Final-Exam

December 8, 2021

Before you turn this problem in, make sure everything runs as expected. First, **restart the kernel** (in the menubar, select Kernel→Restart) and then **run all cells** (in the menubar, select Cell→Run All).

Make sure you fill in any place that says `YOUR CODE HERE` or "YOUR ANSWER HERE", as well as your name and collaborators below:

[1]:
```
NAME = "Win Win Phyo"
ID = "st122314"
```

---

1 AT82.03 Machine Learning Aug 2021: Final Examination

Happy Wednesday! This is the midterm for Machine Learning in the August 2021 semester.

This exam is 2.5 hours long. Once the exam starts, you will have exactly 2.5 hours to finish your work and upload your notebook to Google Classroom.

Please fill in this notebook with your code and short answers. Be sure to put all of your code in the cells marked with

```
# YOUR CODE GOES HERE
```

and please put your answers to the short answer questions exactly where you see the remark

You answer goes here.

Be complete and precise in your answers! Be sure to answer the question that's being asked. Don't dump random information in the hope that it'll give you partial credit. I give generous partial credit, but I will deduct points for answers that are not on point.

Also beware that if I discover any cheating, I will give you a 0 for the entire exam, or worse, and you will likely fail the class. Just don't do it!

OK, that's all for the advice. Relax, take a deep breath, and good luck!

1.1 Question 1: Data exploration plots (10 points)

Consider the dataset below consisting of two input variables/featues and a single target variable:

[2]:
```
```

In the cell below, write code to split the data into training and validation sets in an 80%/20% ratio, then make two scatterplots, each showing the training set or validation set, with the two classes in different colors.

[3]:
```
import numpy as np
import random
import matplotlib.pyplot as plt
```

[4]:
```
data_ = np.array(data)
print(data_.shape)

X_data,y_data = data_[:, :2], data_[:,-1:]
print(X_data.shape)
print(y_data.shape)
```

```
(500, 3)
(500, 2)
(500, 1)
```

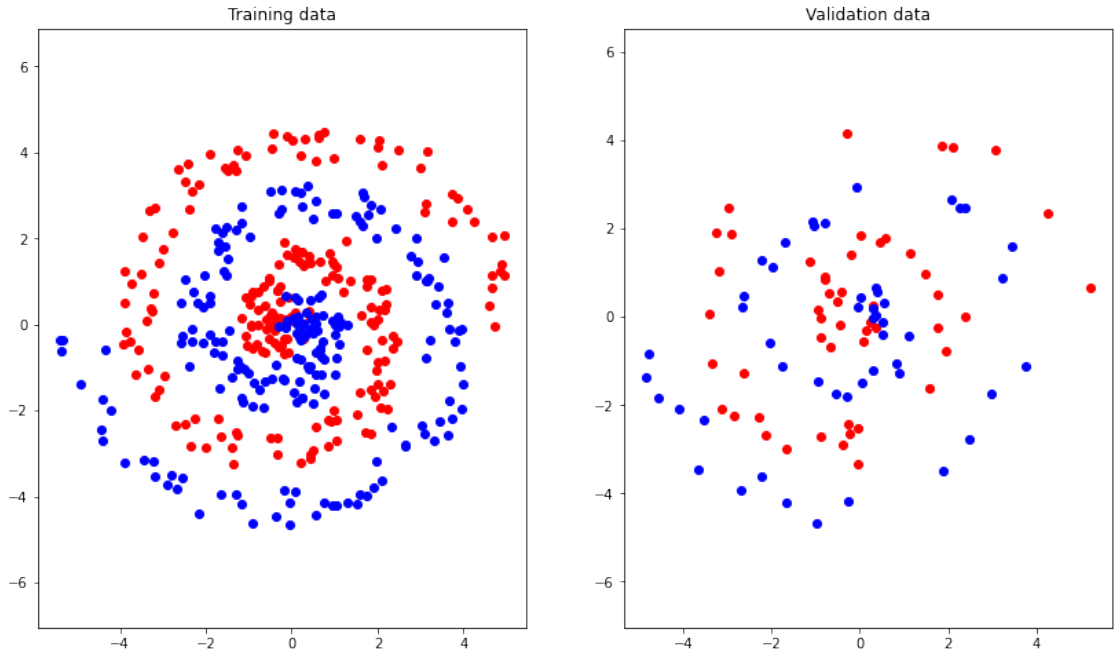
[5]:
```
# YOUR CODE GOES HERE
from sklearn.model_selection import train_test_split

X_train, X_valid, y_train, y_valid = train_test_split(X_data, y_data,␣
 ↪test_size=0.2)
print(X_train.shape)
print(y_train.shape)
print(X_valid.shape)
print(y_valid.shape)
```

```
(400, 2)
(400, 1)
(100, 2)
(100, 1)
```

[6]:
```
def plot_data(X, y, title):
    X_0 = X[(y==-1).flat]
    X_1 = X[(y==1).flat]
    plt.plot(X_0[:,0], X_0[:,1], 'ro')
    plt.plot(X_1[:,0], X_1[:,1], 'bo')
    plt.title(title)
    plt.axis('equal')

plt.rcParams["figure.figsize"] = (14, 8)
plt.subplot(1, 2, 1)
plot_data(X_train, y_train, 'Training data')
plt.subplot(1, 2, 2)
plot_data(X_valid, y_valid, 'Validation data')
```



1.2 Question 2: RBF SVM model (30 points)

Use the RBF SVM code (Gaussian kernel) we developed in lab using cvxopt to find a good model that performs well on the validation set. Visualize the result, showing the validation set with the -1/+1 regions of the input space shown in different colors.

You may normalize the data if you find it necessary.

[7]:
```
# YOUR CODE GOES HERE
def cvxopt_solve_qp(Q, c, A=None, B=None, E=None, d=None):

    Q = (Q + Q.T) / 2
    args = [cvxopt.matrix(Q), cvxopt.matrix(c)]

    if A is not None:
        args.extend([cvxopt.matrix(A), cvxopt.matrix(B)])

        if E is not None:
            args.extend([cvxopt.matrix(E), cvxopt.matrix(d)])

    sol = cvxopt.solvers.qp(*args)
    if sol is not None and 'optimal' not in sol['status']:
        return None
    x = np.array(sol['x']).reshape((Q.shape[1],))
    return x
```

[8]:
```
get_error = False
try:
    Q = np.matrix([[1,0,0], [0,1,0],[0,0,0]])
    c = np.zeros([3])
    m = X_annulus.shape[0]
    A = np.multiply(np.tile(y_annulus,(1,3)), np.concatenate((X_annulus, np.
 ↪ones((m,1))), axis =1))
    B = -np.ones([200])
    x = None
    w = np.matrix([[x[0]],[x[1]]]);
    b = x[2];
    scale = np.linalg.norm(w);                # norm
    x = cvxopt_solve_qp(Q, c, A, B)
    if (x[0] < 0):
        x = -x

    output_str = 'Optimal w: [%f %f] b: %f' % (w[0,0],w[1,0],b)
    get_error = False
except Exception as e:
    output_str = e
    get_error = True
```

[9]:
```
m = X_data.shape[0];
n = X_data.shape[1];

# Transform data set so that each attribute has a
# mean of 0 and a standard deviation of 1

def preprocess(X):
    means = X.mean(0);
    scales = 1/np.std(X,0);
    Xh = np.concatenate([X.T,np.ones([1,20])],0);
    Tm = np.matrix(np.eye(3));
    Tm[0:2,2:3] = -X.mean(0).T;
    Ts = np.matrix(np.eye(3));
    Ts[0:2,0:2] = np.diagflat(scales);
    T = Ts*Tm;
    XX = (T * Xh);
    XX = XX[0:2,:].T;
    return XX, T;

# RBF/Gaussian kernel

def gauss_kernel(X):
    sigma = 0.2
    m = X.shape[0];
    K = np.matrix(np.zeros([m,m]));
```

```
    for i in range(0,m):
        for j in range(0,m):
            K[i,j] = (X[i,:] - X[j,:]).reshape(1,-1) @ (X[i,:] - X[j,:]).
↪reshape(-1,1)
    K = np.exp(-K/(2*sigma*sigma))
    return K;

def linear_kernel(X):
    m = X.shape[0];
    K = np.matrix(np.zeros([m,m]));
    for i in range(0,m):
        for j in range(0,m):
            K[i,j] = (X[i,:].reshape(1,-1)@X[j,:].reshape(-1,1))
    return K;
```

[10]:
```
sigma = 0.2
m = X_data.shape[0]
K_data = np.matrix(np.zeros([m,m]))
# print(K_annulus.shape)
# print(type(K_annulus))

for i in range(0,m):
    for j in range(0,m):
        K_data[i,j] = (X_data[i,:] - X_data[j,:]) @ (X_data[i,:] - X_data[j,:]).
↪T
K_data = np.exp(-K_data/(2*sigma*sigma))


#K_annulus = gauss_kernel(X_annulus)

Q_data = np.multiply(y_data * y_data.T, K_data)
c = -np.ones([m])
A = -np.eye(m)
B = np.zeros([m])
E = y_data.T
d = np.zeros(1)
```

[11]:
```
import cvxopt
alpha_star_data = cvxopt_solve_qp(Q_data, c, A, B, E, d)
```

```
     pcost       dcost       gap    pres   dres
 0: -1.1770e+02 -4.5925e+02  1e+03  2e+01  3e+00
 1: -1.9846e+02 -6.3418e+02  9e+02  1e+01  1e+00
 2: -4.5665e+02 -9.4289e+02  8e+02  1e+01  1e+00
 3: -1.4620e+03 -2.0293e+03  9e+02  9e+00  1e+00
 4: -3.2006e+03 -3.8878e+03  1e+03  9e+00  1e+00
 5: -7.3486e+03 -8.3355e+03  1e+03  9e+00  1e+00
```

```
 6: -1.5210e+04 -1.7175e+04  2e+03  9e+00  1e+00
 7: -2.6556e+04 -3.0716e+04  5e+03  9e+00  1e+00
 8: -6.8494e+04 -9.1586e+04  2e+04  6e+00  7e-01
 9: -8.5665e+04 -9.6411e+04  1e+04  1e+00  2e-01
10: -8.6557e+04 -8.7688e+04  1e+03  1e-01  1e-02
11: -8.6579e+04 -8.6618e+04  4e+01  1e-03  1e-04
12: -8.6585e+04 -8.6590e+04  5e+00  1e-04  2e-05
13: -8.6586e+04 -8.6587e+04  1e+00  2e-05  3e-06
14: -8.6586e+04 -8.6586e+04  1e-01  2e-06  2e-07
15: -8.6586e+04 -8.6586e+04  9e-03  1e-07  1e-08
Optimal solution found.
```

[12]:
```
def predict(x, X, y, alpha):
    s = []
    sigma = 0.2
    for j in range(x.shape[0]):
        ss = 0
        for i in range(X.shape[0]):
            ss += alpha[i]*y[i]*np.exp((-(X[i]-x[j])@(X[i]-x[j]))/
 ↪(2*sigma*sigma))
        s.append(ss)
    s = np.array(s)
    s[s >= 0] = 1
    s[s < 0] = -1
    return s

y_pred = predict(X_data, X_data, y_data, alpha_star_data)
np.sum(y_data == y_pred)/y_data.size
```

[12]:
```
1.0
```

[13]:
```
def plot_annulus(X1, X2):
    ax = plt.axes()
    plt.title('Sample data for classification problem')
    plt.grid(axis='both', alpha=.25)
    plt.plot(X1[:,0],X1[:,1],'b.', label = 'Class 1')
    plt.plot(X2[:,0],X2[:,1],'g*', label = 'Class 2')
    plt.legend(loc=2)
    ax.set_aspect('equal', 'datalim')
    return ax

# fig1 = plt.figure(figsize=(8,8))
# plot_annulus(X_train, X_valid)
# plt.show()
```

[14]:
```
x_series = np.linspace(-15, 15, 100)
y_series = np.linspace(-15, 15, 100)
```

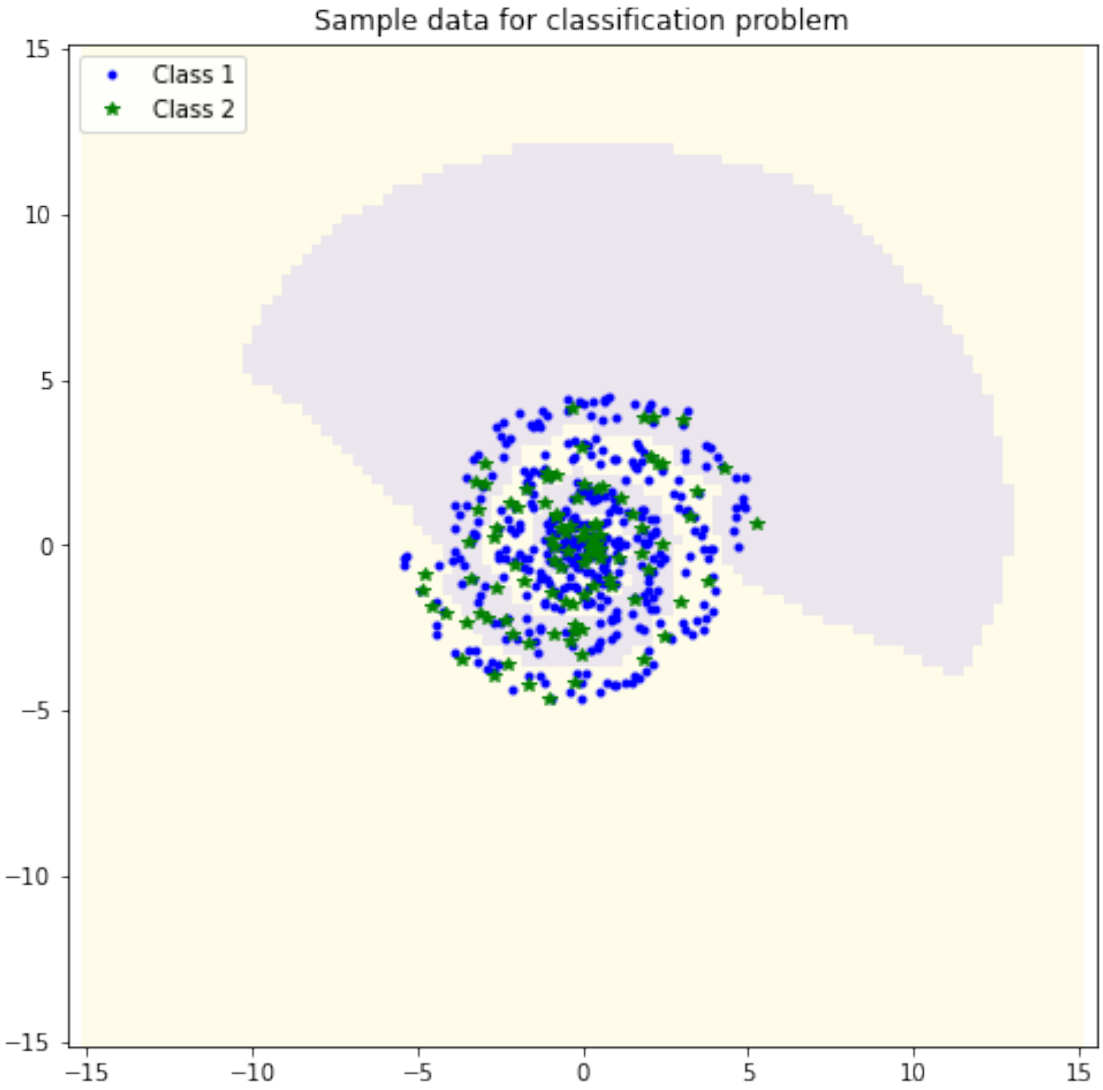
```
x_mesh, y_mesh = np.meshgrid(x_series, y_series)

x_mesh = x_mesh.reshape(-1, 1)
y_mesh = y_mesh.reshape(-1, 1)

mesh = np.append(x_mesh, y_mesh, axis=1)
y_pred = predict(mesh, X_data, y_data, alpha_star_data)

x_mesh = x_mesh.reshape(100, 100)
y_mesh = y_mesh.reshape(100, 100)
y_pred = y_pred.reshape(100, 100)

fig1 = plt.figure(figsize=(8,8))
ax = plot_annulus(X_train, X_valid)
ax.pcolormesh(x_mesh, y_mesh, y_pred, cmap='viridis', shading='auto', alpha=0.1)
plt.show()
```



1.3 Question 3: Neural network model (30 points)

In our deep learning lab, you already found neural networks in the Tensorflow Playground capable of learning this dataset. Using the code developed in lab, find a good multilayer neural network model that performs well on the validation set. Visualize the result, showing the validation set with the -1/+1 regions of the input space shown in different colors.

You may normalize the data if you find it necessary.

[15]:
```
# def normalize(X):
#     M = X.shape[0]
#     XX = X - np.tile(np.mean(X,0),[M,1])
#     XX = np.divide(XX, np.tile(np.std(XX,0),[M,1]))
#     return np.nan_to_num(XX, copy=True,nan=0.0)
```

```
# # XX = normalize(X)
```

[16]:
```
import torch
import torch.cuda as cuda
import torch.nn as nn

from torch.autograd import Variable

from torchvision import datasets
from torchvision import transforms

import torch.nn.functional as F

import torch
import torch.nn as nn
from torch.utils.data import TensorDataset
from torch.utils.data import DataLoader
import torch.nn.functional as F
```

[17]:
```
# X_train = X_train.reshape(X_train.shape[0], 1, 1, 2)
# X_valid = X_valid.reshape(X_valid.shape[0], 1, 1, 2)

# print(X_train.shape)
# print(X_valid.shape)
```

[18]:
```
X_train_ = torch.Tensor(X_train)
X_valid_ = torch.Tensor(X_valid)

y_train_ = torch.Tensor(y_train)
y_valid_ = torch.Tensor(y_valid)

train_ds = TensorDataset(X_train_, y_train_)
valid_ds = TensorDataset(X_valid_, y_valid_)

print(X_train_.size())
print(X_valid_.size())

print(y_train_.size())
print(y_valid_.size())
BATCH_SIZE= 64

train_iterator = torch.utils.data.DataLoader(dataset=train_ds,␣
 ↪batch_size=BATCH_SIZE, shuffle=True)
valid_iterator = torch.utils.data.DataLoader(dataset=valid_ds,␣
 ↪batch_size=BATCH_SIZE, shuffle=False)
```

```
torch.Size([400, 2])
torch.Size([100, 2])
torch.Size([400, 1])
torch.Size([100, 1])
```

[19]:
```
# class Net(nn.Module):
#     def __init__(self, num_class, num_layers, num_classes):
#         super(Net, self).__init__()

#         self.num_classes = num_classes
#         self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
#         self.conv1_drop = nn.Dropout2d(p = 0.5)
#         self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
#         self.conv2_drop = nn.Dropout2d(p = 0.5)
#         self.fc1 = nn.Linear(320, 25)


#         self.hidden_size = num_classes
#         self.num_layers = num_layers
#         self.input_size = 5
#         self.sequence_length = 5

#         self.lstm = nn.LSTM(self.input_size, self.hidden_size, self.
 ↪num_layers, batch_first=True)

#     def forward(self, x):
#         x = F.relu(F.max_pool2d(self.conv1_drop(self.conv1(x)), 2))
#         x = F.relu(F.max_pool2d(self.conv2_drop(self.conv2(x)), 2))
#         x = x.view(-1, 320)
#         x = F.relu(self.fc1(x))
#         x = F.dropout(x, training=self.training)
# #         # out = x
# #         h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size)
# #         c0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size)

#         x = x.reshape(-1, self.sequence_length, self.input_size).to(device)
#         out, hidden = self.lstm(x, (h0, c0))  # out: tensor of shape␣
 ↪(batch_size, seq_length, hidden_size)
#         out = out[:, -1, :]
#         return out
```

[20]:
```
input_size = X_data.shape[1]
output_size = 1
hidden_size = 5
#hidden1_size = None
#hidden2_size = None
```

```
#hidden3_size = None
#hidden4_size = None

epochs = 500
batch_size = 50
learning_rate = 0.01
```

[21]:
```
class Network(nn.Module):

    def __init__(self):
        super(Network, self).__init__()
        self.l1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.l3 = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        x = self.l1(x)
        x = self.relu(x)
        x = self.l3(x)
        return F.log_softmax(x)
```

[22]:
```
# class Network(nn.Module):
#     def __init__(self, num_classes=10):
#         super(Network, self).__init__()

#         #using sequential helps bind multiple operations together
#         self.layer1 = nn.Sequential(
#             #in_channel = 1
#             #out_channel = 16
#             #padding = (kernel_size - 1) / 2 = 2
#             nn.Conv2d(1, 16, kernel_size=5, stride=1, padding=2),
#             nn.BatchNorm2d(16),
#             nn.ReLU(),
#             nn.MaxPool2d(kernel_size=2, stride=2))
#         #after layer 1 will be of shape [100, 16, 14, 14]
#         self.layer2 = nn.Sequential(
#             nn.Conv2d(16, 32, kernel_size=5, stride=1, padding=2),
#             nn.BatchNorm2d(32),
#             nn.ReLU(),
#             nn.MaxPool2d(kernel_size=2, stride=2))
#         #after layer 2 will be of shape [100, 32, 7, 7]
#         self.fc = nn.Linear(32*7*7, num_classes)
#         self.drop_out = nn.Dropout(p=0.2)  #zeroed 0.2% data
#         #after fc will be of shape [100, 10]

#     def forward(self, x):
#         #x shape: [batch, in_channel, img_width, img_height]
```

```
#         #[100, 1, 28, 28]
#         out = self.layer1(x)
#         out = self.drop_out(out)
#         #after layer 1: shape: [100, 16, 14, 14]
#         out = self.layer2(out)
#         out = self.drop_out(out)
#         #after layer 2: shape: [100, 32, 7, 7]
#         out = out.reshape(out.size(0), -1)   #can also use .view()
#         #after squeezing: shape: [100, 1568]
#         #we squeeze so that it can be inputted into the fc layer
#         out = self.fc(out)
#         #after fc layer: shape: [100, 10]
#         return out
```

[23]:
```
net = Network()
print(net)
```

```
Network(
  (l1): Linear(in_features=2, out_features=5, bias=True)
  (relu): ReLU()
  (l3): Linear(in_features=5, out_features=1, bias=True)
)
```

[24]:
```
optimizer = torch.optim.SGD(net.parameters(), lr=learning_rate, momentum=0.9)
loss_func = nn.CrossEntropyLoss()
```

[25]:
```
loss_log = []

for e in range(epochs):
    for i in range(0, X_train_.shape[0], batch_size):
        x_mini = X_train_[i:i + batch_size]
        y_mini = y_train_[i:i + batch_size]

        x_var = Variable(x_mini)
        y_var = Variable(y_mini)

        #print(y_var)
        optimizer.zero_grad()
        net_out = net(x_var)

        loss = loss_func(net_out, y_var)
        loss.backward()
        optimizer.step()

        if i % 100 == 0:
            loss_log.append(loss.item())
```

```
    print('Epoch: {} - Loss: {:.6f}'.format(e, loss.item()))
```

```
Epoch: 0 - Loss: -0.000000
Epoch: 1 - Loss: -0.000000
Epoch: 2 - Loss: -0.000000
Epoch: 3 - Loss: -0.000000
Epoch: 4 - Loss: -0.000000
Epoch: 5 - Loss: -0.000000
Epoch: 6 - Loss: -0.000000
Epoch: 7 - Loss: -0.000000
Epoch: 8 - Loss: -0.000000
Epoch: 9 - Loss: -0.000000
Epoch: 10 - Loss: -0.000000
Epoch: 11 - Loss: -0.000000
Epoch: 12 - Loss: -0.000000
Epoch: 13 - Loss: -0.000000
Epoch: 14 - Loss: -0.000000
Epoch: 15 - Loss: -0.000000
Epoch: 16 - Loss: -0.000000
Epoch: 17 - Loss: -0.000000
Epoch: 18 - Loss: -0.000000
Epoch: 19 - Loss: -0.000000
Epoch: 20 - Loss: -0.000000
Epoch: 21 - Loss: -0.000000
```

```
/tmp/ipykernel_486/3446910693.py:13: UserWarning: Implicit dimension choice for
log_softmax has been deprecated. Change the call to include dim=X as an
argument.
  return F.log_softmax(x)
```

```
Epoch: 22 - Loss: -0.000000
Epoch: 23 - Loss: -0.000000
Epoch: 24 - Loss: -0.000000
Epoch: 25 - Loss: -0.000000
Epoch: 26 - Loss: -0.000000
Epoch: 27 - Loss: -0.000000
Epoch: 28 - Loss: -0.000000
Epoch: 29 - Loss: -0.000000
Epoch: 30 - Loss: -0.000000
Epoch: 31 - Loss: -0.000000
Epoch: 32 - Loss: -0.000000
Epoch: 33 - Loss: -0.000000
Epoch: 34 - Loss: -0.000000
Epoch: 35 - Loss: -0.000000
Epoch: 36 - Loss: -0.000000
Epoch: 37 - Loss: -0.000000
Epoch: 38 - Loss: -0.000000
Epoch: 39 - Loss: -0.000000
```

```
Epoch: 40 - Loss: -0.000000
Epoch: 41 - Loss: -0.000000
Epoch: 42 - Loss: -0.000000
Epoch: 43 - Loss: -0.000000
Epoch: 44 - Loss: -0.000000
Epoch: 45 - Loss: -0.000000
Epoch: 46 - Loss: -0.000000
Epoch: 47 - Loss: -0.000000
Epoch: 48 - Loss: -0.000000
Epoch: 49 - Loss: -0.000000
Epoch: 50 - Loss: -0.000000
Epoch: 51 - Loss: -0.000000
Epoch: 52 - Loss: -0.000000
Epoch: 53 - Loss: -0.000000
Epoch: 54 - Loss: -0.000000
Epoch: 55 - Loss: -0.000000
Epoch: 56 - Loss: -0.000000
Epoch: 57 - Loss: -0.000000
Epoch: 58 - Loss: -0.000000
Epoch: 59 - Loss: -0.000000
Epoch: 60 - Loss: -0.000000
Epoch: 61 - Loss: -0.000000
Epoch: 62 - Loss: -0.000000
Epoch: 63 - Loss: -0.000000
Epoch: 64 - Loss: -0.000000
Epoch: 65 - Loss: -0.000000
Epoch: 66 - Loss: -0.000000
Epoch: 67 - Loss: -0.000000
Epoch: 68 - Loss: -0.000000
Epoch: 69 - Loss: -0.000000
Epoch: 70 - Loss: -0.000000
Epoch: 71 - Loss: -0.000000
Epoch: 72 - Loss: -0.000000
Epoch: 73 - Loss: -0.000000
Epoch: 74 - Loss: -0.000000
Epoch: 75 - Loss: -0.000000
Epoch: 76 - Loss: -0.000000
Epoch: 77 - Loss: -0.000000
Epoch: 78 - Loss: -0.000000
Epoch: 79 - Loss: -0.000000
Epoch: 80 - Loss: -0.000000
Epoch: 81 - Loss: -0.000000
Epoch: 82 - Loss: -0.000000
Epoch: 83 - Loss: -0.000000
Epoch: 84 - Loss: -0.000000
Epoch: 85 - Loss: -0.000000
Epoch: 86 - Loss: -0.000000
Epoch: 87 - Loss: -0.000000
```

```
Epoch: 88 - Loss: -0.000000
Epoch: 89 - Loss: -0.000000
Epoch: 90 - Loss: -0.000000
Epoch: 91 - Loss: -0.000000
Epoch: 92 - Loss: -0.000000
Epoch: 93 - Loss: -0.000000
Epoch: 94 - Loss: -0.000000
Epoch: 95 - Loss: -0.000000
Epoch: 96 - Loss: -0.000000
Epoch: 97 - Loss: -0.000000
Epoch: 98 - Loss: -0.000000
Epoch: 99 - Loss: -0.000000
Epoch: 100 - Loss: -0.000000
Epoch: 101 - Loss: -0.000000
Epoch: 102 - Loss: -0.000000
Epoch: 103 - Loss: -0.000000
Epoch: 104 - Loss: -0.000000
Epoch: 105 - Loss: -0.000000
Epoch: 106 - Loss: -0.000000
Epoch: 107 - Loss: -0.000000
Epoch: 108 - Loss: -0.000000
Epoch: 109 - Loss: -0.000000
Epoch: 110 - Loss: -0.000000
Epoch: 111 - Loss: -0.000000
Epoch: 112 - Loss: -0.000000
Epoch: 113 - Loss: -0.000000
Epoch: 114 - Loss: -0.000000
Epoch: 115 - Loss: -0.000000
Epoch: 116 - Loss: -0.000000
Epoch: 117 - Loss: -0.000000
Epoch: 118 - Loss: -0.000000
Epoch: 119 - Loss: -0.000000
Epoch: 120 - Loss: -0.000000
Epoch: 121 - Loss: -0.000000
Epoch: 122 - Loss: -0.000000
Epoch: 123 - Loss: -0.000000
Epoch: 124 - Loss: -0.000000
Epoch: 125 - Loss: -0.000000
Epoch: 126 - Loss: -0.000000
Epoch: 127 - Loss: -0.000000
Epoch: 128 - Loss: -0.000000
Epoch: 129 - Loss: -0.000000
Epoch: 130 - Loss: -0.000000
Epoch: 131 - Loss: -0.000000
Epoch: 132 - Loss: -0.000000
Epoch: 133 - Loss: -0.000000
Epoch: 134 - Loss: -0.000000
Epoch: 135 - Loss: -0.000000
```

```
Epoch: 136 - Loss: -0.000000
Epoch: 137 - Loss: -0.000000
Epoch: 138 - Loss: -0.000000
Epoch: 139 - Loss: -0.000000
Epoch: 140 - Loss: -0.000000
Epoch: 141 - Loss: -0.000000
Epoch: 142 - Loss: -0.000000
Epoch: 143 - Loss: -0.000000
Epoch: 144 - Loss: -0.000000
Epoch: 145 - Loss: -0.000000
Epoch: 146 - Loss: -0.000000
Epoch: 147 - Loss: -0.000000
Epoch: 148 - Loss: -0.000000
Epoch: 149 - Loss: -0.000000
Epoch: 150 - Loss: -0.000000
Epoch: 151 - Loss: -0.000000
Epoch: 152 - Loss: -0.000000
Epoch: 153 - Loss: -0.000000
Epoch: 154 - Loss: -0.000000
Epoch: 155 - Loss: -0.000000
Epoch: 156 - Loss: -0.000000
Epoch: 157 - Loss: -0.000000
Epoch: 158 - Loss: -0.000000
Epoch: 159 - Loss: -0.000000
Epoch: 160 - Loss: -0.000000
Epoch: 161 - Loss: -0.000000
Epoch: 162 - Loss: -0.000000
Epoch: 163 - Loss: -0.000000
Epoch: 164 - Loss: -0.000000
Epoch: 165 - Loss: -0.000000
Epoch: 166 - Loss: -0.000000
Epoch: 167 - Loss: -0.000000
Epoch: 168 - Loss: -0.000000
Epoch: 169 - Loss: -0.000000
Epoch: 170 - Loss: -0.000000
Epoch: 171 - Loss: -0.000000
Epoch: 172 - Loss: -0.000000
Epoch: 173 - Loss: -0.000000
Epoch: 174 - Loss: -0.000000
Epoch: 175 - Loss: -0.000000
Epoch: 176 - Loss: -0.000000
Epoch: 177 - Loss: -0.000000
Epoch: 178 - Loss: -0.000000
Epoch: 179 - Loss: -0.000000
Epoch: 180 - Loss: -0.000000
Epoch: 181 - Loss: -0.000000
Epoch: 182 - Loss: -0.000000
Epoch: 183 - Loss: -0.000000
```

```
Epoch: 184 - Loss: -0.000000
Epoch: 185 - Loss: -0.000000
Epoch: 186 - Loss: -0.000000
Epoch: 187 - Loss: -0.000000
Epoch: 188 - Loss: -0.000000
Epoch: 189 - Loss: -0.000000
Epoch: 190 - Loss: -0.000000
Epoch: 191 - Loss: -0.000000
Epoch: 192 - Loss: -0.000000
Epoch: 193 - Loss: -0.000000
Epoch: 194 - Loss: -0.000000
Epoch: 195 - Loss: -0.000000
Epoch: 196 - Loss: -0.000000
Epoch: 197 - Loss: -0.000000
Epoch: 198 - Loss: -0.000000
Epoch: 199 - Loss: -0.000000
Epoch: 200 - Loss: -0.000000
Epoch: 201 - Loss: -0.000000
Epoch: 202 - Loss: -0.000000
Epoch: 203 - Loss: -0.000000
Epoch: 204 - Loss: -0.000000
Epoch: 205 - Loss: -0.000000
Epoch: 206 - Loss: -0.000000
Epoch: 207 - Loss: -0.000000
Epoch: 208 - Loss: -0.000000
Epoch: 209 - Loss: -0.000000
Epoch: 210 - Loss: -0.000000
Epoch: 211 - Loss: -0.000000
Epoch: 212 - Loss: -0.000000
Epoch: 213 - Loss: -0.000000
Epoch: 214 - Loss: -0.000000
Epoch: 215 - Loss: -0.000000
Epoch: 216 - Loss: -0.000000
Epoch: 217 - Loss: -0.000000
Epoch: 218 - Loss: -0.000000
Epoch: 219 - Loss: -0.000000
Epoch: 220 - Loss: -0.000000
Epoch: 221 - Loss: -0.000000
Epoch: 222 - Loss: -0.000000
Epoch: 223 - Loss: -0.000000
Epoch: 224 - Loss: -0.000000
Epoch: 225 - Loss: -0.000000
Epoch: 226 - Loss: -0.000000
Epoch: 227 - Loss: -0.000000
Epoch: 228 - Loss: -0.000000
Epoch: 229 - Loss: -0.000000
Epoch: 230 - Loss: -0.000000
Epoch: 231 - Loss: -0.000000
```

```
Epoch: 232 - Loss: -0.000000
Epoch: 233 - Loss: -0.000000
Epoch: 234 - Loss: -0.000000
Epoch: 235 - Loss: -0.000000
Epoch: 236 - Loss: -0.000000
Epoch: 237 - Loss: -0.000000
Epoch: 238 - Loss: -0.000000
Epoch: 239 - Loss: -0.000000
Epoch: 240 - Loss: -0.000000
Epoch: 241 - Loss: -0.000000
Epoch: 242 - Loss: -0.000000
Epoch: 243 - Loss: -0.000000
Epoch: 244 - Loss: -0.000000
Epoch: 245 - Loss: -0.000000
Epoch: 246 - Loss: -0.000000
Epoch: 247 - Loss: -0.000000
Epoch: 248 - Loss: -0.000000
Epoch: 249 - Loss: -0.000000
Epoch: 250 - Loss: -0.000000
Epoch: 251 - Loss: -0.000000
Epoch: 252 - Loss: -0.000000
Epoch: 253 - Loss: -0.000000
Epoch: 254 - Loss: -0.000000
Epoch: 255 - Loss: -0.000000
Epoch: 256 - Loss: -0.000000
Epoch: 257 - Loss: -0.000000
Epoch: 258 - Loss: -0.000000
Epoch: 259 - Loss: -0.000000
Epoch: 260 - Loss: -0.000000
Epoch: 261 - Loss: -0.000000
Epoch: 262 - Loss: -0.000000
Epoch: 263 - Loss: -0.000000
Epoch: 264 - Loss: -0.000000
Epoch: 265 - Loss: -0.000000
Epoch: 266 - Loss: -0.000000
Epoch: 267 - Loss: -0.000000
Epoch: 268 - Loss: -0.000000
Epoch: 269 - Loss: -0.000000
Epoch: 270 - Loss: -0.000000
Epoch: 271 - Loss: -0.000000
Epoch: 272 - Loss: -0.000000
Epoch: 273 - Loss: -0.000000
Epoch: 274 - Loss: -0.000000
Epoch: 275 - Loss: -0.000000
Epoch: 276 - Loss: -0.000000
Epoch: 277 - Loss: -0.000000
Epoch: 278 - Loss: -0.000000
Epoch: 279 - Loss: -0.000000
```

```
Epoch: 280 - Loss: -0.000000
Epoch: 281 - Loss: -0.000000
Epoch: 282 - Loss: -0.000000
Epoch: 283 - Loss: -0.000000
Epoch: 284 - Loss: -0.000000
Epoch: 285 - Loss: -0.000000
Epoch: 286 - Loss: -0.000000
Epoch: 287 - Loss: -0.000000
Epoch: 288 - Loss: -0.000000
Epoch: 289 - Loss: -0.000000
Epoch: 290 - Loss: -0.000000
Epoch: 291 - Loss: -0.000000
Epoch: 292 - Loss: -0.000000
Epoch: 293 - Loss: -0.000000
Epoch: 294 - Loss: -0.000000
Epoch: 295 - Loss: -0.000000
Epoch: 296 - Loss: -0.000000
Epoch: 297 - Loss: -0.000000
Epoch: 298 - Loss: -0.000000
Epoch: 299 - Loss: -0.000000
Epoch: 300 - Loss: -0.000000
Epoch: 301 - Loss: -0.000000
Epoch: 302 - Loss: -0.000000
Epoch: 303 - Loss: -0.000000
Epoch: 304 - Loss: -0.000000
Epoch: 305 - Loss: -0.000000
Epoch: 306 - Loss: -0.000000
Epoch: 307 - Loss: -0.000000
Epoch: 308 - Loss: -0.000000
Epoch: 309 - Loss: -0.000000
Epoch: 310 - Loss: -0.000000
Epoch: 311 - Loss: -0.000000
Epoch: 312 - Loss: -0.000000
Epoch: 313 - Loss: -0.000000
Epoch: 314 - Loss: -0.000000
Epoch: 315 - Loss: -0.000000
Epoch: 316 - Loss: -0.000000
Epoch: 317 - Loss: -0.000000
Epoch: 318 - Loss: -0.000000
Epoch: 319 - Loss: -0.000000
Epoch: 320 - Loss: -0.000000
Epoch: 321 - Loss: -0.000000
Epoch: 322 - Loss: -0.000000
Epoch: 323 - Loss: -0.000000
Epoch: 324 - Loss: -0.000000
Epoch: 325 - Loss: -0.000000
Epoch: 326 - Loss: -0.000000
Epoch: 327 - Loss: -0.000000
```

```
Epoch: 328 - Loss: -0.000000
Epoch: 329 - Loss: -0.000000
Epoch: 330 - Loss: -0.000000
Epoch: 331 - Loss: -0.000000
Epoch: 332 - Loss: -0.000000
Epoch: 333 - Loss: -0.000000
Epoch: 334 - Loss: -0.000000
Epoch: 335 - Loss: -0.000000
Epoch: 336 - Loss: -0.000000
Epoch: 337 - Loss: -0.000000
Epoch: 338 - Loss: -0.000000
Epoch: 339 - Loss: -0.000000
Epoch: 340 - Loss: -0.000000
Epoch: 341 - Loss: -0.000000
Epoch: 342 - Loss: -0.000000
Epoch: 343 - Loss: -0.000000
Epoch: 344 - Loss: -0.000000
Epoch: 345 - Loss: -0.000000
Epoch: 346 - Loss: -0.000000
Epoch: 347 - Loss: -0.000000
Epoch: 348 - Loss: -0.000000
Epoch: 349 - Loss: -0.000000
Epoch: 350 - Loss: -0.000000
Epoch: 351 - Loss: -0.000000
Epoch: 352 - Loss: -0.000000
Epoch: 353 - Loss: -0.000000
Epoch: 354 - Loss: -0.000000
Epoch: 355 - Loss: -0.000000
Epoch: 356 - Loss: -0.000000
Epoch: 357 - Loss: -0.000000
Epoch: 358 - Loss: -0.000000
Epoch: 359 - Loss: -0.000000
Epoch: 360 - Loss: -0.000000
Epoch: 361 - Loss: -0.000000
Epoch: 362 - Loss: -0.000000
Epoch: 363 - Loss: -0.000000
Epoch: 364 - Loss: -0.000000
Epoch: 365 - Loss: -0.000000
Epoch: 366 - Loss: -0.000000
Epoch: 367 - Loss: -0.000000
Epoch: 368 - Loss: -0.000000
Epoch: 369 - Loss: -0.000000
Epoch: 370 - Loss: -0.000000
Epoch: 371 - Loss: -0.000000
Epoch: 372 - Loss: -0.000000
Epoch: 373 - Loss: -0.000000
Epoch: 374 - Loss: -0.000000
Epoch: 375 - Loss: -0.000000
```

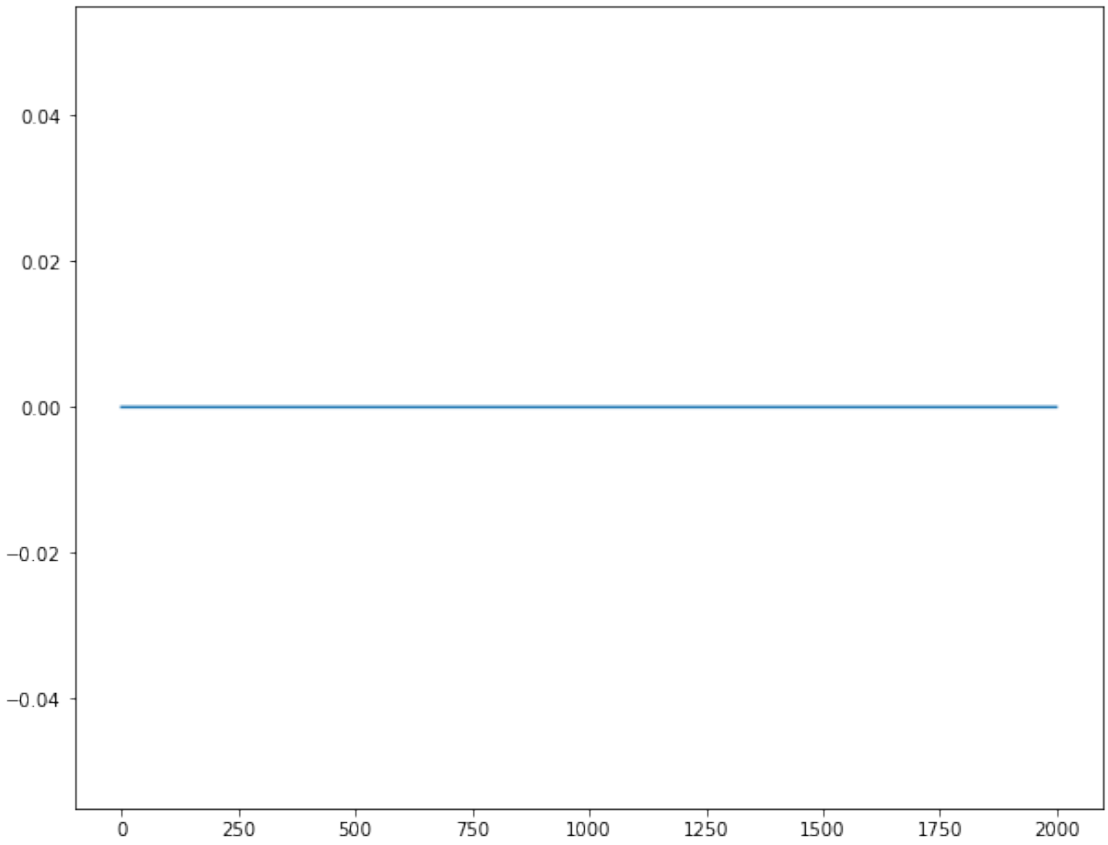
```
Epoch: 376 - Loss: -0.000000
Epoch: 377 - Loss: -0.000000
Epoch: 378 - Loss: -0.000000
Epoch: 379 - Loss: -0.000000
Epoch: 380 - Loss: -0.000000
Epoch: 381 - Loss: -0.000000
Epoch: 382 - Loss: -0.000000
Epoch: 383 - Loss: -0.000000
Epoch: 384 - Loss: -0.000000
Epoch: 385 - Loss: -0.000000
Epoch: 386 - Loss: -0.000000
Epoch: 387 - Loss: -0.000000
Epoch: 388 - Loss: -0.000000
Epoch: 389 - Loss: -0.000000
Epoch: 390 - Loss: -0.000000
Epoch: 391 - Loss: -0.000000
Epoch: 392 - Loss: -0.000000
Epoch: 393 - Loss: -0.000000
Epoch: 394 - Loss: -0.000000
Epoch: 395 - Loss: -0.000000
Epoch: 396 - Loss: -0.000000
Epoch: 397 - Loss: -0.000000
Epoch: 398 - Loss: -0.000000
Epoch: 399 - Loss: -0.000000
Epoch: 400 - Loss: -0.000000
Epoch: 401 - Loss: -0.000000
Epoch: 402 - Loss: -0.000000
Epoch: 403 - Loss: -0.000000
Epoch: 404 - Loss: -0.000000
Epoch: 405 - Loss: -0.000000
Epoch: 406 - Loss: -0.000000
Epoch: 407 - Loss: -0.000000
Epoch: 408 - Loss: -0.000000
Epoch: 409 - Loss: -0.000000
Epoch: 410 - Loss: -0.000000
Epoch: 411 - Loss: -0.000000
Epoch: 412 - Loss: -0.000000
Epoch: 413 - Loss: -0.000000
Epoch: 414 - Loss: -0.000000
Epoch: 415 - Loss: -0.000000
Epoch: 416 - Loss: -0.000000
Epoch: 417 - Loss: -0.000000
Epoch: 418 - Loss: -0.000000
Epoch: 419 - Loss: -0.000000
Epoch: 420 - Loss: -0.000000
Epoch: 421 - Loss: -0.000000
Epoch: 422 - Loss: -0.000000
Epoch: 423 - Loss: -0.000000
```

```
Epoch: 424 - Loss: -0.000000
Epoch: 425 - Loss: -0.000000
Epoch: 426 - Loss: -0.000000
Epoch: 427 - Loss: -0.000000
Epoch: 428 - Loss: -0.000000
Epoch: 429 - Loss: -0.000000
Epoch: 430 - Loss: -0.000000
Epoch: 431 - Loss: -0.000000
Epoch: 432 - Loss: -0.000000
Epoch: 433 - Loss: -0.000000
Epoch: 434 - Loss: -0.000000
Epoch: 435 - Loss: -0.000000
Epoch: 436 - Loss: -0.000000
Epoch: 437 - Loss: -0.000000
Epoch: 438 - Loss: -0.000000
Epoch: 439 - Loss: -0.000000
Epoch: 440 - Loss: -0.000000
Epoch: 441 - Loss: -0.000000
Epoch: 442 - Loss: -0.000000
Epoch: 443 - Loss: -0.000000
Epoch: 444 - Loss: -0.000000
Epoch: 445 - Loss: -0.000000
Epoch: 446 - Loss: -0.000000
Epoch: 447 - Loss: -0.000000
Epoch: 448 - Loss: -0.000000
Epoch: 449 - Loss: -0.000000
Epoch: 450 - Loss: -0.000000
Epoch: 451 - Loss: -0.000000
Epoch: 452 - Loss: -0.000000
Epoch: 453 - Loss: -0.000000
Epoch: 454 - Loss: -0.000000
Epoch: 455 - Loss: -0.000000
Epoch: 456 - Loss: -0.000000
Epoch: 457 - Loss: -0.000000
Epoch: 458 - Loss: -0.000000
Epoch: 459 - Loss: -0.000000
Epoch: 460 - Loss: -0.000000
Epoch: 461 - Loss: -0.000000
Epoch: 462 - Loss: -0.000000
Epoch: 463 - Loss: -0.000000
Epoch: 464 - Loss: -0.000000
Epoch: 465 - Loss: -0.000000
Epoch: 466 - Loss: -0.000000
Epoch: 467 - Loss: -0.000000
Epoch: 468 - Loss: -0.000000
Epoch: 469 - Loss: -0.000000
Epoch: 470 - Loss: -0.000000
Epoch: 471 - Loss: -0.000000
```

```
Epoch: 472 - Loss: -0.000000
Epoch: 473 - Loss: -0.000000
Epoch: 474 - Loss: -0.000000
Epoch: 475 - Loss: -0.000000
Epoch: 476 - Loss: -0.000000
Epoch: 477 - Loss: -0.000000
Epoch: 478 - Loss: -0.000000
Epoch: 479 - Loss: -0.000000
Epoch: 480 - Loss: -0.000000
Epoch: 481 - Loss: -0.000000
Epoch: 482 - Loss: -0.000000
Epoch: 483 - Loss: -0.000000
Epoch: 484 - Loss: -0.000000
Epoch: 485 - Loss: -0.000000
Epoch: 486 - Loss: -0.000000
Epoch: 487 - Loss: -0.000000
Epoch: 488 - Loss: -0.000000
Epoch: 489 - Loss: -0.000000
Epoch: 490 - Loss: -0.000000
Epoch: 491 - Loss: -0.000000
Epoch: 492 - Loss: -0.000000
Epoch: 493 - Loss: -0.000000
Epoch: 494 - Loss: -0.000000
Epoch: 495 - Loss: -0.000000
Epoch: 496 - Loss: -0.000000
Epoch: 497 - Loss: -0.000000
Epoch: 498 - Loss: -0.000000
Epoch: 499 - Loss: -0.000000
```

[26]:
```
plt.figure(figsize=(10,8))
plt.plot(loss_log)
```

[26]:
```
[<matplotlib.lines.Line2D at 0x7f719c4505b0>]
```



[27]:
```
# total_step = len(train_iterator)
# for epoch in range(20):
#     for i, (images, labels) in enumerate(train_iterator):

#         #con2d expects (batch, channel, width, height)
# #         images = images.to(device)
# #         labels = labels.to(device)

#         #print(images.size())
#         #print(labels.size())

#         # Forward pass
#         outputs = net(images)
#         loss = loss_func(outputs, labels)

#         # Backward and optimize
#         optimizer.zero_grad()
#         loss.backward()
#         optimizer.step()
```

```
#         if (i+1) % 100 == 0:
#             sys.stdout.write ('\rEpoch [{}/{}], Step [{}/{}], Loss: {:.4f}'
#                   .format(epoch+1, num_epochs, i+1, total_step, loss.item()))
```

[28]:
```
# YOUR CODE GOES HERE
```

1.4 Question 4: Robot maze RL (30 points)

In class, we developed policies using Q learning and SARSA for multiple grid worlds.

Construct a maze in a 10x10 grid with a starting location in the lower left and ending location in the upper right.

Train your Q learning or SARSA agent to find the goal. Show the resulting policy in a grid representation similar to what we developed in class.

In the space below, write code to split the dataset into separate training and validation datasets in an 80%/20% ratio, then provide two scatter plots, one for the training set and one for the validation set, in which the positive and negative examples are plotted in different colors.

[29]:
```
# YOUR CODE GOES HERE
```

[30]:
```
import numpy as np
import matplotlib.pyplot as plt



gamma = 1
s_initial = (9, 0)
s_terminal = (0, 9)

def reward(s):
    if s == s_terminal:
        return 0
    else:
        return -1

def env(s, a):
    srow = s[0]
    scol = s[1]

#     wind = 0
#     if scol == 3 or scol == 4 or scol == 5 or scol == 8:
#         wind = 1
#     elif scol == 6 or scol == 7:
#         wind = 2

    if a == 0:
        srow -= 1
```

```
    elif a == 1:
        srow += 1
    elif a == 2:
        scol += 1
    elif a == 3:
        scol -= 1
    #srow += wind

    if srow > 9:
        srow = 9
    elif srow < 0:
        srow = 0
    if scol < 0:
        scol = 0
    elif scol > 9:
        scol = 9

    return (srow, scol), reward((srow, scol))

# Policy based on Q: epsilon-greedy

def epsilon_greedy(Q, s, epsilon):
    if np.random.uniform() < epsilon:
        return np.random.randint(n_actions)
    else:
        return np.argmax(Q[s])

# Inputs: alpha, epsilon

alpha = 0.5
epsilon = 0.1

# Initialize Q to 0 for all states, actions

Q = {}
n_actions = 4
for srow in range(10):
    for scol in range(10):
        s = (srow, scol)
        Q[s] = [0] * n_actions

# For each episode

n_episodes = 170
t = 0
episode_time_steps = [0]
for episode in range(n_episodes):
```

```
    s = s_initial
    a = epsilon_greedy(Q, s, epsilon)
    while s != s_terminal:
        sprime, r = env(s, a)
        aprime = epsilon_greedy(Q, sprime, epsilon)
        Q[s][a] = Q[s][a] + alpha * (r + gamma * Q[sprime][aprime] - Q[s][a])
        t += 1
        s = sprime
        a = aprime
    episode_time_steps.append(t)



plt.figure(figsize=(14, 4))

# plt.subplot(1, 2, 1)
# plt.plot(episode_time_steps, range(n_episodes+1), 'r-')
# plt.xlabel('Time steps')
# plt.ylabel('Episodes')
# plt.ylim(0, 170)
# plt.title('SARSA for Windy Gridworld')


ax = plt.subplot(1, 2, 1)
plt.xlim(0, 10)
plt.ylim(0, 10)
plt.xticks(np.arange(0, 10, 1))
plt.yticks(np.arange(0, 10, 1))
ax.axes.get_xaxis().set_ticklabels([])
ax.axes.get_yaxis().set_ticklabels([])
plt.grid()



action_names = ['U', 'D', 'R', 'L']
for srow in range(10):
    for scol in range(10):
        s = (srow, scol)
        a = np.argmax(Q[s])
        plt.text(scol+0.4, srow+0.35, action_names[a])
plt.title('Policy according to Q Learning$Q$')
plt.show()
```

Policy according to Q Learning$Q$

R	R	L	R	R	R	U	R	R	U
U	U	U	R	R	R	R	U	U	U
U	U	L	L	U	R	U	R	U	U
R	U	L	U	U	R	U	U	U	U
R	R	U	U	R	U	R	R	U	U
U	L	U	R	R	R	R	U	R	U
U	R	U	U	R	U	R	U	R	U
L	R	R	R	U	U	U	U	U	U
U	R	R	R	U	R	R	R	U	U
R	R	R	R	R	R	R	R	R	U

[ ]: