INF2008 Machine Learning Assignment Submission 01

PLEASE Read the following submission instructions carefully!

The deadline for submission is end of Week 09, Friday 8th March 2359.

There are two submissions.

- Submission one: the results of this notebook via an xsite quiz
- Submission two: pdf copy of your teams submission notebook via xsite dropbox.

To submit do the following:

1. Replace the group_random_value1 and group_random_value2 in the next cell with your group random values.
2. Find all the cells with the #TODO and paste your correspoding codes there.
3. Run all and copy the outputs generated into the xsite quiz by the deadline.

How to calculate your group random values:

Take the numerical portions of your student IDs and sum all of them together. Lets call this student_sum.

- group_random_value1 = student_sum mod 187
- group_random_value2 = student_sum mod 557

For example, if the sum of all the student IDs in your team is 187441465, your group random values will be:

Keith 2003269

Davis 2003253

Ivan 2003108

Ri Chee 2200747

8210377

```
group_random_value1 = 187441465 mod 187
                    = 145

group_random_value2 = 187441465 mod 557
                    = 382
```

To ensure that the outputs are correct, after pasting your codes, I highly recommend deleting the runtime, and then doing a run all.

When you submit in the quiz, please just paste the output, without any other accompanying text etc..

After that please save a copy of this notebook in pdf and submit it into the xsite dropbox.

```
# TODO Please change the group random values below. Very important!

# these are the original values
group_random_value1 = 42
group_random_value2 = 5

# please replace these two by your group random values
# you can comment out the following two lines if you wish to regenerate the ori
group_random_value1 = 142
group_random_value2 = 197

# these are the original values
random_value1 = group_random_value1
random_value2 = group_random_value2
```

A1.0 Normalization. (2 marks)

```
# Provided Code
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.model_selection import train_test_split

centers = [(-1, -1), (1, 1)]
cluster_std = [1.5, 1.5]

X, y = make_blobs(n_samples=100, cluster_std=cluster_std, centers=centers, n_fe
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, rando
```

```
import numpy as np
# TODO: Normalize the training and testing data.

def normalizedata(x_train, x_test):
    # Calculate mean and standard deviation of training data
    x_train_mean = np.mean(x_train, axis=0)
    x_train_std = np.std(x_train, axis=0)

    # Normalize training data
    x_train_normalized = (x_train - x_train_mean) / x_train_std

    # Normalize test data using mean and standard deviation of training data
    x_test_normalized = (x_test - x_train_mean) / x_train_std

    return x_train_normalized, x_test_normalized


X_train, X_test = normalizedata(X_train, X_test)
```

A1.0.1

```
# TO RUN and submit in xsite
# X_train and X_test should be the normalized values.
print(X_train[:5])
```

```
[[ 0.44300146  1.065435  ]
 [-1.48010146  0.8278021 ]
 [-1.10164804 -1.07549481]
 [ 1.08006907 -0.07288196]
 [ 0.25208597  0.38951366]]
```

A1.0.2

```
# TO RUN and submit in xsite
print(X_test[:5])
```

```
[[-1.08720174  0.03543276]
 [ 0.91034055  0.88386129]
 [-0.69491235  1.28996629]
 [ 0.68600514 -0.70418231]
 [-2.10110375 -1.18586219]]
```

A1.1 Create a simple MLP in pytorch. (5 marks)

```
# Provided Code
import torch

X_train_tensor = torch.FloatTensor(X_train)
y_train_tensor = torch.LongTensor(y_train)

X_test_tensor = torch.FloatTensor(X_test)
y_test_tensor = torch.LongTensor(y_test)

from torch.utils.data import TensorDataset, DataLoader

# Create TensorDatasets
train_dataset = TensorDataset(X_train_tensor, y_train_tensor)
test_dataset = TensorDataset(X_test_tensor, y_test_tensor)

# Create DataLoader
batch_size = 8
train_loader = DataLoader(dataset=train_dataset, batch_size=batch_size, shuffle
test_loader = DataLoader(dataset=test_dataset, batch_size=batch_size, shuffle=F
```

```
# TODO: Create the MLP module here.
import torch
import torch.nn as nn


class MLP(nn.Module):
    def __init__(self):
        super().__init__()
        self.layers = nn.Sequential()
        self.layers.append(nn.Linear(2, 64))
        self.layers.append(nn.ReLU())
        self.layers.append(nn.Linear(64, 32))
        self.layers.append(nn.ReLU())
        self.layers.append(nn.Linear(32, 2))

    def forward(self, x):
        x = self.layers(x)

        return x
```

```
# Provided Code
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = MLP().to(device)
```

```
# TODO: Create the optimizer here
loss_function = nn.CrossEntropyLoss()
optimizer = torch.optim.Adagrad(model.parameters(), lr=0.001)
```

```
# TODO: Create the training loop here
num_epocs = 100

for epoc in range(num_epocs):
    for inputs, labels in train_loader:
        # Zero the gradients
        optimizer.zero_grad()

        # Forward pass
        outputs = model(inputs)

        # Calculate the loss
        loss = loss_function(outputs, labels)

        # Backward pass
        loss.backward()

        # Update weights
        optimizer.step()

    if epoc % 20 == 0:
        print(f'Epoc {epoc + 1}/{num_epocs}, Loss: {loss.item()}')
```

```
Epoc 1/100, Loss: 0.6871647238731384
Epoc 21/100, Loss: 0.47603702545166016
Epoc 41/100, Loss: 0.638272225856781
Epoc 61/100, Loss: 0.4310370683670044
Epoc 81/100, Loss: 0.496544748544693
```

```
# TODO Evaluate the model accuracy
model.eval()

correct = 0
total = 0

with torch.no_grad():
    for inputs, labels, in test_loader:
        # set the forward pass
        outputs = model(inputs)

        # get predictions
        _, predictions = torch.max(outputs, 1)

        batch_size = labels.size(0)

        # get the number of observations in batch
        total += batch_size

        # get the number of correct predictions
        for prediction, label in zip(predictions, labels):
            correct += (prediction == label)
```

A1.1.1

```
# TO RUN and submit in xsite
accuracy = correct / total
print(f'{accuracy * 100:.2f}%')
```

```
90.91%
```

A1.2 Create a simple perceptron using sklearn. (3 marks)

```
#TODO
from sklearn.neural_network import MLPClassifier

clf = MLPClassifier(hidden_layer_sizes=(), solver='lbfgs', max_iter=1000, alpha
clf.fit(X_train, y_train)
```

```
▾ MLPClassifier
MLPClassifier(alpha=0.001, hidden_layer_sizes=(), max_iter=1000, random_sta
              solver='lbfgs')
```

A1.2.1

```
# TO RUN and submit in xsite
print(clf.coefs_[0])
```

```
[[3.58988863]
 [2.66931339]]
```

A1.2.2

```
# TO RUN and submit in xsite
print(clf.intercepts_[0])
```

```
[0.36002615]
```

A1.2.3

```
# TO RUN and submit in xsite
clf.score(X_test, y_test)
```

```
0.9090909090909091
```

A1.3 Create a simple perceptron using python. (5 marks)

```
#TODO Create the perceptron class
class Perceptron:
    def __init__(self):
        self.weights = np.zeros((2, 1))
        self.bias = np.zeros((1, 1))

    def predict(self, inputs):
        # Do the forward pass
        weighted_sum = np.dot(inputs, self.weights) + self.bias
        # Apply a step function (binary threshold) as the activation function.
        prediction = np.where(weighted_sum >= 0, 1, 0)
        return prediction.flatten()

    def train(self, inputs, targets, learning_rate=0.001, epochs=1000):
        for epoch in range(epochs):
            for i in range(len(inputs)):
                # Forward pass
                prediction = self.predict(inputs[i])
                # Compute the error
                error = targets[i] - prediction
                # Update weights and bias
                self.weights += learning_rate * error * inputs[i].reshape(-1, 1
                self.bias += learning_rate * error
```

```
# Provided Code
perceptron = Perceptron()

# Train the perceptron
perceptron.train(X_train, y_train, learning_rate=0.01, epochs=1000)

# Make predictions
predictions = perceptron.predict(X_train)
correct = (predictions == y_train).sum().item()
```

A1.3.1

```
# TO RUN and submit in xsite
print(correct / y_train.shape[0])
```

```
0.8805970149253731
```

Double-click (or enter) to edit

A1.4 Creation of Activation Functions (6 marks)

```
#TODO Create the Sigmoid class
class Sigmoid():
    def __init__(self):
        return

    def forward(self, x):
        self.x = x
        sigmoid_x = 1 / (1 + np.exp(-x))
        return sigmoid_x

    def derivative(self):
        return self.forward(self.x) - self.forward(self.x) ** 2
```

```
#TODO Create the tanh class
class Tanh():
    def __init__(self):
        return

    def forward(self, x):
        self.x = x
        tanh_x = np.sinh(x) / np.cosh(x)
        return tanh_x

    def derivative(self):
        return 1 - self.forward(self.x) ** 2
```

```
#TODO Create the relu class
class ReLU():
    def __init__(self):
        return

    def forward(self, x):
        self.x = x
        relu_x = np.maximum(x, 0)
        return relu_x

    def derivative(self):
        return np.where(self.forward(self.x) > 0, 1, 0)
```

```
# Provided Code
sample_data = X_train[:5]

sigmoid = Sigmoid()
sample_data_sigmoid_forward = sigmoid.forward(sample_data)
sample_data_sigmoid_derivative = sigmoid.derivative()

tanh = Tanh()
sample_data_tanh_forward = tanh.forward(sample_data)
sample_data_tanh_derivative = tanh.derivative()

relu = ReLU()
sample_data_relu_forward = relu.forward(sample_data)
sample_data_relu_derivative = relu.derivative()
```

A1.4.1

```
# TO RUN and submit in xsite
print(sample_data_sigmoid_forward)
```

```
[[0.60897398 0.74372781]
 [0.1854121  0.69588999]
 [0.24943123 0.25435953]
 [0.74650705 0.48178757]
 [0.56268986 0.59616562]]
```

A1.4.2

```
# TO RUN and submit in xsite
print(sample_data_sigmoid_derivative)
```

```
[[0.23812467 0.19059675]
 [0.15103445 0.21162711]
 [0.18721529 0.18966076]
 [0.18923427 0.24966831]
 [0.24606998 0.24075217]]
```

A1.4.3

```
# TO RUN and submit in xsite
print(sample_data_tanh_forward)
```

```
[[ 0.41612925  0.78773514]
 [-0.90148699  0.67929407]
 [-0.80109022 -0.79152243]
 [ 0.79322471 -0.07275319]
 [ 0.2468785   0.37094088]]
```

A1.4.4

```
# TO RUN and submit in xsite
print(sample_data_tanh_derivative)
```

```
[[0.82683645 0.37947334]
 [0.1873212  0.53855956]
 [0.35825446 0.37349224]
 [0.37079456 0.99470697]
 [0.939051   0.86240286]]
```

A1.4.5

```
# TO RUN and submit in xsite
print(sample_data_relu_forward)
```

```
[[0.44300146 1.065435  ]
 [0.         0.8278021 ]
 [0.         0.        ]
 [1.08006907 0.        ]
 [0.25208597 0.38951366]]
```

A1.4.6

```
# TO RUN and submit in xsite
print(sample_data_relu_derivative)
```

```
[[1 1]
 [0 1]
 [0 0]
 [1 0]
 [1 1]]
```

A1.5 Creation of Mean Square Error Loss Function (7 marks)

```
#TODO Class for MSELoss
class MSELoss:
    def forward(self, A, Y):
        self.A = A
        self.Y = Y
        N = A.shape[0]
        C = A.shape[1]
        se = np.sum((A - Y))
        sse = np.sum((A - Y) ** 2)
        mse = (A - Y) * (A - Y) / (2 * N * C)

        return mse

    def backward(self):
        dLdA = self.A - self.Y
        return dLdA
```

```
# Provided code
mseless = MSELoss()

from numpy import random

N = 5
C = 4

random.seed(5)
Y = random.random(size=(N, C))

random.seed(8)
A = Y + random.random(size=(N, C)) / 10
```

A1.5.1

```
# TO RUN and submit in xsite
print(mseless.forward(A, Y))
```

```
[[1.90719730e-04 2.34517754e-04 1.88874787e-04 7.04519413e-05]
 [1.35406187e-05 3.24831847e-08 4.63258509e-05 4.04716542e-05]
 [6.82972030e-05 5.72146776e-05 7.71052033e-05 7.38170910e-05]
 [1.44740519e-04 1.26869383e-04 9.60014751e-05 4.53885492e-05]
 [2.08910930e-05 2.37098508e-04 2.78512784e-05 1.19684761e-05]]
```

A1.5.2

```
# TO RUN and submit in xsite
print(mseless.backward())
```

```
[[0.08734294 0.09685407 0.08691945 0.05308557]
 [0.02327283 0.00113988 0.04304688 0.04023514]
 [0.05226747 0.04783918 0.05553565 0.0543386 ]
 [0.07608956 0.07123746 0.06196821 0.04260918]
 [0.0289075  0.09738552 0.0333774  0.02188011]]
```

A1.6 and A1.7 (12 marks)

```
#TODO Class for single linear layer
class Linear:
    def __init__(self, in_features, out_features, debug=False):
        self.W = np.ones((out_features, in_features), dtype="f")
        self.b = np.zeros((out_features, 1), dtype="f")
        self.dLdW = np.zeros((out_features, in_features), dtype="f")
        self.dLdb = np.zeros((out_features, 1), dtype="f")

    def forward(self, A):
        self.A = A
        self.N = A.shape[0]
        self.Ones = np.ones((self.N, 1), dtype="f")
        Z = np.dot(A, self.W.T) + np.dot(self.Ones, self.b.T)

        return Z

    def backward(self, dLdZ):
        dZdA = self.W.T
        dZdW = self.A
        dZdb = self.Ones
        dLdA = np.dot(dLdZ, dZdA.T)
        dLdW = np.dot(dLdZ.T, dZdW) / self.N
        dLdb = np.dot(dLdZ.T, dZdb) / self.N
        dLdi = None
        dZdi = None

        # Accumulate gradients
        self.dLdW = dLdW
        self.dLdb = dLdb

        return dLdA
```

A1.6.1

```
# TO RUN and submit in xsite
C_in = C
C_out = 1
linear_layer = Linear(C_in, C_out, True)
Z = linear_layer.forward(A)
print(Z)
```

```
[[2.54225757]
 [2.49217563]
 [1.51368419]
 [2.00554699]
 [2.10049121]]
```

A1.6.2

```
# TO RUN and submit in xsite
random.seed(random_value2)
Y = Z + random.random(size=(N, C_out)) / 10
print(Y)
```

```
[[2.62904825]
 [2.57064592]
 [1.61131037]
 [2.0824219 ]
 [2.17404337]]
```

A1.6.3

```
# TO RUN and submit in xsite
mseless = MSELoss()
mse_forward = mseless.forward(Z, Y)
print(mse_forward)
```

```
[[0.00075326]
 [0.00061576]
 [0.00095309]
 [0.00059098]
 [0.00054099]]
```

A1.6.4

```
# TO RUN and submit in xsite
dLdz = mseless.backward()
dLdA = linear_layer.backward(dLdz)
print(dLdA)
```

```
[[-0.08679068 -0.08679068 -0.08679068 -0.08679068]
 [-0.07847029 -0.07847029 -0.07847029 -0.07847029]
 [-0.09762618 -0.09762618 -0.09762618 -0.09762618]
 [-0.07687491 -0.07687491 -0.07687491 -0.07687491]
 [-0.07355216 -0.07355216 -0.07355216 -0.07355216]]
```

A1.6.5

```
# TO RUN and submit in xsite
print(dLdz)
```

```
[[-0.08679068]
 [-0.07847029]
 [-0.09762618]
 [-0.07687491]
 [-0.07355216]]
```

A1.6.6

```
# TO RUN and submit in xsite
print(linear_layer.dLdW)
```

```
[[-0.03468935 -0.04033083 -0.04467615 -0.05483429]]
```

A1.6.7

```
# TO RUN and submit in xsite
print(linear_layer.dLdb)
```

```
[[-0.08266284]]
```