Miniproject_II_Guess_the_year_Hussen_Mohamed_Sebastian_Cajas

April 24, 2021

0.1 Miniproject II, Guess the year !

0.1.1 Import the Required Libraries

[ ]:
```
import pandas as pd
import torch as t
import torch.nn as nn

import matplotlib.pyplot as plt

!nvidia-smi
```

```
Fri Apr 23 00:20:26 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 465.19.01    Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   39C    P8     9W /  70W |      0MiB / 15109MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

[ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[ ]:
```python
#Check cuda version
is_cuda_available=t.cuda.is_available()
print(is_cuda_available)
# Generalize cuda for all comands without writing .cuda()
device = t.device('cuda:0')
t.cuda.set_device(device)
```

```
True
```

0.1.2 Using pandas, load the .csv file.

[ ]:
```python
data = pd.read_csv('/content/drive/MyDrive/Datasets_all/YearPredictionMSD.
 ↪csv',sep=',')
```

0.1.3 Verify whether any qualitative data is present in the dataset.

Remarks

- Column 0 corresponds to the years
- Column 1 corresponds to the frequencies
- Columns 2-91 corresponds to deviations of different songs

[ ]:
```python
data.head()
```

[ ]:
```
      0         1         2         3  …         87        88         89
90
0  2001  49.94357  21.47114  73.07750  …   68.40795  -1.82223  -27.46348
2.26327
1  2001  48.73215  18.42930  70.32679  …   70.49388  12.04941   58.43453
26.92061
2  2001  50.95714  31.85602  55.81851  … -115.00698  -0.05859   39.67068
-0.66345
3  2001  48.24750  -1.89837  36.29772  …  -72.08993   9.90558  199.62971
18.85382
4  2001  50.97020  42.20998  67.09964  …   51.76631   7.88713   55.66926
28.74903

[5 rows x 91 columns]
```

[ ]:
```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
```

```
Data columns (total 91 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   0       50000 non-null  int64
 1   1       50000 non-null  float64
 2   2       50000 non-null  float64
 3   3       50000 non-null  float64
 4   4       50000 non-null  float64
 5   5       50000 non-null  float64
 6   6       50000 non-null  float64
 7   7       50000 non-null  float64
 8   8       50000 non-null  float64
 9   9       50000 non-null  float64
 10  10      50000 non-null  float64
 11  11      50000 non-null  float64
 12  12      50000 non-null  float64
 13  13      50000 non-null  float64
 14  14      50000 non-null  float64
 15  15      50000 non-null  float64
 16  16      50000 non-null  float64
 17  17      50000 non-null  float64
 18  18      50000 non-null  float64
 19  19      50000 non-null  float64
 20  20      50000 non-null  float64
 21  21      50000 non-null  float64
 22  22      50000 non-null  float64
 23  23      50000 non-null  float64
 24  24      50000 non-null  float64
 25  25      50000 non-null  float64
 26  26      50000 non-null  float64
 27  27      50000 non-null  float64
 28  28      50000 non-null  float64
 29  29      50000 non-null  float64
 30  30      50000 non-null  float64
 31  31      50000 non-null  float64
 32  32      50000 non-null  float64
 33  33      50000 non-null  float64
 34  34      50000 non-null  float64
 35  35      50000 non-null  float64
 36  36      50000 non-null  float64
 37  37      50000 non-null  float64
 38  38      50000 non-null  float64
 39  39      50000 non-null  float64
 40  40      50000 non-null  float64
 41  41      50000 non-null  float64
 42  42      50000 non-null  float64
 43  43      50000 non-null  float64
 44  44      50000 non-null  float64
```

```
 45  45      50000 non-null  float64
 46  46      50000 non-null  float64
 47  47      50000 non-null  float64
 48  48      50000 non-null  float64
 49  49      50000 non-null  float64
 50  50      50000 non-null  float64
 51  51      50000 non-null  float64
 52  52      50000 non-null  float64
 53  53      50000 non-null  float64
 54  54      50000 non-null  float64
 55  55      50000 non-null  float64
 56  56      50000 non-null  float64
 57  57      50000 non-null  float64
 58  58      50000 non-null  float64
 59  59      50000 non-null  float64
 60  60      50000 non-null  float64
 61  61      50000 non-null  float64
 62  62      50000 non-null  float64
 63  63      50000 non-null  float64
 64  64      50000 non-null  float64
 65  65      50000 non-null  float64
 66  66      50000 non-null  float64
 67  67      50000 non-null  float64
 68  68      50000 non-null  float64
 69  69      50000 non-null  float64
 70  70      50000 non-null  float64
 71  71      50000 non-null  float64
 72  72      50000 non-null  float64
 73  73      50000 non-null  float64
 74  74      50000 non-null  float64
 75  75      50000 non-null  float64
 76  76      50000 non-null  float64
 77  77      50000 non-null  float64
 78  78      50000 non-null  float64
 79  79      50000 non-null  float64
 80  80      50000 non-null  float64
 81  81      50000 non-null  float64
 82  82      50000 non-null  float64
 83  83      50000 non-null  float64
 84  84      50000 non-null  float64
 85  85      50000 non-null  float64
 86  86      50000 non-null  float64
 87  87      50000 non-null  float64
 88  88      50000 non-null  float64
 89  89      50000 non-null  float64
 90  90      50000 non-null  float64
dtypes: float64(90), int64(1)
memory usage: 34.7 MB
```

0.1.4 Check for missing values.

[ ]:
```
data.isnull().sum()
```

[ ]:
```
0     0
1     0
2     0
3     0
4     0
     ..
86    0
87    0
88    0
89    0
90    0
Length: 91, dtype: int64
```

0.1.5 Check for outliers.

[ ]:
```
outliers = {}
for i in range(data.shape[1]):
    min_t = data[data.columns[i]].mean()- (3 * data[data.columns[i]].std())
    max_t = data[data.columns[i]].mean()+ (3 * data[data.columns[i]].std())
    count = 0
    for j in data[data.columns[i]]:
        if j < min_t or j > max_t:
            count += 1
    percentage = count / data.shape[0]
    outliers[data.columns[i]] = "%.3f" % percentage
outliers
```

[ ]:
```
{'0': '0.009',
 '1': '0.011',
 '10': '0.007',
 '11': '0.008',
 '12': '0.014',
 '13': '0.015',
 '14': '0.016',
 '15': '0.022',
 '16': '0.015',
 '17': '0.016',
 '18': '0.019',
 '19': '0.020',
 '2': '0.013',
 '20': '0.016',
 '21': '0.016',
```

'22': '0.015',
'23': '0.015',
'24': '0.017',
'25': '0.015',
'26': '0.016',
'27': '0.021',
'28': '0.019',
'29': '0.016',
'3': '0.012',
'30': '0.014',
'31': '0.017',
'32': '0.017',
'33': '0.015',
'34': '0.012',
'35': '0.016',
'36': '0.016',
'37': '0.011',
'38': '0.021',
'39': '0.014',
'4': '0.014',
'40': '0.015',
'41': '0.017',
'42': '0.017',
'43': '0.015',
'44': '0.020',
'45': '0.017',
'46': '0.014',
'47': '0.019',
'48': '0.023',
'49': '0.015',
'5': '0.004',
'50': '0.015',
'51': '0.017',
'52': '0.016',
'53': '0.016',
'54': '0.016',
'55': '0.018',
'56': '0.020',
'57': '0.020',
'58': '0.010',
'59': '0.014',
'6': '0.016',
'60': '0.015',
'61': '0.014',
'62': '0.015',
'63': '0.021',
'64': '0.021',

```
 '65': '0.019',
 '66': '0.023',
 '67': '0.017',
 '68': '0.024',
 '69': '0.018',
 '7': '0.012',
 '70': '0.015',
 '71': '0.012',
 '72': '0.022',
 '73': '0.018',
 '74': '0.015',
 '75': '0.014',
 '76': '0.012',
 '77': '0.022',
 '78': '0.014',
 '79': '0.013',
 '8': '0.008',
 '80': '0.018',
 '81': '0.020',
 '82': '0.019',
 '83': '0.014',
 '84': '0.019',
 '85': '0.016',
 '86': '0.019',
 '87': '0.020',
 '88': '0.013',
 '89': '0.017',
 '9': '0.013',
 '90': '0.021'}
```


0.1.6 Data Rescaling

[ ]:
```
X = data.iloc[:, 1:]
Y = data.iloc[:, 0]
X = (X - X.min()) / (X.max() - X.min())
X.head()
```

[ ]:

	1	2	3	…	88	89	90
0	0.860844	0.520015	0.638629	…	0.499917	0.542475	0.459884
1	0.837937	0.514852	0.633315	…	0.529855	0.551845	0.495266
2	0.880010	0.537641	0.605290	…	0.503723	0.549798	0.455684
3	0.828773	0.480350	0.567581	…	0.525228	0.567246	0.483691
4	0.880257	0.555214	0.627082	…	0.520872	0.551543	0.497890

[5 rows x 90 columns]

[ ]:
```
Y.head()
```

[ ]:
```
0    2001
1    2001
2    2001
3    2001
4    2001
Name: 0, dtype: int64
```

0.1.7 Splitting the Dataset

[ ]:
```
X.shape

train_end = int(len(X) * 0.8)
dev_end = int(len(X) * 0.9)

X_shuffle = X.sample(frac=1, random_state=0)
Y_shuffle = Y.sample(frac=1, random_state=0)

x_train = X_shuffle.iloc[:train_end,:]
y_train = Y_shuffle.iloc[:train_end]
x_dev = X_shuffle.iloc[train_end:dev_end,:]
y_dev = Y_shuffle.iloc[train_end:dev_end]
x_test = X_shuffle.iloc[dev_end:,:]
y_test = Y_shuffle.iloc[dev_end:]

print(x_train.shape, y_train.shape)
print(x_dev.shape, y_dev.shape)
print(x_test.shape, y_test.shape)
```

```
(40000, 90) (40000,)
(5000, 90) (5000,)
(5000, 90) (5000,)
```

[ ]:

0.1.8 Part 2: Create and Train the model

0.1.9 import relevant modules

0.1.10 Convert the DataFrames into tensors.

[ ]:
```
x_train = t.tensor(x_train.values).float()
y_train = t.tensor(y_train.values).float()
x_dev = t.tensor(x_dev.values).float()
y_dev = t.tensor(y_dev.values).float()
x_test = t.tensor(x_test.values).float()
y_test = t.tensor(y_test.values).float()
```

[ ]:
```
print(x_train.shape)
print(y_train.shape)
```

```
torch.Size([40000, 90])
torch.Size([40000])
```

[ ]:
```
x_train.shape[1]
```

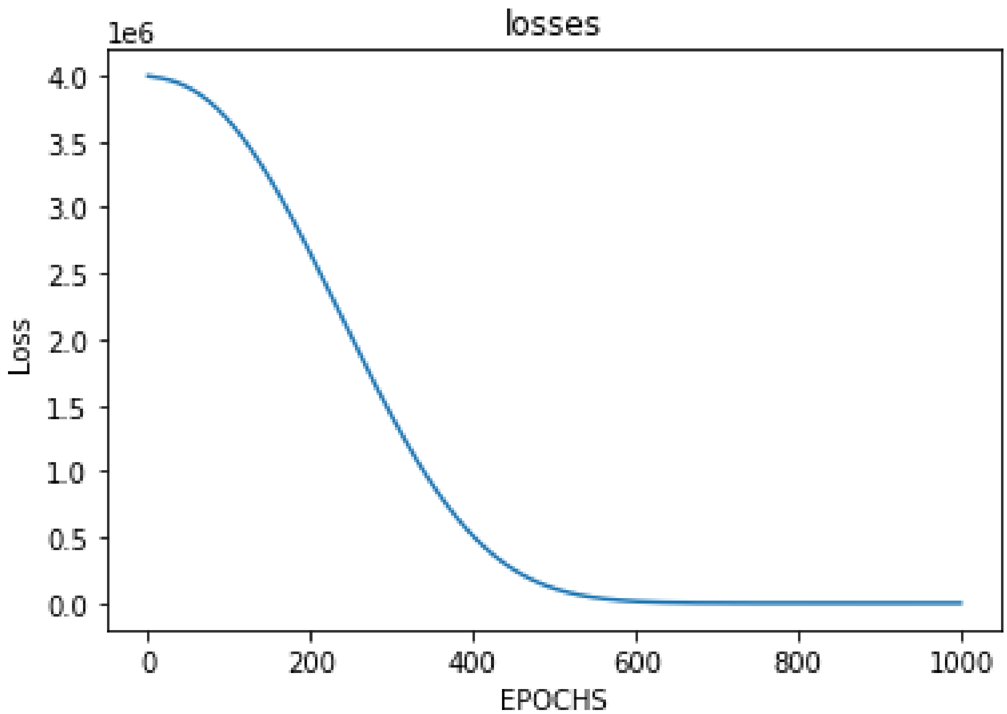
[ ]: 90

0.1.11 Model: Compile and fit

[ ]:
```
# Model architecture
model = nn.Sequential(nn.Linear(x_train.shape[1], 100),nn.Linear(100, 1))
# Loss
loss_function = t.nn.MSELoss()
# Optimizer
optimizer = t.optim.Adam(model.parameters(), lr=0.001)
# Training

EPOCHS=1000

losses = []
for i in range(EPOCHS):
    y_pred = model(x_train).squeeze()
    loss = loss_function(y_pred, y_train)
    losses.append(loss.item())
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    if i%100 == 0:
        print(i, loss.item())
```

```
pred = model(x_test[0])
print("Ground truth:", y_test[0].item(), "Prediction:",pred.item())

plt.plot(range(0,1000), losses)
plt.title('losses')
plt.xlabel('EPOCHS')
plt.ylabel('Loss')
plt.show()
```

```
0 3994038.75
100 3651263.75
200 2646766.0
300 1414638.75
400 508934.09375
500 111702.671875
600 15032.6611328125
700 2055.36865234375
800 1005.4210205078125
900 946.11279296875
Ground truth: 2005.0 Prediction: 2005.5205078125
```



1 Part 3 (Optional): Implement validation and testing using the whole validation and test datasets.

This part is optional, but it will useful to, at least, to try to implement, they are not difficult however. 1. Implement a validation step after each training epoch, you can use the first register of the validation dataset, or the whole validation dataset. 2. Test the model using more than a single sample from the test dataset.

Mandatory: Think about a function that uses the gradient but it can be calculated, so how to calculate the gradient during the training process.

Imagine you have the gradient descent. Imagine you cannot pass the funtion the gradient. So how to solve the problem of calculating the gradient without using the gradient descent. SO is it still possible to train the NN? Think of example of X square. There are methods without usng the function, but we need find the gradient descent without derivatives, i.e, without doing derivatives.

[ ]:
```
# Extract all predictions
pred = model(x_test)
print(pred)
```

```
tensor([[2005.5205],
        [1979.3074],
        [2028.5500],
        …,
        [1980.4119],
        [1962.4541],
        [1992.7382]], grad_fn=<AddmmBackward>)
```

[ ]:
```
import math
loss = loss_function(pred,y_test).item()
print("Square root of MSE: ",math.sqrt(loss))
```

```
Square root of MSE:  30.660170931404945
```

```
/usr/local/lib/python3.7/dist-packages/torch/nn/modules/loss.py:528:
UserWarning: Using a target size (torch.Size([5000])) that is different to the
input size (torch.Size([5000, 1])). This will likely lead to incorrect results
due to broadcasting. Please ensure they have the same size.
  return F.mse_loss(input, target, reduction=self.reduction)
```

[ ]:
```
abs_loss = nn.L1Loss()
loss  = abs_loss(pred, y_test).item()
print("Avg of absolute error of test: ", loss)
```

```
Avg of absolute error of test:  23.342870712280273
```

```
/usr/local/lib/python3.7/dist-packages/torch/nn/modules/loss.py:96: UserWarning:
Using a target size (torch.Size([5000])) that is different to the input size
(torch.Size([5000, 1])). This will likely lead to incorrect results due to
```

```
broadcasting. Please ensure they have the same size.
  return F.l1_loss(input, target, reduction=self.reduction)
```

[ ]:
```python
# MSE
from sklearn.metrics import mean_squared_error

pred = pd.DataFrame(pred)
labels = pd.DataFrame(y_test)
mean_squared_error(labels, pred)
```

[ ]:
```
940.7669028933734
```

2 Validation loss by epochs

[ ]:
```python
# Model
'''
model = nn.Sequential(nn.Linear(x_train.shape[1], 100), \
nn.ReLU(), \
nn.Linear(100, 50), \
nn.ReLU(), \
nn.Linear(50, 25), \
nn.ReLU(), \
nn.Linear(25, 1))
'''
model = nn.Sequential(nn.Linear(x_train.shape[1], 100),nn.Linear(100, 1))
# Loss
loss_function = t.nn.MSELoss()
# Optimizer
optimizer = t.optim.Adam(model.parameters(), lr=0.001)

# Validation

EPOCHS=1000
# Training


losses_val = []
losses_train = []
for i in range(EPOCHS):

    # clear previous gradient computation
    optimizer.zero_grad()

    # forward propagation
    y_pred = model(x_train)
```

```
    # calculate the loss
    loss = loss_function(y_pred, y_train)
    # update average loss
    losses_train.append(loss.item())

    # backpropagate to compute gradients
    loss.backward()

    # update model weights
    optimizer.step()
    if i%100 == 0:
        print("training: ")
        print(i, loss.item())

# validation
for i in range(EPOCHS):
    # clear previous gradient computation
    optimizer.zero_grad()

    # forward propagation
    y_pred_val = model(x_dev)

    # calculate the loss
    loss = loss_function(y_pred_val, y_dev)

    # update average loss
    losses_val.append(loss.item())

    loss.backward()
    # update model weights
    optimizer.step()
    if i%100 == 0:
      print("val: ")
      print(i, loss.item())

#pred = model(x_test[0])
#print("Ground truth:", y_test[0].item(), "Prediction:",pred.item())

plt.plot(range(0,1000), losses)
plt.title('losses over validation set')
plt.xlabel('EPOCHS')
plt.ylabel('Loss')
plt.show()
plt.show()
#Substract the prediction form real target, calculate abs val,
```

```python
plt.figure(figsize=(10,6))
plt.plot(losses_train, '-o', label='Training loss')
plt.plot(losses_val, '-o', label='Validation loss')
plt.legend()
plt.title('Learning curves in Training and validation stage')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.show()
```

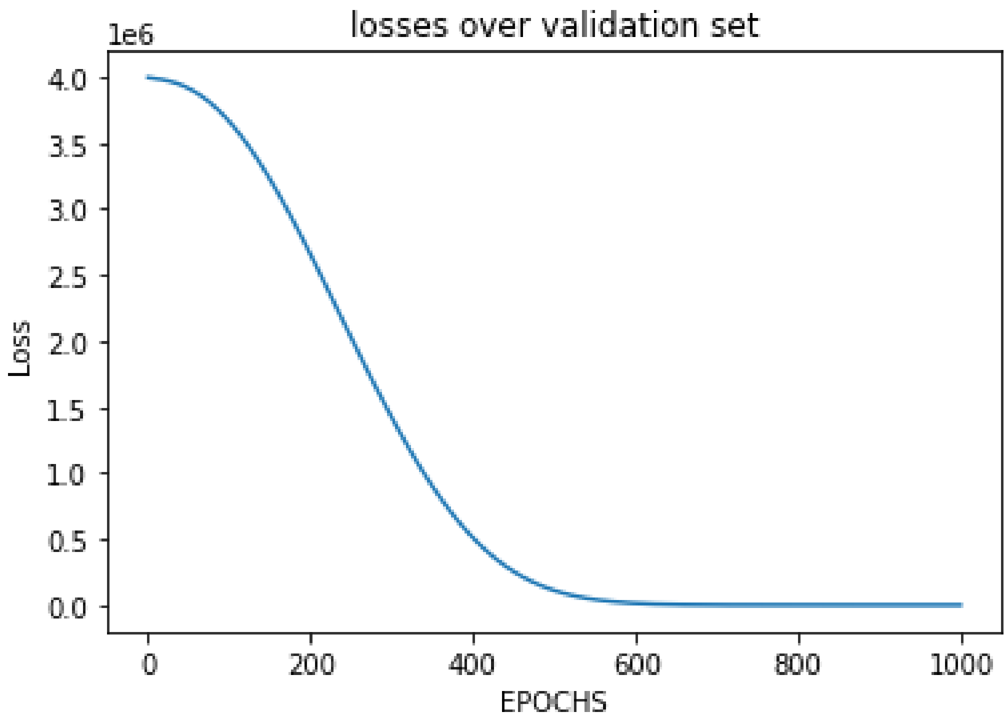
```
/usr/local/lib/python3.7/dist-packages/torch/nn/modules/loss.py:528:
UserWarning: Using a target size (torch.Size([40000])) that is different to the
input size (torch.Size([40000, 1])). This will likely lead to incorrect results
due to broadcasting. Please ensure they have the same size.
  return F.mse_loss(input, target, reduction=self.reduction)
```

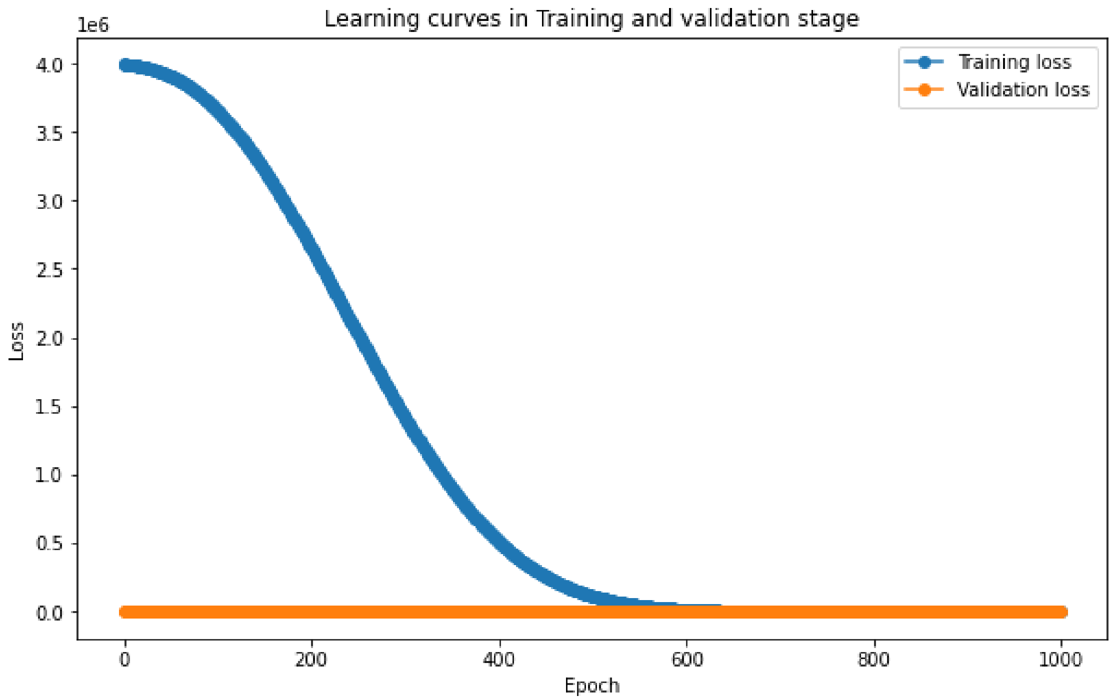
```
training:
0 3993796.75
training:
100 3651965.0
training:
200 2646987.5
training:
300 1414304.5
training:
400 508555.84375
training:
500 111535.5859375
training:
600 14994.9248046875
training:
700 2047.8155517578125
training:
800 1001.3643798828125
training:
900 942.21923828125
```

```
/usr/local/lib/python3.7/dist-packages/torch/nn/modules/loss.py:528:
UserWarning: Using a target size (torch.Size([5000])) that is different to the
input size (torch.Size([5000, 1])). This will likely lead to incorrect results
due to broadcasting. Please ensure they have the same size.
  return F.mse_loss(input, target, reduction=self.reduction)
```

```
val:
0 924.1326904296875
val:
100 918.7447509765625
val:
200 914.0256958007812
```

```
val:
300 909.0357055664062
val:
400 903.7448120117188
val:
500 898.1561279296875
val:
600 892.26904296875
val:
700 886.0803833007812
val:
800 879.5925903320312
val:
900 872.8035888671875
```





[ ]:
```
# Training

EPOCHS=1000

losses_val = []
losses_train = []
for i in range(EPOCHS):

    # clear previous gradient computation
    optimizer.zero_grad()

    # forward propagation
    y_pred = model(x_train)

    # calculate the loss
    loss = loss_function(y_pred, y_train)
    # update average loss
    losses_train.append(loss.item())

    # backpropagate to compute gradients
    loss.backward()

    # update model weights
    optimizer.step()
```

```python
    if i%100 == 0:
        print("training: ")
        print(i, loss.item())

# validation
for i in range(EPOCHS):
    # clear previous gradient computation
    optimizer.zero_grad()

    # forward propagation
    y_pred_val = model(x_dev)

    # calculate the loss
    loss = loss_function(y_pred_val, y_dev)

    # update average loss
    losses_val.append(loss.item())

    loss.backward()
    # update model weights
    optimizer.step()
    if i%100 == 0:
      print("val: ")
      print(i, loss.item())



#pred = model(x_test[0])
#print("Ground truth:", y_test[0].item(), "Prediction:",pred.item())
```

```
/usr/local/lib/python3.7/dist-packages/torch/nn/modules/loss.py:528:
UserWarning: Using a target size (torch.Size([5000])) that is different to the
input size (torch.Size([5000, 1])). This will likely lead to incorrect results
due to broadcasting. Please ensure they have the same size.
  return F.mse_loss(input, target, reduction=self.reduction)
```

```
val:
0 923.8046875
val:
100 919.3480834960938
val:
200 914.6468505859375
val:
300 909.6446533203125
val:
400 904.3450317382812
val:
500 898.7448120117188
```

```
val:
600 892.8479614257812
val:
700 886.6480712890625
val:
800 880.1478271484375
val:
900 873.349853515625
```

[ ]: