.01	5.2648	1.0743	7
7	(100,)	0.10	5.3441	1.0752	8
8	(50, 25)	0.10	5.3630	0.7585	9
9	(200, 100)	0.10	6.1547	2.2152	10

I used the mean_test_score from GridSearchCV to select the best model because it represents the average performance of each hyperparameter combination during cross-validation. This ensures that the selected model performs well consistently across different splits of the training

data, reducing the risk of overfitting. By choosing the model with the lowest mean_test_score, I want to find the most generalizable configuration before testing it on unseen data.

In addition to showing the average test error (Mean Test MSE) for each hyperparameter combination, I also included the **standard deviation of the test error** (Std MSE) across the 5-fold cross-validation.

- Mean Test MSE tells how well the model performed on average across the folds.
- Std MSE shows how **consistent** that performance was from one fold to another.

By including Std MSE, I was able to not only select the model with the best average performance but also ensure that the model is **reliable and generalizes well** across different data splits.

A low Std MSE indicates consistent model performance across validation folds, which is critical for ensuring that the model is not just accurate on average but also reliable in different scenarios.

For these results, the best-performing configuration is: hidden_layer_sizes=(200,),

learning_rate_init=0.01 with the following:

```
Performanța modelului ales (în inele):
Mean Squared Error (MSE): 4.1295
R-squared (R²): 0.5982
Mean Absolute Error (MAE): 1.4680
Explained Variance Score (EVS): 0.5985
```

Interpretation:

- The MSE of 4.1295 is significantly lower than the variance of the target variable (standard deviation squared ≈ 10.4), indicating a reasonably accurate model.

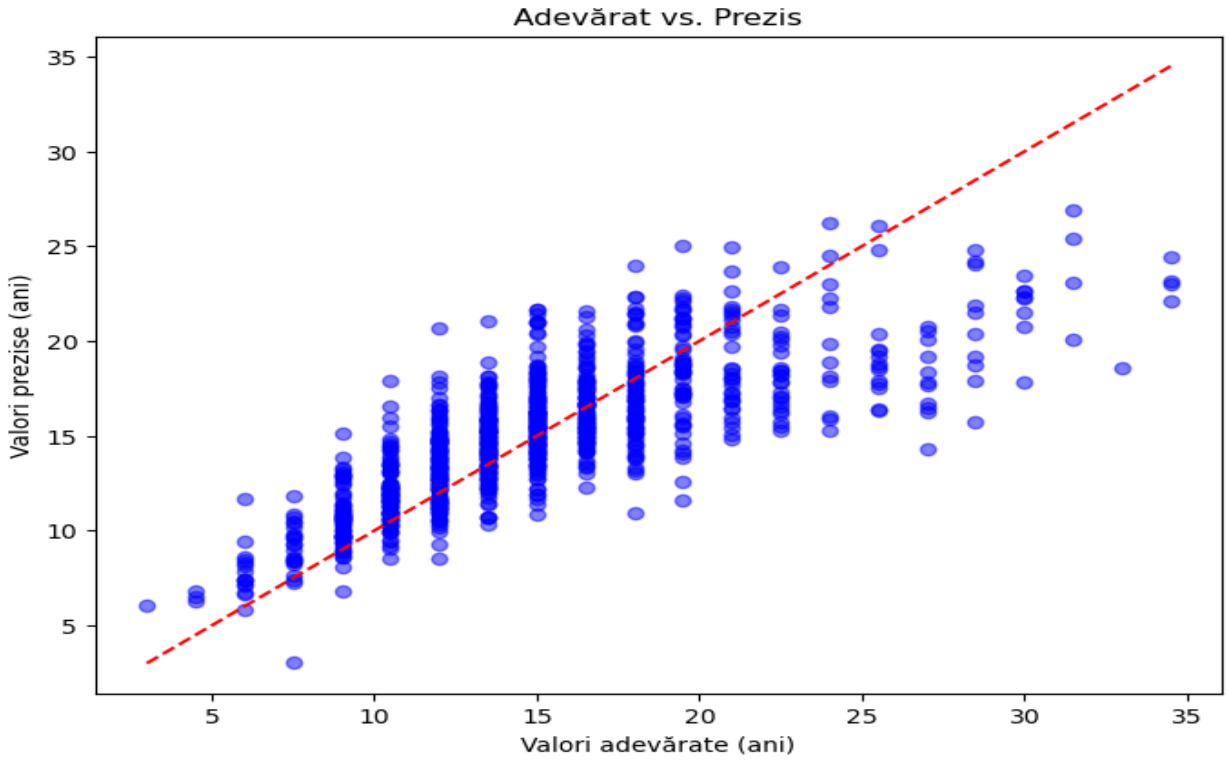
- The MAE of 1.468 implies that, on average, the model's predictions differ from the true number of rings by about 1.47. In terms of years, this corresponds to approximately 2.2 years.

- R^2 and EVS values close to 0.6 indicate that about 60% of the variation in the age is explained by the model. While not perfect, this is a strong result for real-world biological data, which tends to be noisy and complex.

Before training the model, I standardized all feature values using the StandardScaler function. This process rescales each feature to have a mean of 0 and a standard deviation of 1. Standardization ensures that all input variables contribute equally to the training process and helps the neural network converge faster and more reliably.

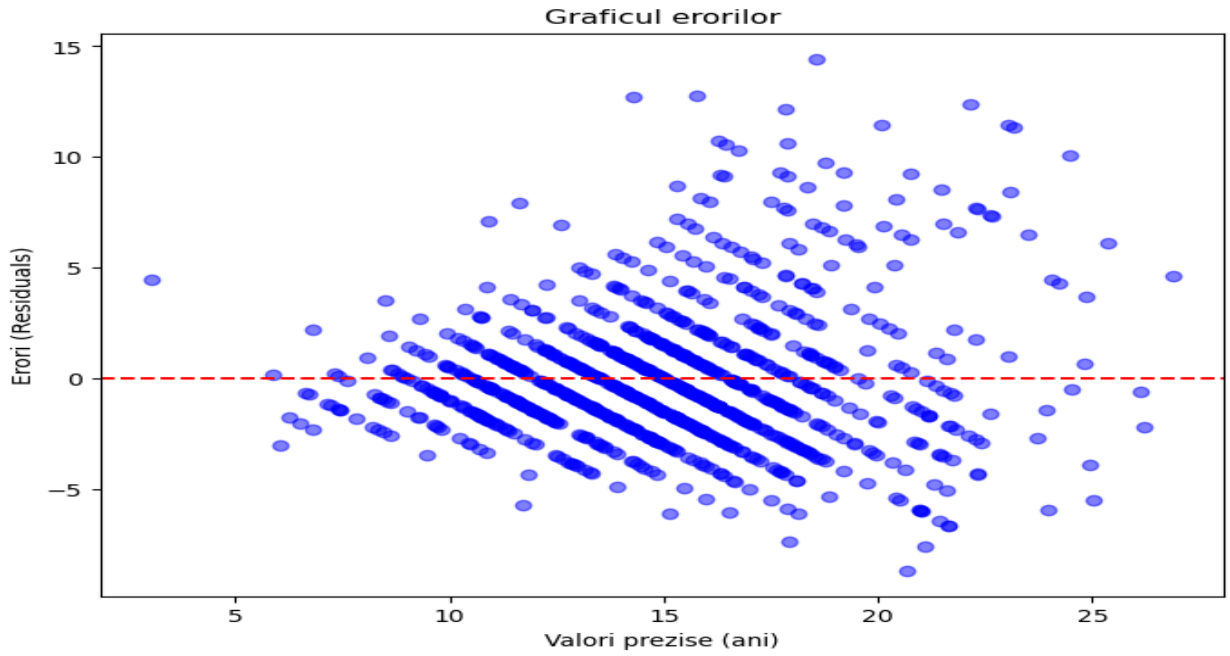
3.Graphs

i. True vs Predicted graph



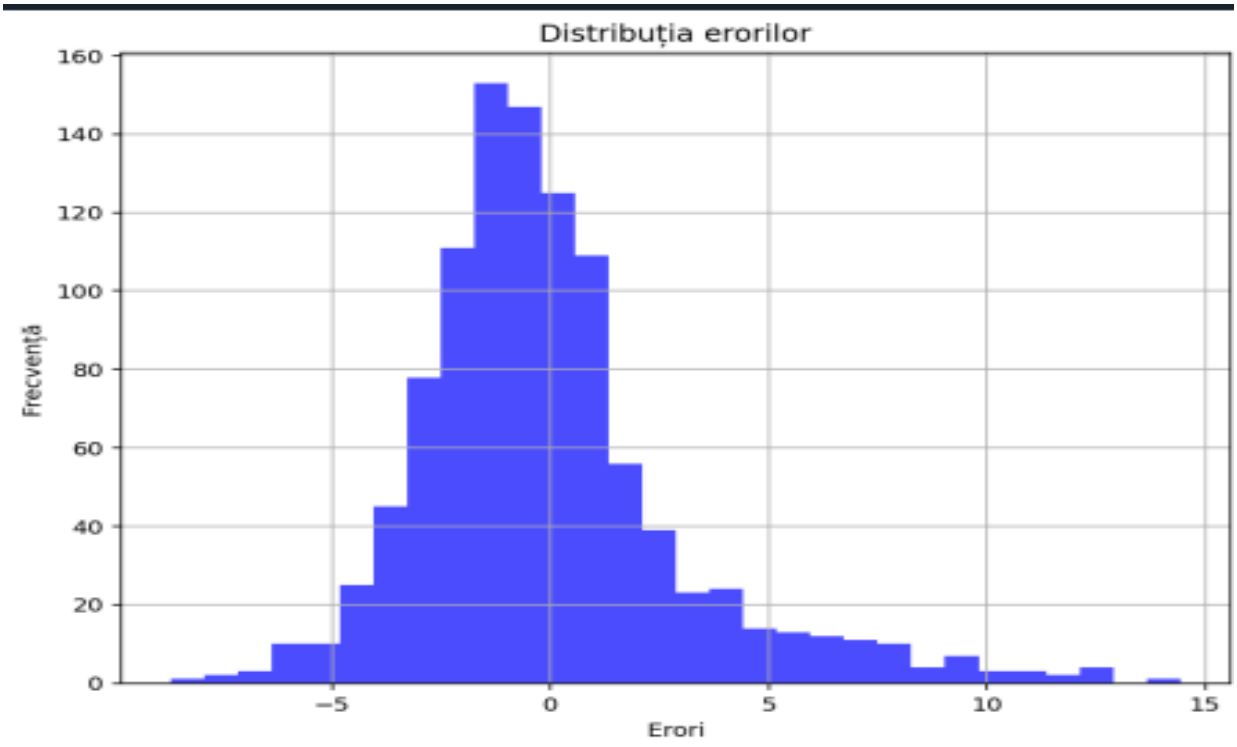
The red line represents the ideal relation between the real and predicted values. A very good model should have the points slightly near the red line and in this model they not vary very far away from it, just in some cases. It shows that this is not the best correlation between datas but as a mean they can predict somewhere near the true value.

ii. Residual Plot



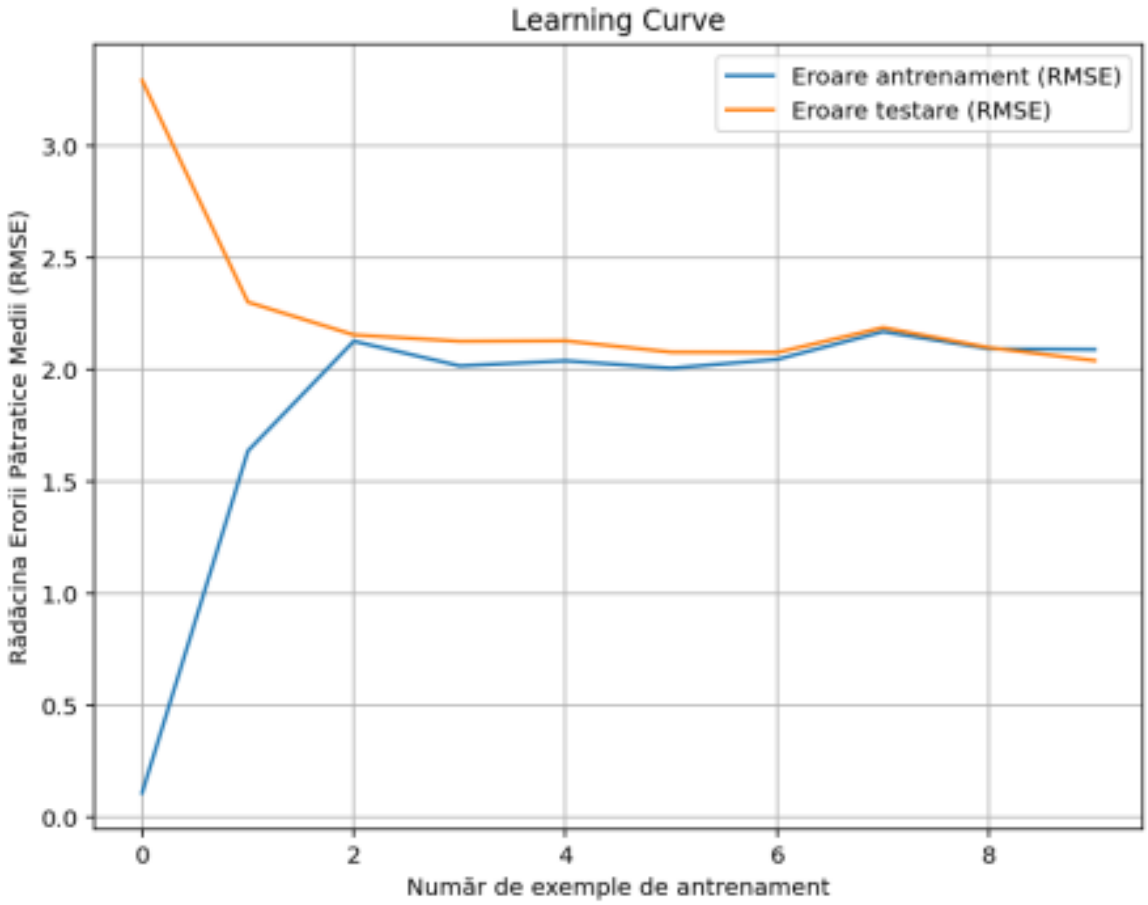
A good system should have the points equally spread around the red line because every point represents an error of every prediction. Here, it is somewhat spread around the line, not in every area, but in the most of the graph.

iii. Error distribution plot



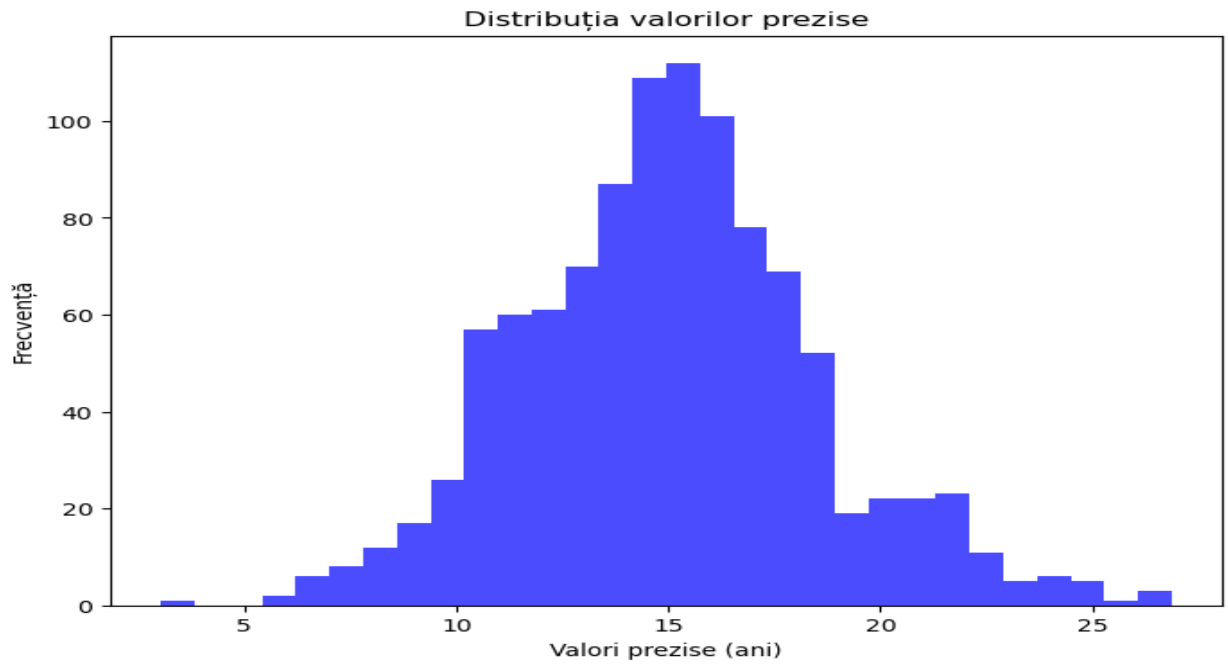
We are looking for a symmetry here around the 0 point, which we can somehow see, it is not the perfect error distribution but we can say that the model is likely to do some precise predictions and some other not that precise. The distribution is approximately centered around zero, indicating no major systematic bias.

iv. Learning curve



As we add training trials, the test errors decrease to a point where they start to stabilize together, which indicates a not-so-good learning rate towards the end, but a good start. We want the learning curve to show a steady decrease in errors as we add training data. The curves converge, suggesting that the model generalizes well and does not overfit.

v. Predicted Values Distribution



This shows the frequency of distribution of different predicted values on training sets and it should cover most of the values in order to have a real prediction based on the real values. The distribution of predicted values shows good coverage of the target range, implying that the model is learning the full structure of the data.

3. Hyperparameters variation

In order to improve the model's performance and ensure that the neural network is well-tuned, I tested multiple hyperparameter combinations using GridSearchCV:

i. <u>hidden_layer_sizes</u>: [(100,), (200,), (200, 100), (100, 50), (50, 25)]

It defines the number of hidden layers and the number of neurons in each layer.

- (100,) means a single hidden layer with 100 neurons.
- (200, 100) means two hidden layers, the first with 200 neurons and the second with 100.
- By testing various architectures (deep vs shallow networks), the model can discover the best structure for learning complex patterns from data.
- Generally, deeper networks (more layers) can capture more complex relationships, but they also increase the risk of overfitting and computational cost.

ii. learning_rate_init: [0.1, 0.01]

This represents the initial learning rate andhow big the steps are when the model updates the weights during training.

- A smaller learning rate (like 0.01) means slower learning but potentially more stable convergence.
- A higher learning rate (like 0.1) can speed up training but may overshoot the optimal weights or cause instability.
- By testing both values, the model chooses the one that allows it to converge to a better minimum error.

iii. max_iter = 1000

This is the maximum number of iterations (epochs) the algorithm is allowed to use during training.

- If convergence is not reached within 1000 iterations, training stops.
- A high enough value is necessary to ensure that the model has time to learn the patterns in the data.

iv. random_state = 150

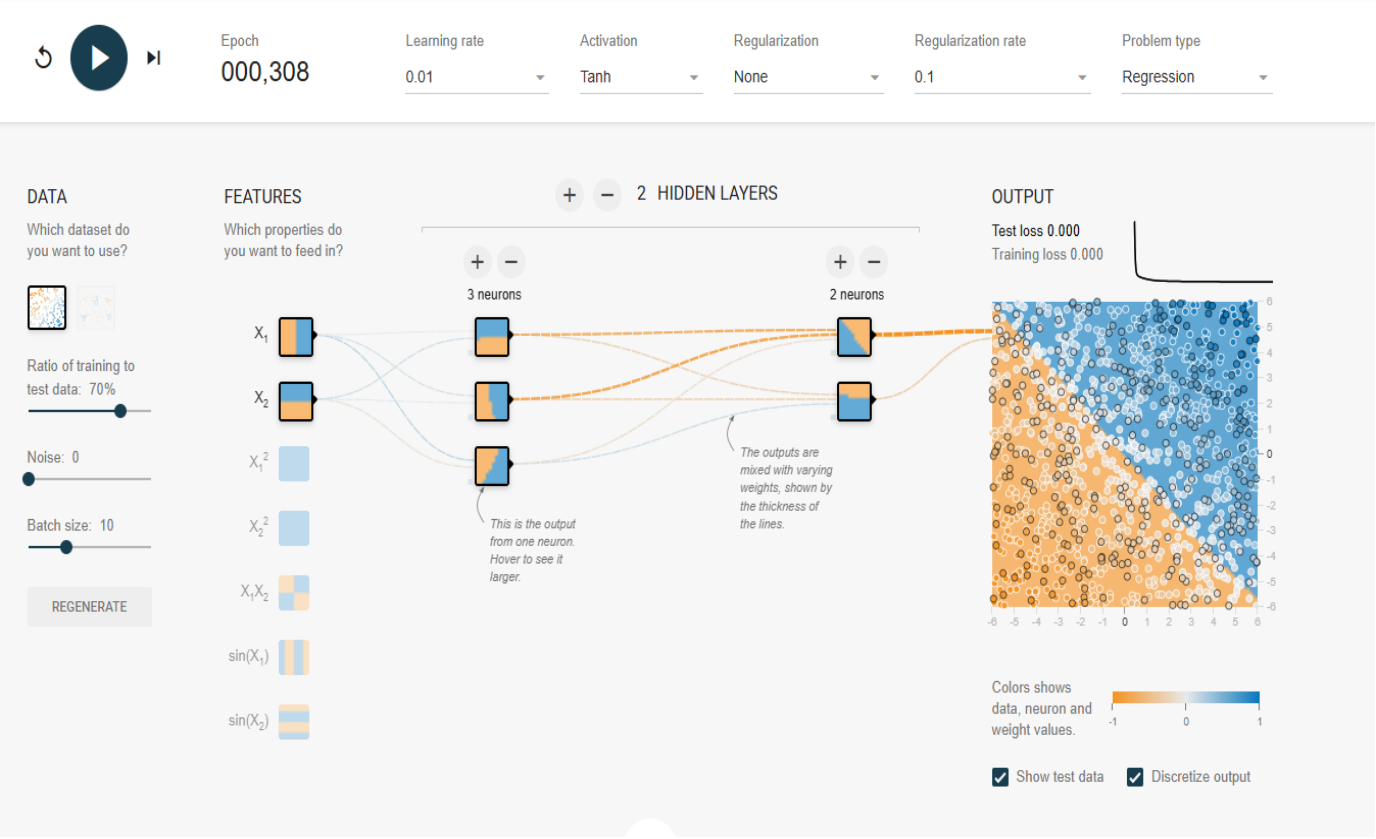
Setting a fixed random state ensures reproducibility.

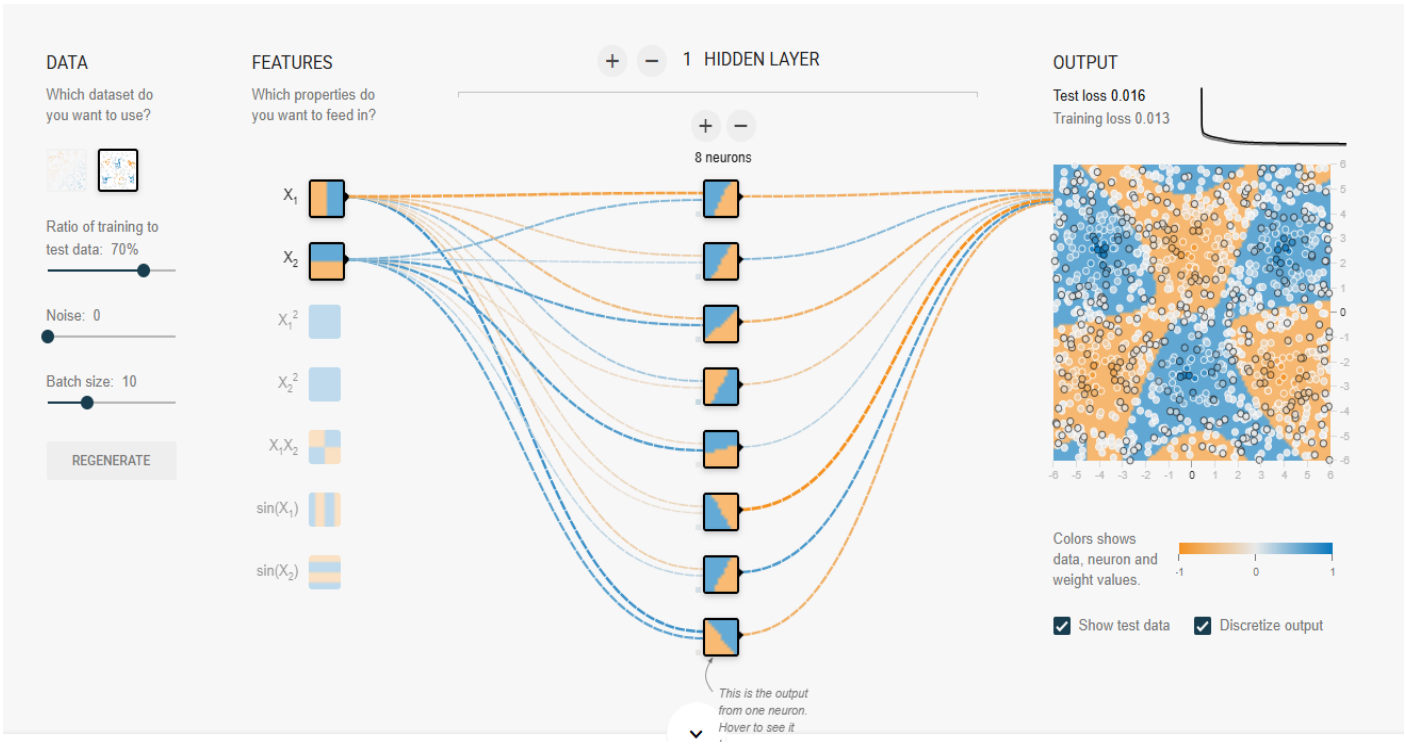
- Each time the code is run, the initialization of weights, data splits, and any other stochastic processes will behave identically.
- This is useful for debugging, consistency in results, and fair comparisons between experiments.

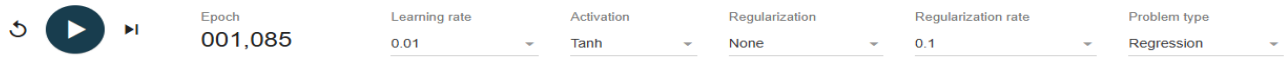
4. Additional program

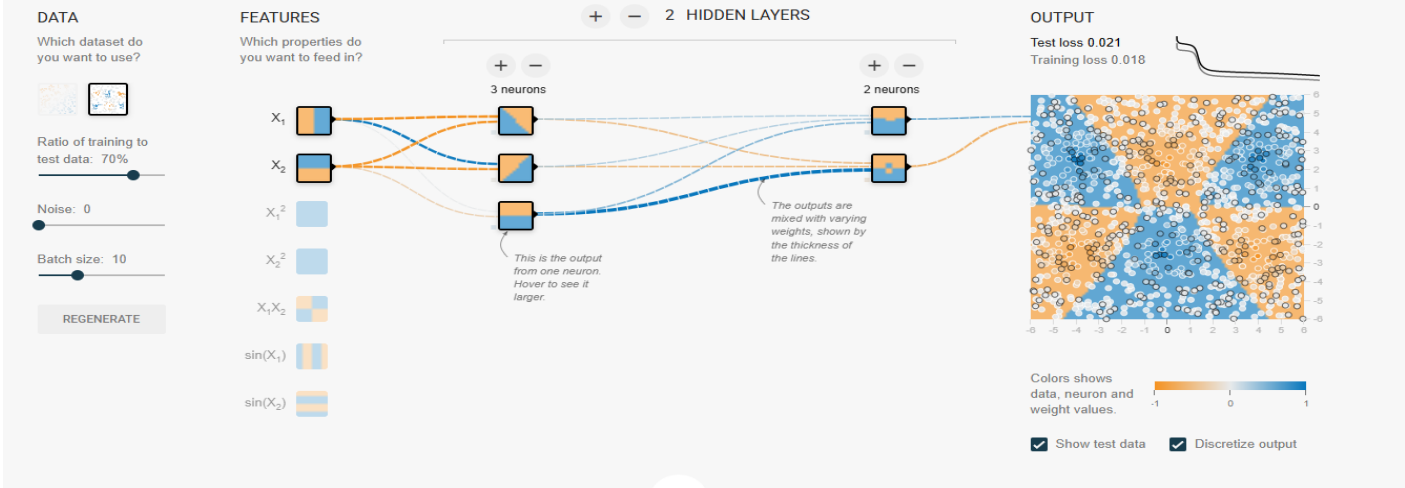
https://playground.tensorflow.org/#activation=tanh&batchSize=10&dataset=circle&regDataset=reg-plane&learningRate=0.03&regularizationRate=0&noise=0&networkShape=4,2&seed=0.05123&showTestData=false&discretize=false&percTrainData=50&x=true&y=true&xTimesY=false&xSquared=false&ySquared=false&cosX=false&sinX=false&cosY=false&sinY=false&collectStats=false&problem=classification&initZero=false&hideText=false

I found this site which simulates a MLP with regression(also with classification) and for the features x1 and x2 I added first 2 hidden layers with 3 and 2 neurons and after that with just 2 and 1 in order to simulate something close visually to what I tried to demonstrate in this project. I set the learning rate at 0.01 as it is set in the code, regularization rate at 0.1 with an activation with tanh because I saw that this is the standard for this type of simulation and a ratio of training and test data of 70%.




DATA
Which dataset do you want to use?
Ratio of training to test data: 70%
Noise: 0
Batch size: 10
REGENERATE
FEATURES
Which properties do you want to feed in?
1 HIDDEN LAYER
8 neurons
X1
X2
X1²
X2²
X1X2
sin(X1)
sin(X2)
This is the output from one neuron. Hover to see it
OUTPUT
Test loss 0.016
Training loss 0.013
Colors shows data, neuron and weight values.
Show test data
Discretize output


Epoch
001,085
Learning rate
0.01
Activation
Tanh
Regularization
None
Regularization rate
0.1
Problem type
Regression


DATA
Which dataset do you want to use?
Ratio of training to test data: 70%
Noise: 0
Batch size: 10
REGENERATE
FEATURES
Which properties do you want to feed in?
2 HIDDEN LAYERS
3 neurons
2 neurons
X1
X2
X1²
X2²
X1X2
sin(X1)
sin(X2)
This is the output from one neuron. Hover to see it larger.
The outputs are mixed with varying weights, shown by the thickness of the lines.
OUTPUT
Test loss 0.021
Training loss 0.018
Colors shows data, neuron and weight values.
Show test data
Discretize output