MACHINE LEARNING LAB

MINIPROJECT - PYTORCH

Juan Fernando Espinosa

303158

1. DATA PRE-PROCESSING

In [20]:

```
import torch
from torch.autograd import Variable
from sklearn.model_selection import train_test_split
import torch.nn.functional as F
import torch.utils.data as Data
import pandas as pd
import numpy as np
from collections import Counter
import math
import matplotlib.pyplot as plt
```

In [21]:

```
missing_values = ['-','na','Nan','nan','n/a','?']

Wine = pd.read_csv("winequality-white.csv", sep=';', na_values = mi
ssing_values)
Wine.head()
```

Out[21]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	al
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	
4	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	

Data normalization

In [22]:

```
def normalize(dataset):
    dataNorm=((dataset-dataset.min())/(dataset.max()-dataset.min())
)
    dataNorm["quality"]=dataset["quality"]
    return dataNorm
Wine = normalize(Wine)
Wine.head()
```

Out[22]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	p
0	0.307692	0.186275	0.216867	0.308282	0.106825	0.149826	0.373550	0.267785	0.25454
1	0.240385	0.215686	0.204819	0.015337	0.118694	0.041812	0.285383	0.132832	0.52727
2	0.413462	0.196078	0.240964	0.096626	0.121662	0.097561	0.204176	0.154039	0.49090
3	0.326923	0.147059	0.192771	0.121166	0.145401	0.156794	0.410673	0.163678	0.42727
4	0.326923	0.147059	0.192771	0.121166	0.145401	0.156794	0.410673	0.163678	0.42727

Check number of classes in the dataset

In [23]:

```
test_classes = Wine.quality
output = len(set(test_classes))
print('Number of classes:', output)
```

```
Number of classes: 7
```

2. INITIALIZATION OF THE REGRESSION

Partition of the dataset

In [24]:

```
y = Wine['quality'].values
X =  Wine.drop(['quality'], axis=1).values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)
X_train.shape, y_train.shape
```

Out[24]:

```
((3428, 11), (3428,))
```

Initialization of the regression and the layer structure

Considering the given info: input / 3 FC layers / output.

In [57]:

```
network = torch.nn.Sequential(
        torch.nn.Linear(11, 64), # input
        torch.nn.Sigmoid(),
        torch.nn.Linear(64, 64),# First hidden layer
        torch.nn.Sigmoid(),
        torch.nn.Linear(64, 64),# Second hidden layer
        torch.nn.Sigmoid(),
        torch.nn.Linear(64, 64),# Third hidden layer
        torch.nn.Sigmoid(),
        torch.nn.Linear(64, 1),
    )
optimizer = torch.optim.Adam(network.parameters(), lr=0.01)
loss = torch.nn.MSELoss()

batch_sizeT = 5
epochs = 200

# Transform Numpy array to Tensor variables.
X = torch.from_numpy(X_train)
y = torch.from_numpy(y_train)

datasets = torch.utils.data.TensorDataset(X, y)

loader = Data.DataLoader(
    dataset=datasets,
    batch_size = batch_sizeT,
    shuffle=True, num_workers=0,)
plt.ion()
loss_array = []
for epoch in range(epochs):

    for step, (batch_X, batch_y) in enumerate(loader): # for each training step
        X_temporal = Variable(batch_X)
        y_temporal = Variable(batch_y)
        prediction = network(X_temporal.float())     # input x and predict based on x
        losse = loss(prediction, y_temporal.float())
        loss_array.append(losse.data)


        optimizer.zero_grad()  # It is required to clear the gradients
        losse.backward()         # backpropagation
        optimizer.step()
    print(f"{epoch+1} epoch | loss = {losse}")
plt.plot(loss_array)
```

```
1 epoch | loss = 0.8605697751045227
2 epoch | loss = 0.11455386132001877
3 epoch | loss = 1.528577446937561
4 epoch | loss = 0.27014482021331787
```

```
5 epoch | loss = 0.6668551564216614
6 epoch | loss = 0.6890454292297363
7 epoch | loss = 0.3706584572792053
8 epoch | loss = 1.1950761079788208
9 epoch | loss = 0.4285852313041687
10 epoch | loss = 0.954093337059021
11 epoch | loss = 0.24326680600643158
12 epoch | loss = 0.4730881154537201
13 epoch | loss = 0.3541933000087738
14 epoch | loss = 0.44385847449302673
15 epoch | loss = 0.8166089057922363
16 epoch | loss = 2.9025466442108154
17 epoch | loss = 0.6668533086776733
18 epoch | loss = 0.4962048828601837
19 epoch | loss = 0.0025077499449253082
20 epoch | loss = 1.849060297012329
21 epoch | loss = 1.5592761039733887
22 epoch | loss = 1.6630982160568237
23 epoch | loss = 1.714152455329895
24 epoch | loss = 0.230685293674469
25 epoch | loss = 0.03535465523600578
26 epoch | loss = 0.8107268214225769
27 epoch | loss = 1.2157328128814697
28 epoch | loss = 1.5101038217544556
29 epoch | loss = 0.3343123495578766
30 epoch | loss = 0.6873752474784851
31 epoch | loss = 1.1954193115234375
32 epoch | loss = 0.0270451121032238
33 epoch | loss = 0.3883790969848633
34 epoch | loss = 0.06888662278652191
35 epoch | loss = 0.0005603685276582837
36 epoch | loss = 0.3156583905220032
37 epoch | loss = 1.2889164686203003
38 epoch | loss = 1.1580111980438232
39 epoch | loss = 0.22049792110919952
40 epoch | loss = 2.6846518516540527
41 epoch | loss = 0.7522990107536316
42 epoch | loss = 0.5706974267959595
43 epoch | loss = 0.261165052652359
44 epoch | loss = 0.40251773595809937
45 epoch | loss = 0.2242138832807541
46 epoch | loss = 0.9314168095588684
47 epoch | loss = 0.22414565086364746
48 epoch | loss = 0.9176021218299866
49 epoch | loss = 0.7638780474662781
50 epoch | loss = 0.0028948106337338686
51 epoch | loss = 0.5321009159088135
52 epoch | loss = 1.2428784370422363
53 epoch | loss = 0.9163723587989807
54 epoch | loss = 0.5898261666297913
55 epoch | loss = 0.4830879271030426
56 epoch | loss = 0.24286368489265442
57 epoch | loss = 1.6315722465515137
58 epoch | loss = 0.692040741443634
```

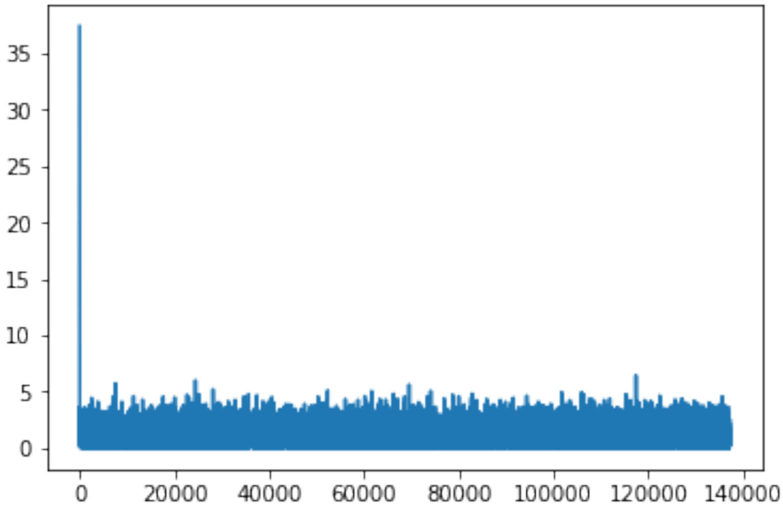
```
59 epoch | loss = 1.925269603729248
60 epoch | loss = 0.28427496552467346
61 epoch | loss = 0.0937444418668747
62 epoch | loss = 2.5331013202667236
63 epoch | loss = 0.8991101384162903
64 epoch | loss = 1.0247712135314941
65 epoch | loss = 3.6971747875213623
66 epoch | loss = 2.026449203491211
67 epoch | loss = 0.23425765335559845
68 epoch | loss = 1.2255176305770874
69 epoch | loss = 0.7042745351791382
70 epoch | loss = 1.2265645265579224
71 epoch | loss = 0.671364426612854
72 epoch | loss = 0.6342029571533203
73 epoch | loss = 0.77750563621521
74 epoch | loss = 0.9502670764923096
75 epoch | loss = 1.5697063207626343
76 epoch | loss = 0.7003514170646667
77 epoch | loss = 1.5580167770385742
78 epoch | loss = 0.22607126832008362
79 epoch | loss = 0.35795092582702637
80 epoch | loss = 0.8999313712120056
81 epoch | loss = 0.9837948679924011
82 epoch | loss = 0.4139083921909332
83 epoch | loss = 0.9482316970825195
84 epoch | loss = 0.6772665977478027
85 epoch | loss = 1.9016300439834595
86 epoch | loss = 0.3681350350379944
87 epoch | loss = 0.03665259853005409
88 epoch | loss = 2.004359722137451
89 epoch | loss = 0.2541216015815735
90 epoch | loss = 0.2849070429801941
91 epoch | loss = 1.2846436500549316
92 epoch | loss = 0.7482082843780518
93 epoch | loss = 0.2360309362411499
94 epoch | loss = 2.052640199661255
95 epoch | loss = 0.6938400864601135
96 epoch | loss = 0.8212472796440125
97 epoch | loss = 1.7859586477279663
98 epoch | loss = 0.7033320665359497
99 epoch | loss = 0.23627635836601257
100 epoch | loss = 2.897874593734741
101 epoch | loss = 2.9971020221710205
102 epoch | loss = 0.3665236234664917
103 epoch | loss = 1.2920193672180176
104 epoch | loss = 0.4079608619213104
105 epoch | loss = 0.1761009842157364
106 epoch | loss = 1.1633164882659912
107 epoch | loss = 0.3938685357570648
108 epoch | loss = 5.083630084991455
109 epoch | loss = 0.7193038463592529
110 epoch | loss = 0.3603970408439636
111 epoch | loss = 0.2737374007701874
112 epoch | loss = 2.854562282562256
```

```
113 epoch | loss = 0.22720478475093842
114 epoch | loss = 0.6885574460029602
115 epoch | loss = 1.791379451751709
116 epoch | loss = 0.22285780310630798
117 epoch | loss = 0.0039308457635343075
118 epoch | loss = 0.287149578332901
119 epoch | loss = 0.7214820384979248
120 epoch | loss = 1.2532192468643188
121 epoch | loss = 0.5286235809326172
122 epoch | loss = 0.33730852603912354
123 epoch | loss = 0.25996914505958557
124 epoch | loss = 0.008113938383758068
125 epoch | loss = 0.12829157710075378
126 epoch | loss = 3.1530728340148926
127 epoch | loss = 0.26047855615615845
128 epoch | loss = 0.9153937101364136
129 epoch | loss = 0.3646056652069092
130 epoch | loss = 1.7094879150390625
131 epoch | loss = 1.677517294883728
132 epoch | loss = 0.22596386075019836
133 epoch | loss = 2.200864553451538
134 epoch | loss = 2.7689766883850098
135 epoch | loss = 0.08124992251396179
136 epoch | loss = 1.346655011177063
137 epoch | loss = 0.27256277203559875
138 epoch | loss = 0.25988879799842834
139 epoch | loss = 3.6680846214294434
140 epoch | loss = 0.9796450734138489
141 epoch | loss = 1.5555717945098877
142 epoch | loss = 0.27516230940818787
143 epoch | loss = 0.675903856754303
144 epoch | loss = 1.075715184211731
145 epoch | loss = 0.04179537296295166
146 epoch | loss = 0.1924344301223755
147 epoch | loss = 0.2332366406917572
148 epoch | loss = 0.736602783203125
149 epoch | loss = 0.5185887813568115
150 epoch | loss = 0.6671059131622314
151 epoch | loss = 0.24717862904071808
152 epoch | loss = 0.5947138071060181
153 epoch | loss = 8.516144589520991e-06
154 epoch | loss = 1.0305638313293457
155 epoch | loss = 2.672248601913452
156 epoch | loss = 0.9592097997665405
157 epoch | loss = 1.4985147714614868
158 epoch | loss = 0.2226729393005371
159 epoch | loss = 0.8525935411453247
160 epoch | loss = 0.25773024559020996
161 epoch | loss = 2.051586627960205
162 epoch | loss = 0.8440478444099426
163 epoch | loss = 0.5479058623313904
164 epoch | loss = 0.7232239842414856
165 epoch | loss = 1.1365852355957031
166 epoch | loss = 0.22714947164058685
```

```
167 epoch | loss = 0.6758930683135986
168 epoch | loss = 0.41186419129371643
169 epoch | loss = 0.692514538764953 6
170 epoch | loss = 1.9722868204116821
171 epoch | loss = 1.3260842561721802
172 epoch | loss = 0.26249146461486816
173 epoch | loss = 1.3600261211395264
174 epoch | loss = 0.5844133496284485
175 epoch | loss = 0.7529309391975403
176 epoch | loss = 2.6761536598205566
177 epoch | loss = 0.3040260970592499
178 epoch | loss = 0.032547734677791595
179 epoch | loss = 0.2937135696411133
180 epoch | loss = 0.7880871891975403
181 epoch | loss = 0.288226842880249
182 epoch | loss = 0.7215981483459473
183 epoch | loss = 0.3783431649208069
184 epoch | loss = 0.9255162477493286
185 epoch | loss = 1.780129075050354
186 epoch | loss = 3.0108718872070312
187 epoch | loss = 0.31547626852989197
188 epoch | loss = 0.26069578528404236
189 epoch | loss = 0.7221987843513489
190 epoch | loss = 0.30328133702278137
191 epoch | loss = 0.6055524349212646
192 epoch | loss = 0.0031095927115529776
193 epoch | loss = 1.17967689037323
194 epoch | loss = 0.2249920666217804
195 epoch | loss = 0.24989885091781616
196 epoch | loss = 1.7622992992401123
197 epoch | loss = 0.20433947443962097
198 epoch | loss = 0.002308703726157546
199 epoch | loss = 0.26140064001083374
200 epoch | loss = 0.9518703818321228
```

Out[57]:

```
[<matplotlib.lines.Line2D at 0x12f3f6b90>]
```



In [59]:

```
X_testing = torch.from_numpy(X_test)
y_testing = torch.from_numpy(y_test)
prediction = network(X_testing.float())
losse = loss(prediction, y_testing.float())


print("The test loss is {:.2}".format(losse.data.item()))
```

```
The test loss is 0.77
```

Observations

1. Although the RMSE of the prediction is **low** it lacks to predict correctly the quality for which each wine belongs.

2. By searching for a solution of why the behavior is poor the description of the target column "quality" explicitly mentioned that the values are based on **sensory data** which could be named as perception results, therefore the prediction is not easy to execute.

3. It is always good practice to execute **cross-validation** for all hyperparameters. While executing the model it is not possible to have certainty of which combination of nodes in hidden layers, batch size, etc., is the optimal.

4. Moreover, for curiosity, I tested the model with different activation functions. **Leaky ReLu** returns a RMSE around 0.7 (Cross-validated | hyperparameters: batch size, # nodes in hidden layers) which is slightly lower in comparison with the Sigmoid function presented in the model above. Additionally, the convergence of the loss is better. (check 4. Tests)

3. BIBLIOGRAPHY

1. P. Cortez, A. Cerdeira, F. Almeida, T. Matos and J. Reis. Modeling wine preferences by data mining from physicochemical properties. In Decision Support Systems, Elsevier, 47(4):547-553, 2009.

2. Phillips, B. (2018, December 15). Simple Regression with Neural Networks in PyTorch. Retrieved from https://medium.com/@benjamin.phillips22/simple-regression-with-neural-networks-in-pytorch-313f06910379.

3. Tsvetkov, V. (2019, September 17). A comprehensive intro to PyTorch. Retrieved from https://blog.tensorpad.com/a-comprehensive-intro-to-pytorch/.

4. TESTS

Leaky-ReLu as activation function

In [68]:

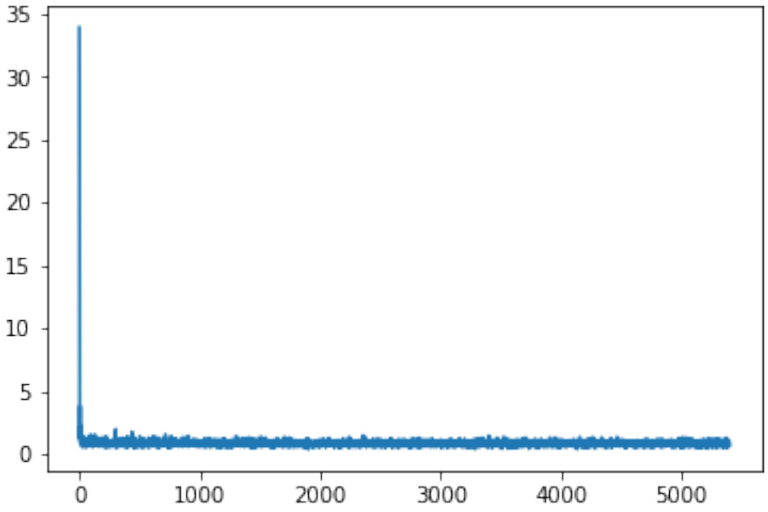
```
network = torch.nn.Sequential(
        torch.nn.Linear(11, 150), # input
        torch.nn.LeakyReLU(),
        torch.nn.Linear(150, 150),# First hidden layer
        torch.nn.LeakyReLU(),
        torch.nn.Linear(150, 150),# Second hidden layer
        torch.nn.LeakyReLU(),
        torch.nn.Linear(150, 150),# Third hidden layer
        torch.nn.LeakyReLU(),
        torch.nn.Linear(150, 1),
    )
optimizer = torch.optim.Adam(network.parameters(), lr=0.01)
loss = torch.nn.MSELoss()

batch_sizeT = 64
epochs = 100

# Transform Numpy array to Tensor variables.
X = torch.from_numpy(X_train)
y = torch.from_numpy(y_train)

datasets = torch.utils.data.TensorDataset(X, y)

loader = Data.DataLoader(
    dataset=datasets,
    batch_size = batch_sizeT,
    shuffle=True, num_workers=0,)
plt.ion()
loss_array = []
for epoch in range(epochs):

    for step, (batch_X, batch_y) in enumerate(loader): # for each training step
        X_temporal = Variable(batch_X)
        y_temporal = Variable(batch_y)
        prediction = network(X_temporal.float())     # input x and predict based on x
        losse = loss(prediction, y_temporal.float())
        loss_array.append(losse.data)


        optimizer.zero_grad()  # It is required to clear the gradients
        losse.backward()         # backpropagation
        optimizer.step()
    print(f"{epoch+1} epoch | loss = {losse}")
plt.plot(loss_array)
```

```
1 epoch | loss = 0.5057490468025208
2 epoch | loss = 0.913483917713165З
3 epoch | loss = 0.8051160573959351
4 epoch | loss = 0.4858332574367523
```

```
5 epoch | loss = 0.8255210518836975
6 epoch | loss = 0.6772353649139404
7 epoch | loss = 1.1681170463562012
8 epoch | loss = 0.8066381812095642
9 epoch | loss = 0.950116753578186
10 epoch | loss = 0.6099951863288879
11 epoch | loss = 0.6362884044647217
12 epoch | loss = 0.7095951437950134
13 epoch | loss = 0.7884309887886047
14 epoch | loss = 0.5140870809555054
15 epoch | loss = 0.9416120648384094
16 epoch | loss = 1.026793360710144
17 epoch | loss = 0.9579735398292542
18 epoch | loss = 0.7208535075187683
19 epoch | loss = 0.5793512463569641
20 epoch | loss = 0.7516117691993713
21 epoch | loss = 0.8176302313804626
22 epoch | loss = 1.067955732345581
23 epoch | loss = 0.7560921907424927
24 epoch | loss = 0.7811601161956787
25 epoch | loss = 0.8453911542892456
26 epoch | loss = 0.8882365822792053
27 epoch | loss = 0.7066816687583923
28 epoch | loss = 0.7036915421485901
29 epoch | loss = 0.8689120411872864
30 epoch | loss = 0.5448341369628906
31 epoch | loss = 0.9946936368942261
32 epoch | loss = 0.9161320924758911
33 epoch | loss = 0.8449593782424927
34 epoch | loss = 1.0660035610198975
35 epoch | loss = 0.7410057783126831
36 epoch | loss = 0.5122541785240173
37 epoch | loss = 0.6275328993797302
38 epoch | loss = 0.9769017696380615
39 epoch | loss = 0.7311204671859741
40 epoch | loss = 1.0889606475830078
41 epoch | loss = 1.0274133682250977
42 epoch | loss = 0.7508194446563721
43 epoch | loss = 0.6605762839317322
44 epoch | loss = 0.7440598607063293
45 epoch | loss = 0.8548521399497986
46 epoch | loss = 1.0203204154968262
47 epoch | loss = 1.0871562957763672
48 epoch | loss = 0.6453996300697327
49 epoch | loss = 0.6241939067840576
50 epoch | loss = 0.7334551811218262
51 epoch | loss = 0.6328409314155579
52 epoch | loss = 0.7638850808143616
53 epoch | loss = 0.9388380646705627
54 epoch | loss = 0.7914954423904419
55 epoch | loss = 0.8595948815345764
56 epoch | loss = 0.8168482780456543
57 epoch | loss = 0.608056902885437
58 epoch | loss = 0.6956735849380493
```

```
59 epoch | loss = 0.8480135202407837
60 epoch | loss = 0.6677202582359314
61 epoch | loss = 0.6973062753677368
62 epoch | loss = 1.2552589178085327
63 epoch | loss = 1.4543849229812622
64 epoch | loss = 0.6765437722206116
65 epoch | loss = 0.6892909407615662
66 epoch | loss = 0.4258921444416046
67 epoch | loss = 1.1097345352172852
68 epoch | loss = 1.0656335353851318
69 epoch | loss = 0.7291654348373413
70 epoch | loss = 0.6026569604873657
71 epoch | loss = 1.180633783340454
72 epoch | loss = 0.7922965884208679
73 epoch | loss = 0.732864260673523
74 epoch | loss = 0.4980090856552124
75 epoch | loss = 0.5979751348495483
76 epoch | loss = 0.5923035144805908
77 epoch | loss = 1.3493902683258057
78 epoch | loss = 0.9289383292198181
79 epoch | loss = 0.7724866271018982
80 epoch | loss = 0.620320737361908
81 epoch | loss = 1.0752524137496948
82 epoch | loss = 0.8210203051567078
83 epoch | loss = 0.6196243166923523
84 epoch | loss = 0.8014512658119202
85 epoch | loss = 0.7797408699989319
86 epoch | loss = 1.0861241817474365
87 epoch | loss = 0.664879322052002
88 epoch | loss = 0.5390986204147339
89 epoch | loss = 0.6471565365791321
90 epoch | loss = 0.5595607161521912
91 epoch | loss = 0.5940065383911133
92 epoch | loss = 1.0776687860488892
93 epoch | loss = 1.2231833934783936
94 epoch | loss = 0.6032888889312744
95 epoch | loss = 1.0783177614212036
96 epoch | loss = 1.2006195783615112
97 epoch | loss = 0.7854986190795898
98 epoch | loss = 0.4510979652404785
99 epoch | loss = 0.5238127112388611
100 epoch | loss = 0.7707371711730957
```

```
Out[68]: [<matplotlib.lines.Line2D at 0x12778fdd0>]
```



In [69]:

```
loss_test = []
X_testing = torch.from_numpy(X_test)
y_testing = torch.from_numpy(y_test)
prediction = network(X_testing.float())
losse = loss(prediction, y_testing.float())



print("The test loss is {:.2}".format(losse.data.item()))
```

```
The test loss is 0.75
```