Student Performance Prediction

Abstract:

For this study, for student performance prediction which is a regression task, after preprocessing the data, we started by constructing a simple mlp model with adam optimizer and l1loss which led into overfitting and from here it determined the flow of our investigation fo performance enhancement by implementing regularization techniques, testing different activation function, and different optimizers to reach the best optimal model.

Introduction:

Overcoming overfitting is an open challenge these days, but certain methods can be practiced to prevent it such as regularization techniques which in this study we tried l2 regularization, dropout and batch normalization, also the choice of activation function or optimizer can influence model performance significantly so we also tried with activation functions: relu, sigmoid, tanh, leaky relu, gelu, and scaled tanh. And for optimizers we tried with adam and SGD.

Training:

After loading the data into a dataframe it was observed that the shape is:
(395, 33)
Meaning we have 395 data samples and 33 features.
Now let's take a look at the features:

```
Index(['age', 'Medu', 'Fedu', 'traveltime', 'studytime', 'failures',
       'schoolsup', 'famsup', 'paid', 'activities', 'nursery', 'higher',
       'internet', 'romantic', 'famrel', 'freetime', 'goout', 'Dalc', 'Walc',
       'health', 'absences', 'G1', 'G2', 'G3', 'sex_M', 'guardian_mother',
       'guardian_other', 'reason_home', 'reason_other', 'reason_reputation',
       'Fjob_health', 'Fjob_other', 'Fjob_services', 'Fjob_teacher',
       'Mjob_health', 'Mjob_other', 'Mjob_services', 'Mjob_teacher',
       'famsize_LE3', 'school_MS', 'address_U', 'Pstatus_T'],
      dtype='object')
```

Target value is the 'G3' column.

Description of the data:

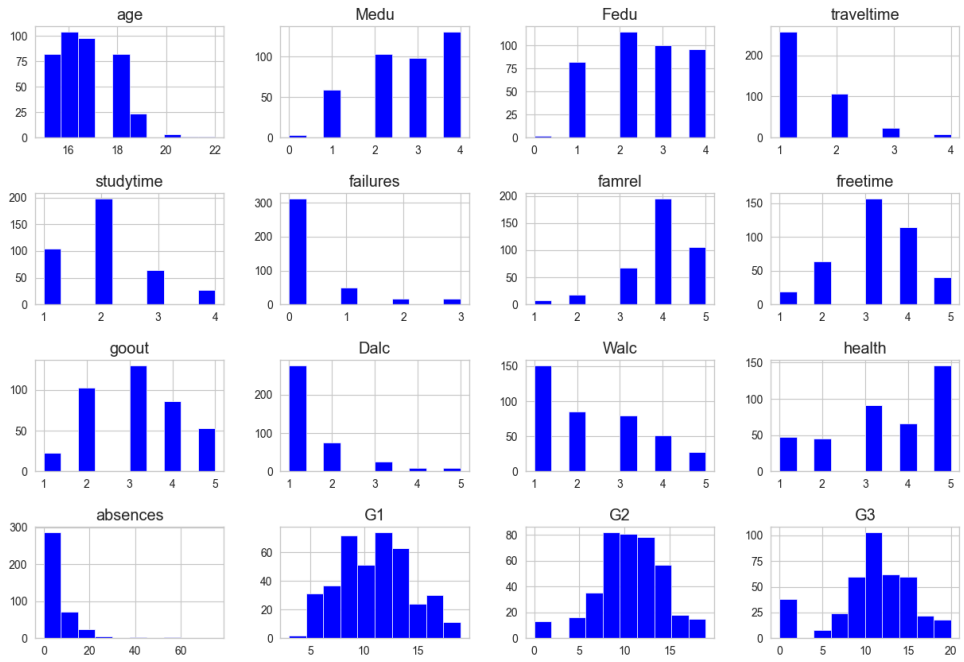
The range of given values of columns suggests a
Need for normalization.

	count	mean	std	min	25%	50%	75%	max
age	395.0	16.696203	1.276043	15.0	16.0	17.0	18.0	22.0
Medu	395.0	2.749367	1.094735	0.0	2.0	3.0	4.0	4.0
Fedu	395.0	2.521519	1.088201	0.0	2.0	2.0	3.0	4.0
traveltime	395.0	1.448101	0.697505	1.0	1.0	1.0	2.0	4.0
studytime	395.0	2.035443	0.839240	1.0	1.0	2.0	2.0	4.0
failures	395.0	0.334177	0.743651	0.0	0.0	0.0	0.0	3.0
famrel	395.0	3.944304	0.896659	1.0	4.0	4.0	5.0	5.0
freetime	395.0	3.235443	0.998862	1.0	3.0	3.0	4.0	5.0
goout	395.0	3.108861	1.113278	1.0	2.0	3.0	4.0	5.0
Dalc	395.0	1.481013	0.890741	1.0	1.0	1.0	2.0	5.0
Walc	395.0	2.291139	1.287897	1.0	1.0	2.0	3.0	5.0
health	395.0	3.554430	1.390303	1.0	3.0	4.0	5.0	5.0
absences	395.0	5.708861	8.003096	0.0	0.0	4.0	8.0	75.0
G1	395.0	10.908861	3.319195	3.0	8.0	11.0	13.0	19.0
G2	395.0	10.713924	3.761505	0.0	9.0	11.0	13.0	19.0
G3	395.0	10.415190	4.581443	0.0	8.0	11.0	14.0	20.0

For missing value removal step, we don't have to do anything since the data is already not missing any value.
We also got unique values for each column to see any error was made:
school ['GP' 'MS']
sex ['F' 'M']
age [18 17 15 16 19 22 20 21]
address ['U' 'R']
famsize ['GT3' 'LE3']
Pstatus ['A' 'T']
Medu [4 1 3 2 0]
Fedu [4 1 2 3 0]
Mjob ['at_home' 'health' 'other' 'services' 'teacher']
Fjob ['teacher' 'other' 'services' 'health' 'at_home']

reason ['course' 'other' 'home' 'reputation']
guardian ['mother' 'father' 'other']
traveltime [2 1 3 4]
studytime [2 3 1 4]
failures [0 3 2 1]
schoolsup ['yes' 'no']
famsup ['no' 'yes']
paid ['no' 'yes']
activities ['no' 'yes']
nursery ['yes' 'no']
higher ['yes' 'no']
internet ['no' 'yes']
romantic ['no' 'yes']
famrel [4 5 3 1 2]
freetime [3 2 4 1 5]
goout [4 3 2 1 5]
Dalc [1 2 5 3 4]
Walc [1 3 2 4 5]
health [3 5 1 2 4]
absences [6 4 10 2 0 16 14 7 8 25 12 54 18 26 20 56 24 28 5 13 15 22 3 21
  1 75 30 19 9 11 38 40 23 17]
G1 [5 7 15 6 12 16 14 10 13 8 11 9 17 19 18 4 3]
G2 [6 5 8 14 10 15 12 18 16 13 9 11 7 19 17 4 0]
G3 [6 10 15 11 19 9 12 14 16 5 8 17 18 13 20 7 0 4]

Which doesn’t seems to be any.
Now let’s take a look at the distribution of numberical columns:



From the given plot it seemed to me that g3 and g2 have noticable outlier. However getting a scire under 5 is not exactly considered outlier but i decided by removing them my model can focus more on predicting the more populated ,outlier free area so after splitting the data into train and test i removed outliers of these two functions from the train data but since our dataset is very small (375 samples) I didn't removed other feature's outliers.

Fo the encoding process all the yes and no values were converted to 0 and 1 and all the other categorical columns were one hot encoded. At the end we ended up with 42 features.

For the next step the datafame was divinded into train and test (80% and 20%):
Train set shape: (316, 42)
Test set shape: (79, 42)

And then the outliers i mentioned were deleted from the dataset.

Then MinMaxScaler was implemented on both train and test separately. And g3 was separated as target.

Now for the futures model we are going to train and evaluated i created three functions for train, test and plotting. To preserve and plot mse, mae and r2-score.

Primary model architecture:

```
MLP(
  (layers): Sequential(
    (0): Linear(in_features=41, out_features=64, bias=True)
    (1): ReLU()
    (2): Linear(in_features=64, out_features=128, bias=True)
    (3): ReLU()
    (4): Linear(in_features=128, out_features=256, bias=True)
    (5): ReLU()
    (6): Linear(in_features=256, out_features=1, bias=True)
  )
)
```

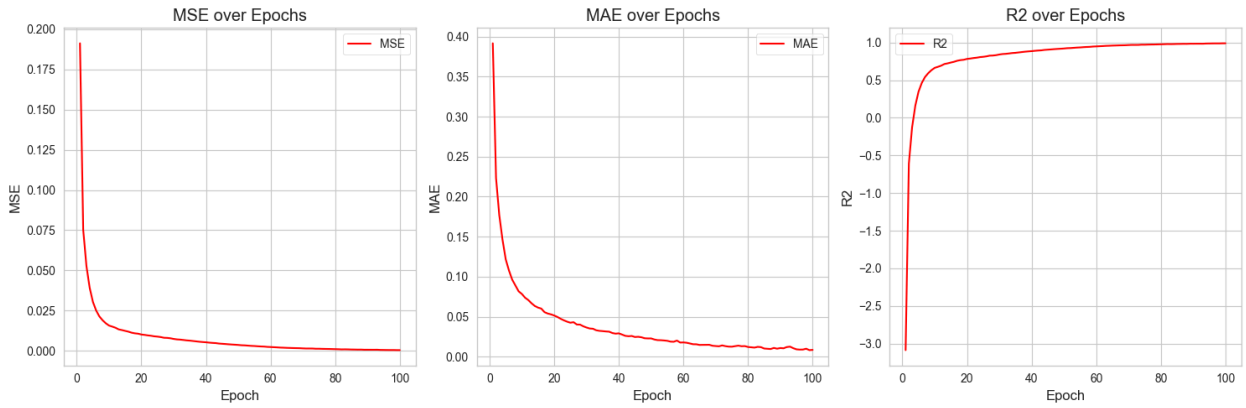
This is the primary model we are going to use and enhance further to reach the optimal evaluation.

Model 1:

Let's train this model with adam optimizer and l1loss.
Train results:

```
Epoch 1 finished
Mean Squared Error: 0.1911933571100235, Mean Absolute Error: 0.3915400505065918, R2 Score: -3.0876555757342805
Starting epoch 2
Loss after mini-batch     1: 0.001
Loss after mini-batch    11: 0.005
Loss after mini-batch    21: 0.004
Loss after mini-batch    31: 0.004
Epoch 2 finished
Mean Squared Error: 0.07529658079147339, Mean Absolute Error: 0.22335456311702728, R2 Score: -0.609817881652648
Starting epoch 3
Loss after mini-batch     1: 0.000
Loss after mini-batch    11: 0.004
Loss after mini-batch    21: 0.004
Loss after mini-batch    31: 0.003
Epoch 3 finished
Mean Squared Error: 0.05273020640015602, Mean Absolute Error: 0.17703169584274292, R2 Score: -0.12735555311797775
Starting epoch 4
Loss after mini-batch     1: 0.000
Loss after mini-batch    11: 0.003
Loss after mini-batch    21: 0.003
...
Loss after mini-batch    31: 0.000
Epoch 100 finished
Mean Squared Error: 0.0004598449741024524, Mean Absolute Error: 0.008641615509986877, R2 Score: 0.9901686556525102
Training process has finished.
```

Final r2 score suggests the case of overfitting. Let's see the test results to be sure.
Test score:
Mean Squared Error: 0.022596426
Mean Absolute Error: 0.1142134
R2 Score: 0.5934380116665545



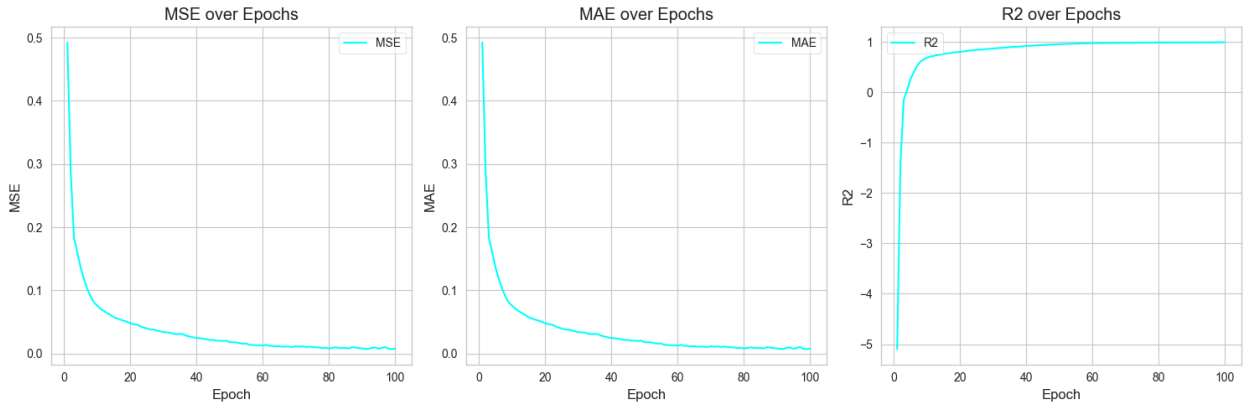
Now that we know we came across the case of overfitting we can direct the flow of our procces into resolving this issue.therefor the next model we can try is:

The same model but with l2 regularization:
Train results:

```
Epoch 1 finished
Mean Squared Error: 0.28521907329559326, Mean Absolute Error: 0.4927802085876465, R2 Score: -5.09789636290787
Starting epoch 2
Loss after mini-batch     1: 0.001
Loss after mini-batch    11: 0.007
Loss after mini-batch    21: 0.006
Loss after mini-batch    31: 0.004
Epoch 2 finished
Mean Squared Error: 0.11186031252145767, Mean Absolute Error: 0.288011759519577, R2 Score: -1.3915390108692036
Starting epoch 3
Loss after mini-batch     1: 0.000
Loss after mini-batch    11: 0.004
Loss after mini-batch    21: 0.004
Loss after mini-batch    31: 0.004
Epoch 3 finished
Mean Squared Error: 0.053526099771261215, Mean Absolute Error: 0.18325276672840118, R2 Score: -0.1443714883562779
Starting epoch 4
Loss after mini-batch     1: 0.000
Loss after mini-batch    11: 0.004
Loss after mini-batch    21: 0.003
...
Loss after mini-batch    31: 0.000
Epoch 100 finished
Mean Squared Error: 0.0004300615983083844, Mean Absolute Error: 0.0077139707282185555, R2 Score: 0.9908054147149297
```

Test results:
Mean Squared Error: 0.022290315
Mean Absolute Error: 0.11380757
R2 Score: 0.5989456487078781

The test results has improved a little so from now on, we experiment other models with addition of l2 regularization.



Mlp with dropout:
One efficient way to overcome overfitting is dropout layers.
First we started with dropout = 0.5 and by decreasing it to 0.3, 0.2 and finally 0.1, we observed that 0.1 gave the best result.

```
MLP_dropout(
  (layers): Sequential(
    (0): Linear(in_features=41, out_features=64, bias=True)
```

```
    (1): ReLU()
    (2): Dropout(p=0.1, inplace=False)
    (3): Linear(in_features=64, out_features=128, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.1, inplace=False)
    (6): Linear(in_features=128, out_features=256, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.1, inplace=False)
    (9): Linear(in_features=256, out_features=1, bias=True)
  )
)
```

Train result:

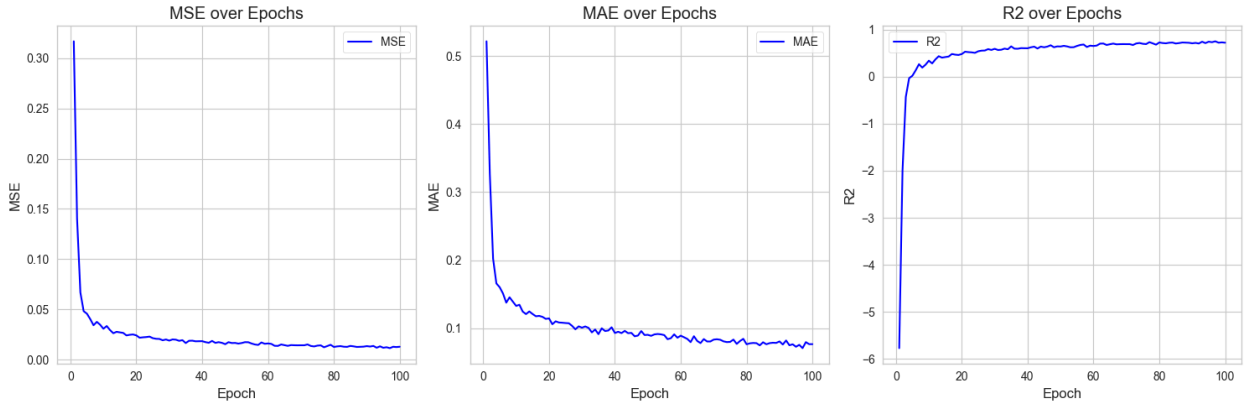
```
Epoch 1 finished
Mean Squared Error: 0.18606562912464142, Mean Absolute Error: 0.39028096199035645, R2 Score: -2.97802595821118
Starting epoch 2
Loss after mini-batch     1: 0.001
Loss after mini-batch    11: 0.005
Loss after mini-batch    21: 0.004
Loss after mini-batch    31: 0.004
Epoch 2 finished
Mean Squared Error: 0.06834963709115982, Mean Absolute Error: 0.21775643527507782, R2 Score: -0.4612942023004001
Starting epoch 3
Loss after mini-batch     1: 0.000
Loss after mini-batch    11: 0.004
Loss after mini-batch    21: 0.003
Loss after mini-batch    31: 0.003
Epoch 3 finished
Mean Squared Error: 0.04816880077123642, Mean Absolute Error: 0.1689808964729309, R2 Score: -0.02983418550000705
Starting epoch 4
Loss after mini-batch     1: 0.000
Loss after mini-batch    11: 0.003
Loss after mini-batch    21: 0.003
...
Loss after mini-batch    31: 0.001
Epoch 100 finished
Mean Squared Error: 0.005419175606220961, Mean Absolute Error: 0.05020134150981903, R2 Score: 0.8841396958295357
Training process has finished.
```

Test result:

Mean Squared Error: 0.020590972

Mean Absolute Error: 0.112254605

R2 Score: 0.6295208140245825



Test score has improved significantly so in addition to l2, we are also going to keep dropout layers as well.

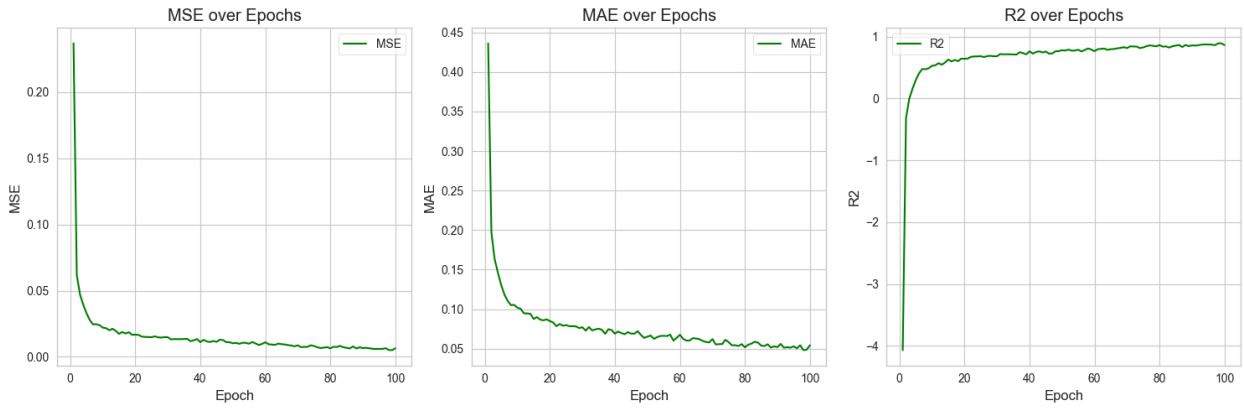
Dropout with additional layers:

I also tested adding a new layer of 512 neurons with dropout to see the effect. I also increased dropout prob to 0.2 to regulate it.

Train result:

```
Epoch 1 finished
Mean Squared Error: 0.21227198839187622, Mean Absolute Error: 0.41383740305900574, R2 Score: -3.5383099477545503
Starting epoch 2
Loss after mini-batch     1: 0.000
Loss after mini-batch    11: 0.004
Loss after mini-batch    21: 0.004
Loss after mini-batch    31: 0.004
Epoch 2 finished
Mean Squared Error: 0.06790407747030258, Mean Absolute Error: 0.2061394900083542, R2 Score: -0.4517682879825824
Starting epoch 3
Loss after mini-batch     1: 0.001
Loss after mini-batch    11: 0.004
Loss after mini-batch    21: 0.003
Loss after mini-batch    31: 0.004
Epoch 3 finished
Mean Squared Error: 0.052945662289857864, Mean Absolute Error: 0.1770511418581009, R2 Score: -0.13196197106188778
Starting epoch 4
Loss after mini-batch     1: 0.000
Loss after mini-batch    11: 0.003
Loss after mini-batch    21: 0.003
...
Loss after mini-batch    31: 0.001
Epoch 100 finished
Mean Squared Error: 0.00845759641379118, Mean Absolute Error: 0.06357593089342117, R2 Score: 0.8191791909843088
Training process has finished.
```

Test result:
Mean Squared Error: 0.023744931
Mean Absolute Error: 0.12071036
R2 Score: 0.5727737798917552



It seems to be not working better than our previous structure.

Mlp with tanh:

Here we dicided to experiment with different activation functions to see if they enhance out performance or not.

```
MLP_tanh(
  (layers): Sequential(
    (0): Linear(in_features=41, out_features=64, bias=True)
    (1): Tanh()
    (2): Dropout(p=0.1, inplace=False)
    (3): Linear(in_features=64, out_features=128, bias=True)
    (4): Tanh()
    (5): Dropout(p=0.1, inplace=False)
    (6): Linear(in_features=128, out_features=256, bias=True)
    (7): Tanh()
    (8): Dropout(p=0.1, inplace=False)
    (9): Linear(in_features=256, out_features=1, bias=True)
  )
)
```

Train result:

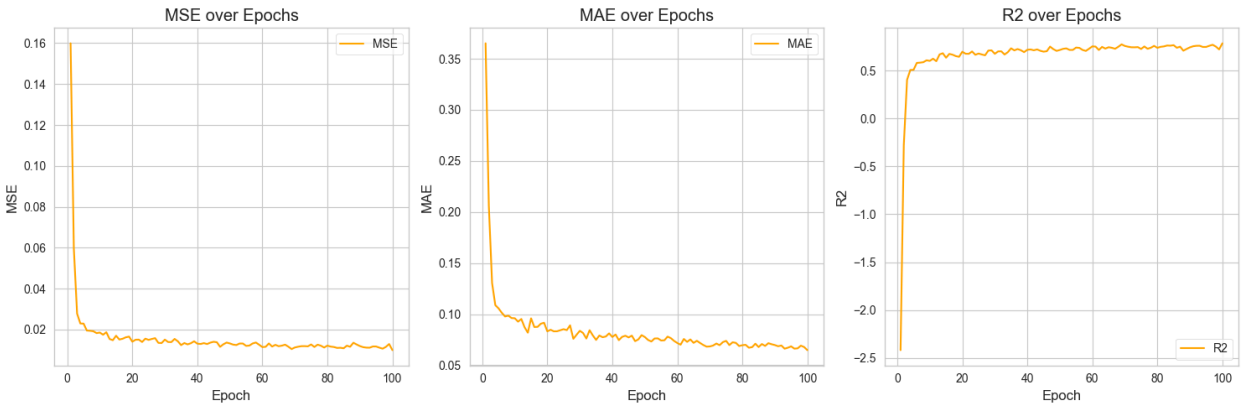
```
Mean Squared Error: 0.15994536876678467, Mean Absolute Error: 0.36511170864105225, R2 Score: -2.419582630689947
Starting epoch 2
Loss after mini-batch     1: 0.000
Loss after mini-batch    11: 0.005
Loss after mini-batch    21: 0.005
Loss after mini-batch    31: 0.003
Epoch 2 finished
Mean Squared Error: 0.0596437007188797, Mean Absolute Error: 0.20631259679794312, R2 Score: -0.27516408619851185
Starting epoch 3
Loss after mini-batch     1: 0.000
Loss after mini-batch    11: 0.003
Loss after mini-batch    21: 0.002
Loss after mini-batch    31: 0.002
Epoch 3 finished
Mean Squared Error: 0.027835508808493614, Mean Absolute Error: 0.13052751123905182, R2 Score: 0.4048853431115394
Starting epoch 4
Loss after mini-batch     1: 0.000
Loss after mini-batch    11: 0.002
Loss after mini-batch    21: 0.002
...
Loss after mini-batch    31: 0.001
Epoch 100 finished
Mean Squared Error: 0.00997978262603283, Mean Absolute Error: 0.06481368839740753, R2 Score: 0.7866353377260336
```

Test result:

Mean Squared Error: 0.015534445

Mean Absolute Error: 0.08382388

R2 Score: 0.7204994253143818



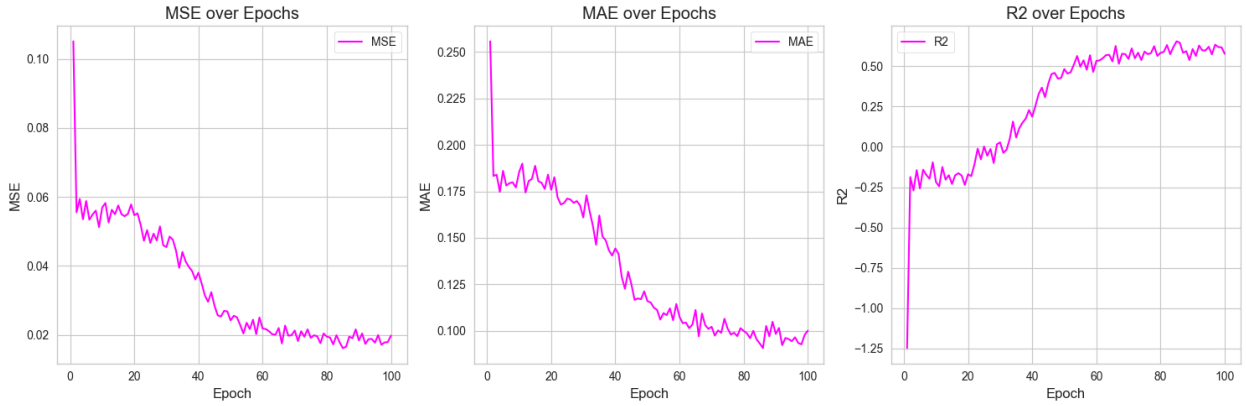
However the model seems to be underfitting on train, but it resulted in way better test score than relu, the train set to be underfitting gives us the hope that if we train this model for further epochs we can reach even better total results.

Mlp with sigmoid:

Train result:

```
Mean Squared Error: 0.10513739287853241, Mean Absolute Error: 0.2557757496833801, R2 Score: -1.2478053841862962
Starting epoch 2
Loss after mini-batch     1: 0.000
Loss after mini-batch    11: 0.004
Loss after mini-batch    21: 0.004
Loss after mini-batch    31: 0.004
Epoch 2 finished
Mean Squared Error: 0.055532798171043396, Mean Absolute Error: 0.18329142034053802, R2 Score: -0.18727422800952742
Starting epoch 3
Loss after mini-batch     1: 0.000
Loss after mini-batch    11: 0.004
Loss after mini-batch    21: 0.003
Loss after mini-batch    31: 0.004
Epoch 3 finished
Mean Squared Error: 0.059400882571935654, Mean Absolute Error: 0.18393965065479279, R2 Score: -0.2699726401957825
Starting epoch 4
Loss after mini-batch     1: 0.000
Loss after mini-batch    11: 0.004
Loss after mini-batch    21: 0.004
...
Loss after mini-batch    31: 0.002
Epoch 100 finished
Mean Squared Error: 0.019803117960691452, Mean Absolute Error: 0.10003434121608734, R2 Score: 0.5766154257175391
```

Test result:
Mean Squared Error: 0.020751942
Mean Absolute Error: 0.08680644
R2 Score: 0.6266245536436741



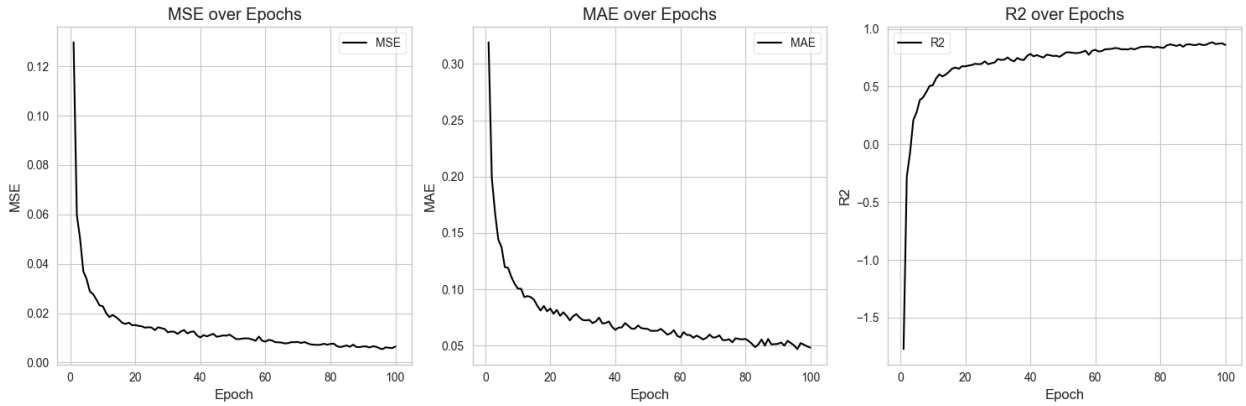
This model is highly underfitted but still seems to be working better than relu.
The r2 plot suggests after 80 epochs improvent is very little.

Mlp with leaky relu:

Train result:

```
Mean Squared Error: 0.12968137860298157, Mean Absolute Error: 0.3190944790840149, R2 Score: -1.7725479259220376
Starting epoch 2
Loss after mini-batch     1: 0.000
Loss after mini-batch    11: 0.004
Loss after mini-batch    21: 0.004
Loss after mini-batch    31: 0.003
Epoch 2 finished
Mean Squared Error: 0.05976494774222374, Mean Absolute Error: 0.1987968236207962, R2 Score: -0.27775626676060905
Starting epoch 3
Loss after mini-batch     1: 0.000
Loss after mini-batch    11: 0.004
Loss after mini-batch    21: 0.003
Loss after mini-batch    31: 0.003
Epoch 3 finished
Mean Squared Error: 0.05016280338168144, Mean Absolute Error: 0.16764405369758606, R2 Score: -0.07246527169275097
Starting epoch 4
Loss after mini-batch     1: 0.000
Loss after mini-batch    11: 0.003
Loss after mini-batch    21: 0.002
...
Loss after mini-batch    31: 0.001
Epoch 100 finished
Mean Squared Error: 0.006522571202367544, Mean Absolute Error: 0.04832863807678223, R2 Score: 0.8605494159482268
Training process has finished.
```

Test result:
Mean Squared Error: 0.02172223
Mean Absolute Error: 0.11421287
R2 Score: 0.60916681271292



It looks like relu and its extention leaky relu both tend to overfit.

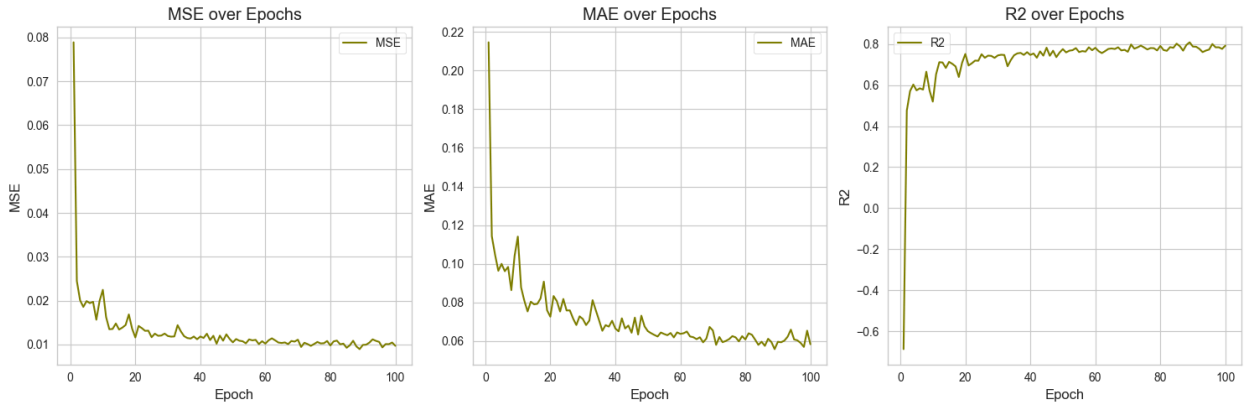
Mlp with sgd:

Now that we have experimented different activation functions, it is also a good idea to experiment a different optimizer, here we chose SGD.

Train result:

```
Loss after mini-batch    11: 0.005
Loss after mini-batch    21: 0.004
Loss after mini-batch    31: 0.003
Epoch 1 finished
Mean Squared Error: 0.07890824973583221, Mean Absolute Error: 0.21461783349514008, R2 Score: -0.6870339153406644
Starting epoch 2
Loss after mini-batch     1: 0.000
Loss after mini-batch    11: 0.002
Loss after mini-batch    21: 0.002
Loss after mini-batch    31: 0.002
Epoch 2 finished
Mean Squared Error: 0.024510735645890236, Mean Absolute Error: 0.11427118629217148, R2 Score: 0.4759680181555933
Starting epoch 3
Loss after mini-batch     1: 0.000
Loss after mini-batch    11: 0.002
Loss after mini-batch    21: 0.002
Loss after mini-batch    31: 0.002
Epoch 3 finished
Mean Squared Error: 0.020086748525500298, Mean Absolute Error: 0.10467381030321121, R2 Score: 0.5705514977958852
Starting epoch 4
Loss after mini-batch     1: 0.000
Loss after mini-batch    11: 0.002
Loss after mini-batch    21: 0.002
...
Loss after mini-batch    31: 0.001
Epoch 100 finished
Mean Squared Error: 0.00970558449625969, Mean Absolute Error: 0.05833876505494118, R2 Score: 0.792497576603692
Training process has finished.
```

Test result:
Mean Squared Error: 0.014364576
Mean Absolute Error: 0.07520748
R2 Score: 0.7415480627547395



Suprisingly sgd worked better than adam for our task.
Until now we have reached the optimal solution of combination of sgd, tanh, and dropout layers.

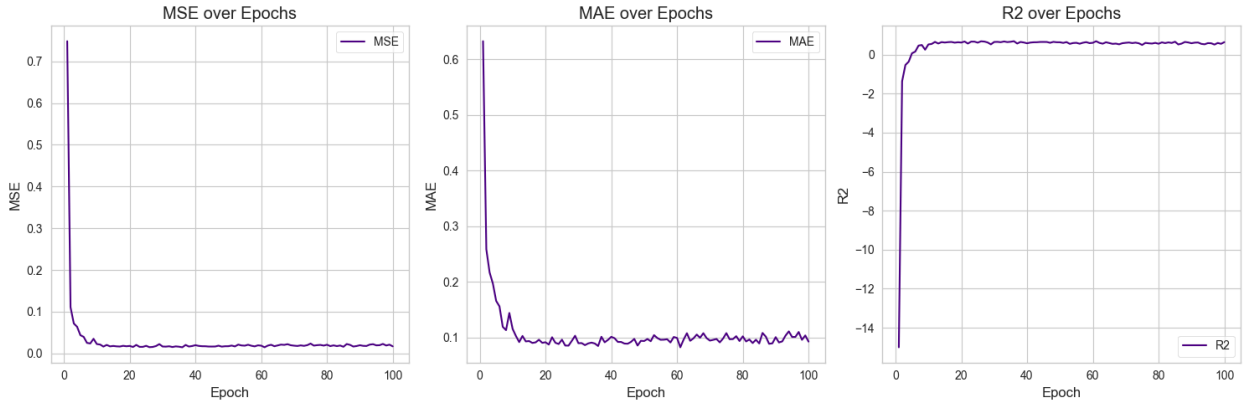
MIp with batch normalization:
Batch normalization as a regularization technique can be helpful with the problem of overfitting.

```
MLP_batch_normal(
  (layers): Sequential(
    (0): Linear(in_features=41, out_features=64, bias=True)
    (1): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): Tanh()
    (3): Dropout(p=0.1, inplace=False)
    (4): Linear(in_features=64, out_features=128, bias=True)
    (5): Tanh()
    (6): Dropout(p=0.1, inplace=False)
    (7): Linear(in_features=128, out_features=256, bias=True)
    (8): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (9): Tanh()
    (10): Dropout(p=0.1, inplace=False)
    (11): Linear(in_features=256, out_features=1, bias=True)
  )
)
```

Train result:

```
Epoch 1 finished
Mean Squared Error: 0.7482773661613464, Mean Absolute Error: 0.6325470209121704, R2 Score: -14.997939465843473
Starting epoch 2
Loss after mini-batch     1: 0.001
Loss after mini-batch    11: 0.007
Loss after mini-batch    21: 0.004
Loss after mini-batch    31: 0.004
Epoch 2 finished
Mean Squared Error: 0.11095727980136871, Mean Absolute Error: 0.25833752751350403, R2 Score: -1.3722324708998541
Starting epoch 3
Loss after mini-batch     1: 0.000
Loss after mini-batch    11: 0.005
Loss after mini-batch    21: 0.004
Loss after mini-batch    31: 0.004
Epoch 3 finished
Mean Squared Error: 0.07185216248035431, Mean Absolute Error: 0.2167060226202011, R2 Score: -0.5361770472551648
Starting epoch 4
Loss after mini-batch     1: 0.000
Loss after mini-batch    11: 0.004
Loss after mini-batch    21: 0.003
...
Loss after mini-batch    31: 0.002
Epoch 100 finished
Mean Squared Error: 0.01703282818198204, Mean Absolute Error: 0.09227093309164047, R2 Score: 0.6358433463347954
Training process has finished.
```

Test result:
Mean Squared Error: 0.023956357
Mean Absolute Error: 0.114968024
R2 Score: 0.5689697641346463



The model was underfitted and even by decreasing batch normalization layers and increasing neurons, it remained. And the plot of r2 doesn't gives us much hope of improving by more epochs.

I also added a new layer to the new architect with tanh as activation function and dropout 0.15. It resulted in

```
Mean Squared Error: 0.014603795
Mean Absolute Error: 0.07748335
R2 Score: 0.7372439528892245
```

Which is less than before even with dropout prob hyperparameter, tuned.

Mlp with gelu:

Train result:

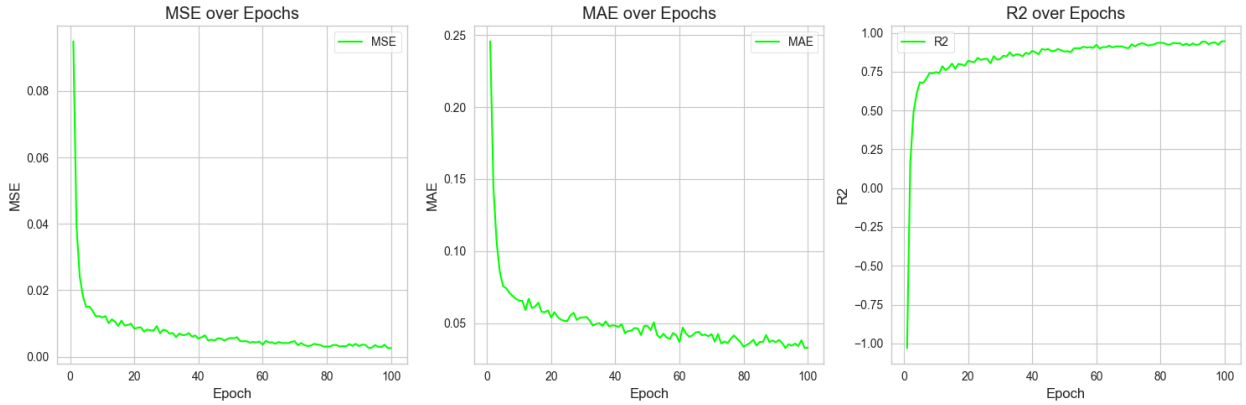
```
Epoch 1 finished
Mean Squared Error: 0.09496429562568665, Mean Absolute Error: 0.245620995759964, R2 Score: -1.0303075074797041
Starting epoch 2
Loss after mini-batch     1: 0.000
Loss after mini-batch    11: 0.003
Loss after mini-batch    21: 0.003
Loss after mini-batch    31: 0.002
Epoch 2 finished
Mean Squared Error: 0.03902152180671692, Mean Absolute Error: 0.1443554013967514, R2 Score: 0.16573181372778334
Starting epoch 3
Loss after mini-batch     1: 0.000
Loss after mini-batch    11: 0.002
Loss after mini-batch    21: 0.002
Loss after mini-batch    31: 0.002
Epoch 3 finished
Mean Squared Error: 0.024019572883844376, Mean Absolute Error: 0.1062866672873497, R2 Score: 0.48646889588223563
Starting epoch 4
Loss after mini-batch     1: 0.000
Loss after mini-batch    11: 0.002
Loss after mini-batch    21: 0.002
...
Loss after mini-batch    31: 0.001
Epoch 100 finished
Mean Squared Error: 0.0025455437134951353, Mean Absolute Error: 0.03311586007475853, R2 Score: 0.9455770587232679
```

Test result:

Mean Squared Error: 0.017056882

Mean Absolute Error: 0.08528805

R2 Score: 0.6931072421214521



It reemphasized the fact that relu and its variations are prone to overfitting.

Scaled tanh:

Scaled tanh is defined as scale_factor * torch.tanh(x)
Train result:

```
Mean Squared Error: 0.12140633165836334, Mean Absolute Error: 0.2743074893951416, R2 Score: -1.595629827638843
Starting epoch 2
Loss after mini-batch     1: 0.000
Loss after mini-batch    11: 0.003
Loss after mini-batch    21: 0.003
Loss after mini-batch    31: 0.003
Epoch 2 finished
Mean Squared Error: 0.04075092822313309, Mean Absolute Error: 0.1480179876089096, R2 Score: 0.12875766971477554
Starting epoch 3
Loss after mini-batch     1: 0.000
Loss after mini-batch    11: 0.003
Loss after mini-batch    21: 0.002
Loss after mini-batch    31: 0.002
Epoch 3 finished
Mean Squared Error: 0.030902737751603127, Mean Absolute Error: 0.12191211432218552, R2 Score: 0.33930895384144466
Starting epoch 4
Loss after mini-batch     1: 0.000
Loss after mini-batch    11: 0.002
Loss after mini-batch    21: 0.002
...
Loss after mini-batch    31: 0.001
Epoch 100 finished
Mean Squared Error: 0.010108260437846184, Mean Absolute Error: 0.05782197043299675, R2 Score: 0.7838885054178785
Training process has finished.
```

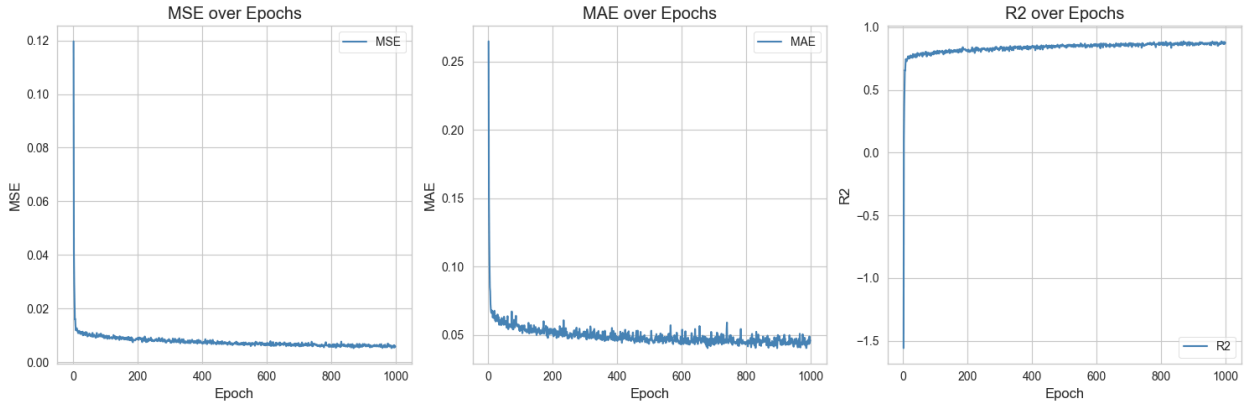
Test result:
Mean Squared Error: 0.014931433
Mean Absolute Error: 0.08223872
R2 Score: 0.7313489800295014

Although its not as good as the result we had with tanh but when i trained tanh for 1000 epochs it didn’t resulted in any improvement but with scaled tanh with 1000 epochs:

Train result:

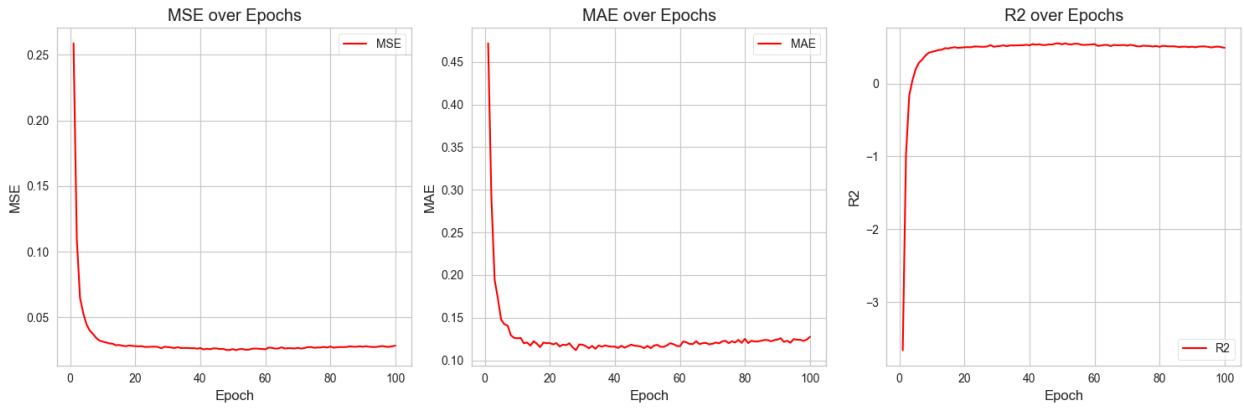
```
Mean Squared Error: 0.11968071758747101, Mean Absolute Error: 0.26465946435928345, R2 Score: -1.5587371609276528
Starting epoch 2
Loss after mini-batch     1: 0.000
Loss after mini-batch    11: 0.003
Loss after mini-batch    21: 0.003
Loss after mini-batch    31: 0.003
Epoch 2 finished
Mean Squared Error: 0.04130703955888748, Mean Absolute Error: 0.15247303247451782, R2 Score: 0.11686820164062683
Starting epoch 3
Loss after mini-batch     1: 0.000
Loss after mini-batch    11: 0.003
Loss after mini-batch    21: 0.002
Loss after mini-batch    31: 0.002
Epoch 3 finished
Mean Squared Error: 0.029368340969085693, Mean Absolute Error: 0.12110570818185806, R2 Score: 0.3721138443413653
Starting epoch 4
Loss after mini-batch     1: 0.000
Loss after mini-batch    11: 0.002
Loss after mini-batch    21: 0.002
...
Loss after mini-batch    31: 0.001
Epoch 1000 finished
Mean Squared Error: 0.005866364575922489, Mean Absolute Error: 0.04596465826034546, R2 Score: 0.8745789255372639
Training process has finished.
```

Test result:
Mean Squared Error: 0.013751178
Mean Absolute Error: 0.07862977
R2 Score: 0.7525845078135946

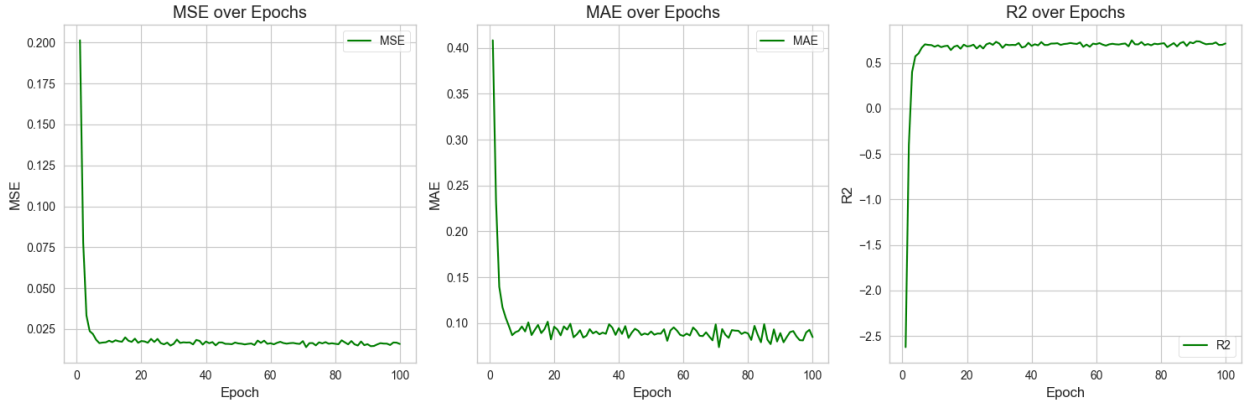


We reached the most optimal value of test r2 score.

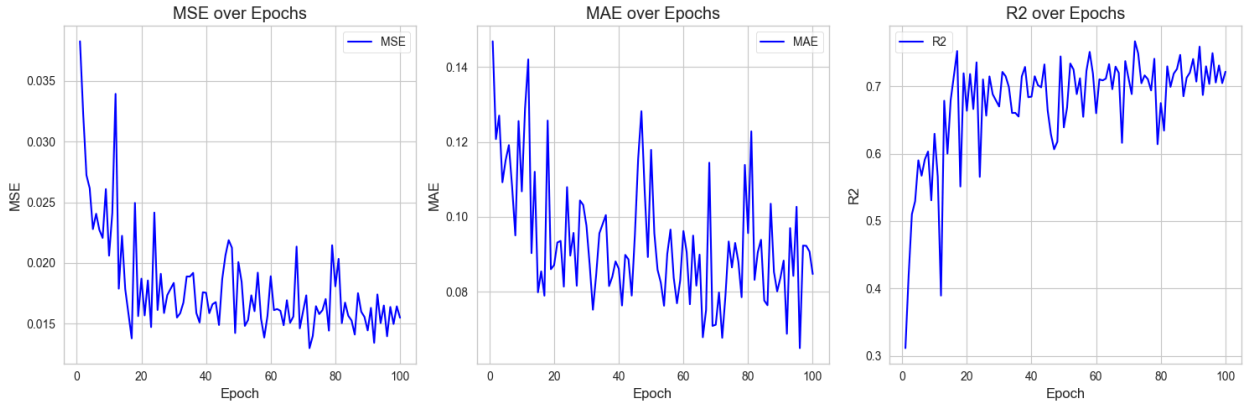
Afterward i added test plots of some models, below are the results:
Primary mlp test model:



Tanh with dropout test plot:



With SGD test plot:



Conclusion

From the various model we tested, a combination of scaled tanh as the activation function, dropout layers with dropout probibility 0.1, SGD optimizer for 1000 epochs gave us the best test result:
Mean Squared Error: 0.013751178
Mean Absolute Error: 0.07862977
R2 Score: 0.7525845078135946

Model architecture:

```
MLP_tanh_scaled(
  (layers): Sequential(
    (0): Linear(in_features=41, out_features=64, bias=True)
    (1): ScaledTanh()
    (2): Dropout(p=0.1, inplace=False)
    (3): Linear(in_features=64, out_features=128, bias=True)
    (4): ScaledTanh()
    (5): Dropout(p=0.1, inplace=False)
    (6): Linear(in_features=128, out_features=256, bias=True)
    (7): ScaledTanh()
    (8): Dropout(p=0.1, inplace=False)
    (9): Linear(in_features=256, out_features=1, bias=True)
  )
)
```