CSE 598 Introduction to Deep Learning

HW1a The Perceptron (20 pt)

```
# Get the datasets
!wget http://www.cse.unt.edu/~blanco/csce5218/hw1a/train.dat
!wget http://www.cse.unt.edu/~blanco/csce5218/hw1a/test.dat
```

```
--2021-09-11 06:36:13--  http://www.cse.unt.edu/~blanco/csce5218/hw1a/train.da
Resolving www.cse.unt.edu (www.cse.unt.edu)... 129.120.151.91
Connecting to www.cse.unt.edu (www.cse.unt.edu)|129.120.151.91|:80... connect
HTTP request sent, awaiting response... 200 OK
Length: 11244 (11K) [application/x-ns-proxy-autoconfig]
Saving to: ‘train.dat’

train.dat           100%[===================>]  10.98K  --.-KB/s    in 0s

2021-09-11 06:36:13 (183 MB/s) - ‘train.dat’ saved [11244/11244]

--2021-09-11 06:36:13--  http://www.cse.unt.edu/~blanco/csce5218/hw1a/test.dat
Resolving www.cse.unt.edu (www.cse.unt.edu)... 129.120.151.91
Connecting to www.cse.unt.edu (www.cse.unt.edu)|129.120.151.91|:80... connect
HTTP request sent, awaiting response... 200 OK
Length: 2844 (2.8K) [application/x-ns-proxy-autoconfig]
Saving to: ‘test.dat’

test.dat            100%[===================>]   2.78K  --.-KB/s    in 0s

2021-09-11 06:36:13 (322 MB/s) - ‘test.dat’ saved [2844/2844]
```

```
# Take a peek at the datasets
!head train.dat
!head test.dat
```

```
A1    A2    A3    A4    A5    A6    A7    A8    A9    A10
1     1     0     0     0     0     0     0     1     1
0     0     1     1     0     1     1     0     0     0
0     1     0     1     1     0     1     0     1     1
0     0     1     0     0     1     0     1     0     1
0     1     0     0     0     0     0     1     1     1
0     1     1     1     0     0     0     1     0     1
0     1     1     0     0     0     1     0     0     0
0     0     0     1     1     0     1     1     1     0
0     0     0     0     0     0     1     0     1     0
A1    A2    A3    A4    A5    A6    A7    A8    A9    A10
1     1     1     1     0     0     1     1     0     0
0     0     0     1     0     0     1     1     0     1
0     1     1     1     0     1     1     1     1     0
0     1     1     0     1     0     1     1     1     0
0     1     0     0     0     1     0     1     0     1
0     1     1     0     0     1     1     1     1     1
0     1     1     1     0     0     1     1     0     0
0     1     0     0     1     0     0     1     1     0
1     1     1     1     0     0     1     1     0     0
```

```
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):
    # You do not to write code like this, but get used to it
    return sum([w * x for w, x in zip(array1, array2)])


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


# 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)


# Predict a new instance; this is the definition of the perceptron
def predict(weights, instance):
    if sigmoid(dot_product(weights, instance)) >= 0.5:
        return 1
    return 0


# Train a perceptron with instances
#   and hyperparameters lr (leearning rate) and epochs
# The implementation comes from the definition of the perceptron
# Training consists on fitting the parameters
#   The parameters are the weights, that's the only thing training is responsible
#     (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 iterat
def train_perceptron(instances, lr, epochs):
    weights = [0] * (len(instances[0])-1)
    # weights = [0, 0, 0, ...,  0]

    while epochs > 0:
        for instance in instances:
            in_value = dot_product(weights, instance)
            output = sigmoid(in_value)
            error = instance[-1] - output
            for i in range(0, len(weights)):
                weights[i] += lr * error * output * (1-output) * instance[i]
```

```
        epochs -= 1
        if epochs == 0:
            break

    return weights


instances_tr = read_data("train.dat")
instances_te = read_data("test.dat")
tr_percent = [5, 10, 25, 50, 75, 100] # percent of the training dataset to train
num_epochs = [5, 10, 20, 50, 100]           # number of epochs
learning_rate = [0.005, 0.01, 0.05]          # learning rate
for epochs in num_epochs :
  for lr in learning_rate :
    for tp in tr_percent :
      limit = int(len(instances_tr)*tp/100)
      new_tr = instances_tr[:limit]
      weights = train_perceptron(new_tr, lr, epochs)
      accuracy = get_accuracy(weights, instances_te)
      print(f"#tr: {len(instances_tr):3}, epochs: {epochs:3}, learning rate: {lr:
      f"Accuracy (test, {len(instances_te)} instances): {accuracy:.1f}")
```

```
#tr: 400, epochs:   5, learning rate: 0.005; Accuracy (test, 100 instances):
#tr: 400, epochs:   5, learning rate: 0.005; Accuracy (test, 100 instances):
#tr: 400, epochs:   5, learning rate: 0.005; Accuracy (test, 100 instances):
#tr: 400, epochs:   5, learning rate: 0.005; Accuracy (test, 100 instances):
#tr: 400, epochs:   5, learning rate: 0.005; Accuracy (test, 100 instances):
#tr: 400, epochs:   5, learning rate: 0.005; Accuracy (test, 100 instances):
#tr: 400, epochs:   5, learning rate: 0.010; Accuracy (test, 100 instances):
#tr: 400, epochs:   5, learning rate: 0.010; Accuracy (test, 100 instances):
#tr: 400, epochs:   5, learning rate: 0.010; Accuracy (test, 100 instances):
#tr: 400, epochs:   5, learning rate: 0.010; Accuracy (test, 100 instances):
#tr: 400, epochs:   5, learning rate: 0.010; Accuracy (test, 100 instances):
#tr: 400, epochs:   5, learning rate: 0.010; Accuracy (test, 100 instances):
#tr: 400, epochs:   5, learning rate: 0.050; Accuracy (test, 100 instances):
#tr: 400, epochs:   5, learning rate: 0.050; Accuracy (test, 100 instances):
#tr: 400, epochs:   5, learning rate: 0.050; Accuracy (test, 100 instances):
#tr: 400, epochs:   5, learning rate: 0.050; Accuracy (test, 100 instances):
#tr: 400, epochs:   5, learning rate: 0.050; Accuracy (test, 100 instances):
#tr: 400, epochs:   5, learning rate: 0.050; Accuracy (test, 100 instances):
#tr: 400, epochs:  10, learning rate: 0.005; Accuracy (test, 100 instances):
#tr: 400, epochs:  10, learning rate: 0.005; Accuracy (test, 100 instances):
#tr: 400, epochs:  10, learning rate: 0.005; Accuracy (test, 100 instances):
#tr: 400, epochs:  10, learning rate: 0.005; Accuracy (test, 100 instances):
#tr: 400, epochs:  10, learning rate: 0.005; Accuracy (test, 100 instances):
#tr: 400, epochs:  10, learning rate: 0.005; Accuracy (test, 100 instances):
#tr: 400, epochs:  10, learning rate: 0.010; Accuracy (test, 100 instances):
#tr: 400, epochs:  10, learning rate: 0.010; Accuracy (test, 100 instances):
```

```
#tr: 400, epochs:  10, learning rate: 0.010; Accuracy (test, 100 instances): 6
#tr: 400, epochs:  10, learning rate: 0.010; Accuracy (test, 100 instances): 6
#tr: 400, epochs:  10, learning rate: 0.010; Accuracy (test, 100 instances): 6
#tr: 400, epochs:  10, learning rate: 0.010; Accuracy (test, 100 instances): 6
#tr: 400, epochs:  10, learning rate: 0.010; Accuracy (test, 100 instances): 6
#tr: 400, epochs:  10, learning rate: 0.050; Accuracy (test, 100 instances): 6
#tr: 400, epochs:  10, learning rate: 0.050; Accuracy (test, 100 instances): 6
#tr: 400, epochs:  10, learning rate: 0.050; Accuracy (test, 100 instances): 6
#tr: 400, epochs:  10, learning rate: 0.050; Accuracy (test, 100 instances): 7
#tr: 400, epochs:  10, learning rate: 0.050; Accuracy (test, 100 instances): 7
#tr: 400, epochs:  10, learning rate: 0.050; Accuracy (test, 100 instances): 7
#tr: 400, epochs:  20, learning rate: 0.005; Accuracy (test, 100 instances): 6
#tr: 400, epochs:  20, learning rate: 0.005; Accuracy (test, 100 instances): 6
#tr: 400, epochs:  20, learning rate: 0.005; Accuracy (test, 100 instances): 6
#tr: 400, epochs:  20, learning rate: 0.005; Accuracy (test, 100 instances): 6
#tr: 400, epochs:  20, learning rate: 0.005; Accuracy (test, 100 instances): 6
#tr: 400, epochs:  20, learning rate: 0.005; Accuracy (test, 100 instances): 6
#tr: 400, epochs:  20, learning rate: 0.010; Accuracy (test, 100 instances): 6
#tr: 400, epochs:  20, learning rate: 0.010; Accuracy (test, 100 instances): 6
#tr: 400, epochs:  20, learning rate: 0.010; Accuracy (test, 100 instances): 6
#tr: 400, epochs:  20, learning rate: 0.010; Accuracy (test, 100 instances): 6
#tr: 400, epochs:  20, learning rate: 0.010; Accuracy (test, 100 instances): 7
#tr: 400, epochs:  20, learning rate: 0.010; Accuracy (test, 100 instances): 7
#tr: 400, epochs:  20, learning rate: 0.050; Accuracy (test, 100 instances): 6
#tr: 400, epochs:  20, learning rate: 0.050; Accuracy (test, 100 instances): 6
#tr: 400, epochs:  20, learning rate: 0.050; Accuracy (test, 100 instances): 7
#tr: 400, epochs:  20, learning rate: 0.050; Accuracy (test, 100 instances): 7
#tr: 400, epochs:  20, learning rate: 0.050; Accuracy (test, 100 instances): 7
#tr: 400, epochs:  20, learning rate: 0.050; Accuracy (test, 100 instances): 8
#tr: 400, epochs:  50, learning rate: 0.005; Accuracy (test, 100 instances): 6
#tr: 400, epochs:  50, learning rate: 0.005; Accuracy (test, 100 instances): 6
#tr: 400, epochs:  50, learning rate: 0.005; Accuracy (test, 100 instances): 6
#tr: 400, epochs:  50, learning rate: 0.005; Accuracy (test, 100 instances): 6
#tr: 400, epochs:  50, learning rate: 0.005; Accuracy (test, 100 instances): 7
#tr: 400, epochs:  50, learning rate: 0.005; Accuracy (test, 100 instances): 7
```

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 follosing 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)

ANSWER :

```
The perceptron is being trained here.
So, the input, which is the dot product of weights and instance is feeded to the perc
eptron.

Predict function is used to finding out accuracy but here we are training and looking
 better results.
```

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 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 differet hyperparameters.

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

Code :

```
instances_tr = read_data("train.dat")
instances_te = read_data("test.dat")
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
learning_rate = [0.005, 0.01, 0.05]              # learning rate
for epochs in num_epochs :
  for lr in learning_rate :
    for tp in tr_percent :
      limit = int(len(instances_tr)*tp/100)
      new_tr = instances_tr[:limit]
      weights = train_perceptron(new_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}")
```

Output :

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

```
# tr: 400, epochs:   5, learning rate: 0.005; Accuracy (test, 100 instances): 68.0
# tr: 400, epochs:   5, learning rate: 0.010; Accuracy (test, 100 instances): 68.0
# tr: 400, epochs:   5, learning rate: 0.010; Accuracy (test, 100 instances): 68.0
# tr: 400, epochs:   5, learning rate: 0.010; Accuracy (test, 100 instances): 68.0
# tr: 400, epochs:   5, learning rate: 0.010; Accuracy (test, 100 instances): 68.0
# tr: 400, epochs:   5, learning rate: 0.010; Accuracy (test, 100 instances): 68.0
# tr: 400, epochs:   5, learning rate: 0.010; Accuracy (test, 100 instances): 68.0
# tr: 400, epochs:   5, learning rate: 0.050; Accuracy (test, 100 instances): 68.0
# tr: 400, epochs:   5, learning rate: 0.050; Accuracy (test, 100 instances): 68.0
# tr: 400, epochs:   5, learning rate: 0.050; Accuracy (test, 100 instances): 68.0
# tr: 400, epochs:   5, learning rate: 0.050; Accuracy (test, 100 instances): 71.0
# tr: 400, epochs:   5, learning rate: 0.050; Accuracy (test, 100 instances): 74.0
# tr: 400, epochs:   5, learning rate: 0.050; Accuracy (test, 100 instances): 69.0
# tr: 400, epochs:  10, learning rate: 0.005; Accuracy (test, 100 instances): 68.0
# tr: 400, epochs:  10, learning rate: 0.005; Accuracy (test, 100 instances): 68.0
# tr: 400, epochs:  10, learning rate: 0.005; Accuracy (test, 100 instances): 68.0
# tr: 400, epochs:  10, learning rate: 0.005; Accuracy (test, 100 instances): 68.0
# tr: 400, epochs:  10, learning rate: 0.005; Accuracy (test, 100 instances): 68.0
# tr: 400, epochs:  10, learning rate: 0.005; Accuracy (test, 100 instances): 68.0
# tr: 400, epochs:  10, learning rate: 0.010; Accuracy (test, 100 instances): 68.0
# tr: 400, epochs:  10, learning rate: 0.010; Accuracy (test, 100 instances): 68.0
# tr: 400, epochs:  10, learning rate: 0.010; Accuracy (test, 100 instances): 68.0
# tr: 400, epochs:  10, learning rate: 0.010; Accuracy (test, 100 instances): 68.0
# tr: 400, epochs:  10, learning rate: 0.010; Accuracy (test, 100 instances): 68.0
# tr: 400, epochs:  10, learning rate: 0.010; Accuracy (test, 100 instances): 69.0
# tr: 400, epochs:  10, learning rate: 0.050; Accuracy (test, 100 instances): 68.0
# tr: 400, epochs:  10, learning rate: 0.050; Accuracy (test, 100 instances): 68.0
# tr: 400, epochs:  10, learning rate: 0.050; Accuracy (test, 100 instances): 67.0
# tr: 400, epochs:  10, learning rate: 0.050; Accuracy (test, 100 instances): 77.0
# tr: 400, epochs:  10, learning rate: 0.050; Accuracy (test, 100 instances): 78.0
# tr: 400, epochs:  10, learning rate: 0.050; Accuracy (test, 100 instances): 76.0
# tr: 400, epochs:  20, learning rate: 0.005; Accuracy (test, 100 instances): 68.0
# tr: 400, epochs:  20, learning rate: 0.005; Accuracy (test, 100 instances): 68.0
# tr: 400, epochs:  20, learning rate: 0.005; Accuracy (test, 100 instances): 68.0
# tr: 400, epochs:  20, learning rate: 0.005; Accuracy (test, 100 instances): 68.0
# tr: 400, epochs:  20, learning rate: 0.005; Accuracy (test, 100 instances): 68.0
# tr: 400, epochs:  20, learning rate: 0.005; Accuracy (test, 100 instances): 69.0
# tr: 400, epochs:  20, learning rate: 0.010; Accuracy (test, 100 instances): 68.0
```

```
# tr: 400, epochs:  20, learning rate: 0.010; Accuracy (test, 100 instances): 68.0
# tr: 400, epochs:  20, learning rate: 0.010; Accuracy (test, 100 instances): 68.0
# tr: 400, epochs:  20, learning rate: 0.010; Accuracy (test, 100 instances): 68.0
# tr: 400, epochs:  20, learning rate: 0.010; Accuracy (test, 100 instances): 70.0
# tr: 400, epochs:  20, learning rate: 0.010; Accuracy (test, 100 instances): 70.0
# tr: 400, epochs:  20, learning rate: 0.050; Accuracy (test, 100 instances): 68.0
# tr: 400, epochs:  20, learning rate: 0.050; Accuracy (test, 100 instances): 68.0
# tr: 400, epochs:  20, learning rate: 0.050; Accuracy (test, 100 instances): 70.0
# tr: 400, epochs:  20, learning rate: 0.050; Accuracy (test, 100 instances): 78.0
# tr: 400, epochs:  20, learning rate: 0.050; Accuracy (test, 100 instances): 79.0
# tr: 400, epochs:  20, learning rate: 0.050; Accuracy (test, 100 instances): 80.0
# tr: 400, epochs:  50, learning rate: 0.005; Accuracy (test, 100 instances): 68.0
# tr: 400, epochs:  50, learning rate: 0.005; Accuracy (test, 100 instances): 68.0
# tr: 400, epochs:  50, learning rate: 0.005; Accuracy (test, 100 instances): 68.0
# tr: 400, epochs:  50, learning rate: 0.005; Accuracy (test, 100 instances): 67.0
# tr: 400, epochs:  50, learning rate: 0.005; Accuracy (test, 100 instances): 74.0
# tr: 400, epochs:  50, learning rate: 0.005; Accuracy (test, 100 instances): 73.0
# tr: 400, epochs:  50, learning rate: 0.010; Accuracy (test, 100 instances): 68.0
# tr: 400, epochs:  50, learning rate: 0.010; Accuracy (test, 100 instances): 68.0
# tr: 400, epochs:  50, learning rate: 0.010; Accuracy (test, 100 instances): 68.0
# tr: 400, epochs:  50, learning rate: 0.010; Accuracy (test, 100 instances): 74.0
# tr: 400, epochs:  50, learning rate: 0.010; Accuracy (test, 100 instances): 78.0
# tr: 400, epochs:  50, learning rate: 0.010; Accuracy (test, 100 instances): 77.0
# tr: 400, epochs:  50, learning rate: 0.050; Accuracy (test, 100 instances): 68.0
# tr: 400, epochs:  50, learning rate: 0.050; Accuracy (test, 100 instances): 71.0
# tr: 400, epochs:  50, learning rate: 0.050; Accuracy (test, 100 instances): 74.0
# tr: 400, epochs:  50, learning rate: 0.050; Accuracy (test, 100 instances): 78.0
# tr: 400, epochs:  50, learning rate: 0.050; Accuracy (test, 100 instances): 78.0
# tr: 400, epochs:  50, learning rate: 0.050; Accuracy (test, 100 instances): 80.0
# tr: 400, epochs: 100, learning rate: 0.005; Accuracy (test, 100 instances): 68.0
# tr: 400, epochs: 100, learning rate: 0.005; Accuracy (test, 100 instances): 68.0
# tr: 400, epochs: 100, learning rate: 0.005; Accuracy (test, 100 instances): 68.0
# tr: 400, epochs: 100, learning rate: 0.005; Accuracy (test, 100 instances): 74.0
# tr: 400, 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: 400, epochs: 100, learning rate: 0.010; Accuracy (test, 100 instances): 68.0
# tr: 400, epochs: 100, learning rate: 0.010; Accuracy (test, 100 instances): 68.0
# tr: 400, epochs: 100, learning rate: 0.010; Accuracy (test, 100 instances): 71.0
```

```
# tr: 400, epochs: 100, learning rate: 0.010; Accuracy (test, 100 instances): 78.0
# tr: 400, 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: 400, epochs: 100, learning rate: 0.050; Accuracy (test, 100 instances): 64.0
# tr: 400, epochs: 100, learning rate: 0.050; Accuracy (test, 100 instances): 69.0
# tr: 400, epochs: 100, learning rate: 0.050; Accuracy (test, 100 instances): 77.0
# tr: 400, epochs: 100, learning rate: 0.050; Accuracy (test, 100 instances): 76.0
# tr: 400, 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:

- Do you need to train with all the training dataset to get the highest accuracy with the test dataset?
- 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
```

- Can you get higher accuracy with additional hyperparameters (higher than `80.0`)?
- Is it always worth training for more epochs (while keeping all other hyperparameters fixed)?

TODO Add your answer here (code and text)

ANSWER :

```
1. No. Training with all the training dataset doesn't guarentee highest accuracy. The
 accuracy indeed depends on the quality of test dataset.

3. No. Sometimes lower accuracy is observed with additional hyperparameters.

4. Epochs become stagnant after a while and we have to change other parameters to get
 different results.
```

3s completed at 11:36 PM