

# CSCE 5218 / CSCE 4930 Deep Learning

## HW1a The Perceptron (20 pt)

```
In [1]: #!pip install wget
```

```
In [2]: import wget
```

```
In [3]: # # Get the datasets
# !wget http://huang.eng.unt.edu/CSCE-5218/test.dat
# !wget http://huang.eng.unt.edu/CSCE-5218/train.dat
```

```
In [4]: # # Take a peek at the datasets
# !head train.dat
# !head test.dat
```

```
In [5]: url_train = 'http://huang.eng.unt.edu/CSCE-5218/train.dat'
filename_train = wget.download(url_train)
print(filename_train)
```

```
100% [.....] 11
244 / 11244train (10).dat
```

```
In [6]: url_test = 'http://huang.eng.unt.edu/CSCE-5218/test.dat'
filename_test = wget.download(url_test)
print(filename_test)
```

```
100% [.....]
2844 / 2844test (9).dat
```

```
In [7]: with open(filename_train) as f_train:
    for _ in range(10):
        print(f_train.readline())
```

A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11
A12	A13	.								
1	1	0	0	0	0	0	0	1	1	0
0	1	0								
0	0	1	1	0	1	1	0	0	0	0
0	1	0	1	1	0	1	0	1	1	1
0	1	1								

0	0	1	0	0	1	0	1	0	1	1
1	1	0								
0	1	0	0	0	0	0	1	1	1	1
1	1	0								
0	1	1	1	0	0	0	1	0	1	1
0	1	1								
0	1	1	0	0	0	1	0	0	0	0
0	1	0								
0	0	0	1	1	0	1	1	1	0	0
0	1	0								
0	0	0	0	0	0	1	0	1	0	1
0	1	0								

```
In [8]: with open(filename_test) as f_test:
    for _ in range(10):
        print(f_test.readline())
```

A1 A12	A2 A13	A3	A4	A5	A6	A7	A8	A9	A10	A11
1	1	1	1	0	0	1	1	0	0	0
1	1	0								
0	0	0	1	0	0	1	1	0	1	0
0	1	0								
0	1	1	1	0	1	1	1	1	0	0
0	1	0								
0	1	1	0	1	0	1	1	1	0	1
0	1	0								
0	1	0	0	0	1	0	1	0	1	0
0	1	0								
0	1	1	0	0	1	1	1	1	1	1
0	1	0								
0	1	1	1	0	0	1	1	0	0	0
1	1	0								
0	1	0	0	1	0	0	1	1	0	1
1	1	0								
1	1	1	1	0	0	1	1	0	0	0
0	1	0								

## Build the Perceptron Model

You will need to complete some of the function definitions below. DO NOT import any other libraries to complete this.

In [9]:

```
import math
import itertools
import re

# Corpus reader, all columns but the last one are coordinates;
# the last column is the label
def read_data(file_name):
    f = open(file_name, 'r')

    data = []
    # Discard header line
    f.readline()
    for instance in f.readlines():
        if not re.search('\t', instance): continue
        instance = list(map(int, instance.strip().split('\t')))
        # Add a dummy input so that w0 becomes the bias
        instance = [-1] + instance
        data += [instance]
    return data

def dot_product(array1, array2):
    result = sum([i*j for (i, j) in zip(array1, array2)])
    return result

def sigmoid(x):
    result = 1 / (1 + math.exp(-x))
    return result

# The output of the model, which for the perceptron is
# the sigmoid function applied to the dot product of
# the instance and the weights
def output(weight, instance):
    #TODO: return the output of the model
    result = sigmoid(dot_product(weight, instance))
    return result

# Predict the label of an instance; this is the definition of the perceptron
# you should output 1 if the output is >= 0.5 else output 0
def predict(weights, instance):
    result = sigmoid(dot_product(weights, instance))
    if result >= 0.5 :
        return 1
    else:
        return 0

# Accuracy = percent of correct predictions
def get_accuracy(weights, instances):
    # You do not to write code like this, but get used to it
    correct = sum([1 if predict(weights, instance) == instance[-1] else 0
                  for instance in instances])
    return correct * 100 / len(instances)
```

```

# Train a perceptron with instances and hyperparameters:
#     lr (learning rate)
#     epochs
# The implementation comes from the definition of the perceptron
#
# Training consists on fitting the parameters which are the weights
# that's the only thing training is responsible to fit
# (recall that w0 is the bias, and w1..wn are the weights for each coordinate)
#
# Hyperparameters (lr and epochs) are given to the training algorithm
# We are updating weights in the opposite direction of the gradient of the error,
# so with a "decent" lr we are guaranteed to reduce the error after each iteration.
def train_perceptron(instances, lr, epochs):

    #TODO: zero initial weights
    weights = [0] * (len(instances[0])-1)

    for _ in range(epochs):
        for instance in instances:
            #TODO: Calculate error
            in_value = dot_product(weights, instance)
            output = sigmoid(in_value)
            error = instance[-1] - output
            #TODO: weights update
            for i in range(0, len(weights)):
                weights[i] += lr * error * output * (1-output) * instance[i]

    return weights

```

## Run it

```
In [10]: instances_tr = read_data("train.dat")
instances_te = read_data("test.dat")
```

```
In [11]: lr = 0.005
epochs = 5

weights = train_perceptron(instances_tr, lr, epochs)
accuracy = get_accuracy(weights, instances_te)
print(f"#tr: {len(instances_tr):3}, epochs: {epochs:3}, learning rate: {lr:.3f}; "
      f"Accuracy (test, {len(instances_te)} instances): {accuracy:.1f}")
```

```
#tr: 400, epochs: 5, learning rate: 0.005; Accuracy (test, 100 instances): 68.0
```

## Questions

Answer the following questions. Include your implementation and the output for each question.

### Question 1

In `train_perceptron(instances, lr, epochs)`, we have the following code:

```

in_value = dot_product(weights, instance)
output = sigmoid(in_value)
error = instance[-1] - output

```

Why don't we have the following code snippet instead?

```

output = predict(weights, instance)
error = instance[-1] - output

```

**TODO Add your answer here (text only)**

```
def train_perceptron(instances, lr, epochs):
```

```

#zero initial weights
weights = [0] * (len(instances[0])-1)

for _ in range(epochs):
    for instance in instances:
        #TODO: Calculate error
        output = predict(weights, instance)
        error = instance[-1] - output
        #TODO: weights update
        for i in range(0, len(weights)):
            weights[i] += lr * error * output * (1-output) * instance[i]
return weights

```

## Question 2

Train the perceptron with the following hyperparameters and calculate the accuracy with the test dataset.

```

tr_percent = [5, 10, 25, 50, 75, 100] # percent of the training dataset
to train with
num_epochs = [5, 10, 20, 50, 100]           # number of epochs
lr = [0.005, 0.01, 0.05]                  # learning rate

```

TODO: Write your code below and include the output at the end of each training loop (NOT AFTER EACH EPOCH) of your code. The output should look like the following:

```

# tr: 20, epochs: 5, learning rate: 0.005; Accuracy (test, 100
instances): 68.0
# tr: 20, epochs: 10, learning rate: 0.005; Accuracy (test, 100
instances): 68.0
# tr: 20, epochs: 20, learning rate: 0.005; Accuracy (test, 100
instances): 68.0
[and so on for all the combinations]

```

You will get different results with different hyperparameters.

**TODO Add your answer here (code and output in the format above)**

In [12]:

```

instances_tr = read_data("train.dat")
instances_te = read_data("test.dat")

# percent of the training dataset to train with ->tr
tr_percent = [5, 10, 25, 50, 75, 100]

# number of epochs -> num_epochs
num_epochs = [5, 10, 20, 50, 100]

# Learning rate -> lr_array
lr_array = [0.005, 0.01, 0.05]

tr_plot = []
lr_plot = []
acc_plot = []
for lr in lr_array:
    for tr_size in tr_percent:
        for epochs in num_epochs:
            size = round(len(instances_tr)*tr_size/100)
            pre_instances = instances_tr[0:size]
            weights = train_perceptron(pre_instances, lr, epochs)
            accuracy = get_accuracy(weights, instances_te)
            print(f"#tr: {len(pre_instances)}:0, epochs: {epochs}:1, learning rate: {lr:.3f}")
            f"Accuracy (test, {len(instances_te)}) instances): {accuracy:.1f}"
            tr_plot.append(len(pre_instances))
            lr_plot.append(lr)
            acc_plot.append(accuracy)

```

```

#tr: 20, epochs: 100, learning rate: 0.005; Accuracy (test, 100 instances): 68.0
#tr: 40, epochs: 100, learning rate: 0.005; Accuracy (test, 100 instances): 68.0
#tr: 100, epochs: 100, learning rate: 0.005; Accuracy (test, 100 instances): 68.0
#tr: 200, epochs: 100, learning rate: 0.005; Accuracy (test, 100 instances): 74.0
#tr: 300, epochs: 100, learning rate: 0.005; Accuracy (test, 100 instances): 78.0
#tr: 400, epochs: 100, learning rate: 0.005; Accuracy (test, 100 instances): 77.0
#tr: 20, epochs: 100, learning rate: 0.010; Accuracy (test, 100 instances): 68.0
#tr: 40, epochs: 100, learning rate: 0.010; Accuracy (test, 100 instances): 68.0
#tr: 100, epochs: 100, learning rate: 0.010; Accuracy (test, 100 instances): 71.0
#tr: 200, epochs: 100, learning rate: 0.010; Accuracy (test, 100 instances): 78.0
#tr: 300, epochs: 100, learning rate: 0.010; Accuracy (test, 100 instances): 80.0
#tr: 400, epochs: 100, learning rate: 0.010; Accuracy (test, 100 instances): 80.0
#tr: 20, epochs: 100, learning rate: 0.050; Accuracy (test, 100 instances): 64.0
#tr: 40, epochs: 100, learning rate: 0.050; Accuracy (test, 100 instances): 69.0
#tr: 100, epochs: 100, learning rate: 0.050; Accuracy (test, 100 instances): 77.0
#tr: 200, epochs: 100, learning rate: 0.050; Accuracy (test, 100 instances): 76.0
#tr: 300, epochs: 100, learning rate: 0.050; Accuracy (test, 100 instances): 77.0
#tr: 400, epochs: 100, learning rate: 0.050; Accuracy (test, 100 instances): 80.0

```

In [13]:

```

instances_tr = read_data("train.dat")
instances_te = read_data("test.dat")

# percent of the training dataset to train with ->tr_percent
tr_percent = [5, 10, 25, 50, 75, 100]

# number of epochs ->num_epochs
num_epochs = [5, 10, 20, 50, 100]

# Learning rate ->lr_array
lr_array = [0.005, 0.01, 0.05]

```

```

for lr in lr_array:
    for tr_size in tr_percent:
        for epochs in num_epochs:
            size = round(len(instances_tr)*tr_size/100)
            pre_instances = instances_tr[0:size]
            weights = train_perceptron(pre_instances, lr, epochs)
            accuracy = get_accuracy(weights, instances_te)
            print(f"#tr: {len(pre_instances):0}, epochs: {epochs:3}, learning rate: {lr:.3f}")
            f"Accuracy (test, {len(instances_te)} instances): {accuracy:.1f}"

```

```

#tr: 20, epochs: 100, learning rate: 0.005; Accuracy (test, 100 instances): 68.0
#tr: 40, epochs: 100, learning rate: 0.005; Accuracy (test, 100 instances): 68.0
#tr: 100, epochs: 100, learning rate: 0.005; Accuracy (test, 100 instances): 68.0
#tr: 200, epochs: 100, learning rate: 0.005; Accuracy (test, 100 instances): 74.0
#tr: 300, epochs: 100, learning rate: 0.005; Accuracy (test, 100 instances): 78.0
#tr: 400, epochs: 100, learning rate: 0.005; Accuracy (test, 100 instances): 77.0
#tr: 20, epochs: 100, learning rate: 0.010; Accuracy (test, 100 instances): 68.0
#tr: 40, epochs: 100, learning rate: 0.010; Accuracy (test, 100 instances): 68.0
#tr: 100, epochs: 100, learning rate: 0.010; Accuracy (test, 100 instances): 71.0
#tr: 200, epochs: 100, learning rate: 0.010; Accuracy (test, 100 instances): 78.0
#tr: 300, epochs: 100, learning rate: 0.010; Accuracy (test, 100 instances): 80.0
#tr: 400, epochs: 100, learning rate: 0.010; Accuracy (test, 100 instances): 80.0
#tr: 20, epochs: 100, learning rate: 0.050; Accuracy (test, 100 instances): 64.0
#tr: 40, epochs: 100, learning rate: 0.050; Accuracy (test, 100 instances): 69.0
#tr: 100, epochs: 100, learning rate: 0.050; Accuracy (test, 100 instances): 77.0
#tr: 200, epochs: 100, learning rate: 0.050; Accuracy (test, 100 instances): 76.0
#tr: 300, epochs: 100, learning rate: 0.050; Accuracy (test, 100 instances): 77.0
#tr: 400, epochs: 100, learning rate: 0.050; Accuracy (test, 100 instances): 80.0

```

## Question 3

Write a couple paragraphs interpreting the results with all the combinations of hyperparameters. Drawing a plot will probably help you make a point. In particular, answer the following:

- A. Do you need to train with all the training dataset to get the highest accuracy with the test dataset?
- B. How do you justify that training the second run obtains worse accuracy than the first one (despite the second one uses more training data)?  

```
#tr: 100, epochs: 20, learning rate: 0.050; Accuracy (test, 100 instances): 71.0
#tr: 200, epochs: 20, learning rate: 0.005; Accuracy (test, 100 instances): 68.0
```
- C. Can you get higher accuracy with additional hyperparameters (higher than 80.0 )?
- D. Is it always worth training for more epochs (while keeping all other hyperparameters fixed)?

### TODO: Add your answer here (code and text)

- A. Yes, as training the dataset being used for more times, there is a greater chance of getting a higher accuracy
- B. On analysing the data it is observed that the learning rate of the second run is too low which results in a lower accuracy in a short term.
- C. In this scenario, there is a small chance to get higher accuracy with additional hyperparamters itself.

- D. It is not worth training for more epochs as there is a probability or chance that it may lead to overfitting problem.

In [14]:

```
import pandas as pd
import matplotlib.pyplot as plt

result_df = pd.DataFrame({'tr':tr_plot,'learning_rate':lr_plot,'Accuracy':acc_plot})
```

In [15]:

```
plt.figure()
for lr in set(result_df['learning_rate']):
    selected_data = result_df.loc[result_df['learning_rate'] == lr]
    plt.plot(selected_data['tr'],selected_data['Accuracy'],label=lr)
plt.legend()
plt.show()
```

