```
from collections import OrderedDict
import pandas as pd
from sklearn.model_selection import train_test_split
import torch
import torch.nn as nn
import torch.nn.functional as F
```

Data

```
data = pd.read_csv('examen.csv', header=None)
y = data.iloc[:, 0].values
X = data.iloc[:, 1:].values
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
X_train = (X_train) / X_train.max(axis=0)
X_test = (X_test) / X_train.max(axis=0)

X_train = torch.Tensor(X_train)
X_test = torch.Tensor(X_test)

y_train = torch.Tensor(y_train).reshape(-1, 1)
y_test = torch.Tensor(y_test).reshape(-1, 1)
```

Proposed architecture

Built in Pytorch, the proposed architecture is 2 hidden layers, each with 5 neural units and Sigmoid ac

```
exam = nn.Sequential(
    nn.Linear(4, 5),
    nn.Sigmoid(),
    nn.Linear(5, 5),
    nn.Sigmoid(),
    nn.Linear(5, 5),
    nn.Sigmoid(),
    nn.Linear(5, 1)
)

exam
```

```
Sequential(
  (0): Linear(in_features=4, out_features=5, bias=True)
  (1): Sigmoid()
```

```
for i in exam:
  print(i)
  print(i._parameters)
```

```
Linear(in_features=4, out_features=5, bias=True)
OrderedDict([('weight', Parameter containing:
tensor([[-0.1337,  0.0874, -0.1069, -0.2706],
        [ 0.1733,  0.3348,  0.2583,  0.1232],
        [-0.0607, -0.1557, -0.3172,  0.3525],
        [-0.2465,  0.3329, -0.0024,  0.2357],
        [ 0.4099,  0.1306, -0.1127, -0.0377]], requires_grad=True)), ('bias', Pa
tensor([-0.4700,  0.2583, -0.1138,  0.2368, -0.2142], requires_grad=True))])
Sigmoid()
OrderedDict()
Linear(in_features=5, out_features=5, bias=True)
OrderedDict([('weight', Parameter containing:
tensor([[ 0.2459,  0.4003, -0.2752,  0.0781, -0.4327],
        [-0.2687, -0.1221, -0.0464, -0.2257, -0.2737],
        [ 0.2551, -0.1137,  0.0599, -0.1505,  0.2294],
        [-0.3911, -0.2979, -0.0358, -0.4079, -0.1854],
        [-0.1694, -0.2072, -0.3281, -0.3312,  0.3809]], requires_grad=True)), ('
tensor([-0.2059, -0.4174, -0.1822, -0.1115,  0.3775], requires_grad=True))])
Sigmoid()
OrderedDict()
Linear(in_features=5, out_features=5, bias=True)
OrderedDict([('weight', Parameter containing:
tensor([[-0.3884, -0.3393, -0.0121, -0.2346,  0.2158],
        [ 0.2595, -0.1074,  0.4093,  0.3875, -0.2710],
        [ 0.2406, -0.1602, -0.3381, -0.4144,  0.2412],
        [ 0.0811, -0.1548,  0.0645, -0.2701, -0.0554],
        [ 0.1744,  0.3234,  0.1110,  0.2217, -0.0673]], requires_grad=True)), ('
tensor([ 0.4363, -0.3330, -0.2262, -0.0358, -0.4128], requires_grad=True))])
Sigmoid()
OrderedDict()
Linear(in_features=5, out_features=1, bias=True)
OrderedDict([('weight', Parameter containing:
tensor([[-0.2442,  0.1191,  0.3998, -0.4047,  0.4461]], requires_grad=True)), ('
tensor([0.3991], requires_grad=True))])
```

```
criterion = torch.nn.MSELoss() # Regression loss
optimizer = torch.optim.SGD(exam.parameters(), lr=3e-4) # Gradient descent optimizer

# Batches = 50
for e in range(50):
  y_pred = exam(X_train)
  loss = criterion(y_pred, y_train)
  print(f'{e}: mse = {loss.item()}')
  optimizer.zero_grad()
  loss.backward()
```

```
    loss.backward()
    optimizer.step()
```

```
0: mse = 1.128915548324585
1: mse = 1.1281629800796509
2: mse = 1.1274125576019287
3: mse = 1.126664638519287
4: mse = 1.1259182691574097
5: mse = 1.125173807144165
6: mse = 1.1244314908981323
7: mse = 1.1236910820007324
8: mse = 1.1229528188705444
9: mse = 1.1222162246704102
10: mse = 1.1214816570281982
11: mse = 1.1207488775253296
12: mse = 1.1200183629989624
13: mse = 1.1192896366119385
14: mse = 1.118562936782837
15: mse = 1.1178377866744995
16: mse = 1.1171150207519531
17: mse = 1.116393804550171
18: mse = 1.1156747341156006
19: mse = 1.114957571029663
20: mse = 1.1142420768737793
21: mse = 1.1135286092758179
22: mse = 1.1128170490264893
23: mse = 1.1121076345443726
24: mse = 1.111399531364441
25: mse = 1.1106936931610107
26: mse = 1.1099895238876343
27: mse = 1.1092873811721802
28: mse = 1.1085870265960693
29: mse = 1.1078885793685913
30: mse = 1.1071919202804565
31: mse = 1.1064969301223755
32: mse = 1.1058039665222168
33: mse = 1.1051126718521118
34: mse = 1.1044236421585083
35: mse = 1.1037359237670898
36: mse = 1.1030503511428833
37: mse = 1.102366328239441
38: mse = 1.101684331893921
39: mse = 1.1010040044784546
40: mse = 1.1003254652023315
41: mse = 1.0996488332748413
42: mse = 1.0989737510681152
43: mse = 1.0983009338378906
44: mse = 1.0976296663284302
45: mse = 1.0969599485397339
46: mse = 1.0962918996810913
47: mse = 1.0956259965896606
48: mse = 1.0949617624282837
49: mse = 1.0942994356155396
```

```
output = exam(X_train)
output.mean(), output.std()
```

```
(tensor(0.5410, grad_fn=<MeanBackward0>),
 tensor(0.0004, grad_fn=<StdBackward0>))
```

```
output
```

```
tensor([[0.5415],
        [0.5408],
        [0.5417],
        [0.5408],
        [0.5419],
        [0.5408],
        [0.5408],
        [0.5408],
        [0.5415],
        [0.5411],
        [0.5416],
        [0.5407],
        [0.5420],
        [0.5420],
        [0.5418],
        [0.5409],
        [0.5409],
        [0.5416],
        [0.5409],
        [0.5409],
        [0.5418],
        [0.5409],
        [0.5408],
        [0.5407],
        [0.5408],
        [0.5408],
        [0.5408],
        [0.5410],
        [0.5407],
        [0.5408],
        [0.5408],
        [0.5408],
        [0.5412],
        [0.5408],
        [0.5408],
        [0.5407],
        [0.5420],
        [0.5408],
        [0.5412],
        [0.5413],
        [0.5417],
        [0.5408],
        [0.5408],
        [0.5409],
        [0.5411],
        [0.5407],
        [0.5418],
        [0.5408],
        [0.5409],
        [0.5407],
        [0.5415],
        [0.5407],
        [0.5414],
        [0.5407]
```

```
[0.5407],
[0.5407],
[0.5409],
[0.5409],
[0.5419],
[0.5407],
[0.5407],
[0.5407],
[0.5421],
[0.5408],
[0.5419],
[0.5407],
[0.5408],
[0.5408],
[0.5412],
[0.5408],
[0.5408],
[0.5408],
[0.5408],
[0.5408],
[0.5409],
[0.5408],
[0.5409],
[0.5420],
[0.5413],
[0.5408],
[0.5408],
[0.5408],
[0.5408],
[0.5407],
[0.5407],
[0.5408],
[0.5408],
[0.5407],
[0.5408],
[0.5409],
[0.5409],
[0.5408],
[0.5418],
[0.5407],
[0.5412],
[0.5409],
[0.5407],
[0.5409],
[0.5408],
[0.5411],
[0.5414],
[0.5408],
[0.5414],
[0.5407],
[0.5408],
[0.5410],
[0.5421],
[0.5415],
```

```
[0.5407],
[0.5410],
[0.5408],
[0.5408],
[0.5408],
[0.5407],
[0.5409],
[0.5408],
[0.5408],
[0.5408],
[0.5407],
[0.5409],
[0.5412],
[0.5408],
[0.5418],
[0.5421],
[0.5408],
[0.5420],
[0.5420],
[0.5411],
[0.5408],
[0.5409],
[0.5408],
[0.5408],
[0.5409],
[0.5408],
[0.5408],
[0.5408],
[0.5408],
[0.5408],
[0.5412],
[0.5408],
[0.5421],
[0.5408],
[0.5408],
[0.5409],
[0.5408],
[0.5407],
[0.5409],
[0.5413],
[0.5411],
[0.5409],
[0.5408],
[0.5415],
[0.5408],
[0.5408],
[0.5414],
[0.5420],
[0.5408],
[0.5408],
[0.5409],
[0.5408],
[0.5411],
[0.5408],
```