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

```
--2023-02-18 05:10:06--  http://huang.eng.unt.edu/CSCE-5218/train.dat
Resolving huang.eng.unt.edu (huang.eng.unt.edu)... 129.120.123.155
Connecting to huang.eng.unt.edu (huang.eng.unt.edu)|129.120.123.155|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 11244 (11K)
Saving to: ‘train.dat.7’

train.dat.7         100%[===================>]  10.98K  --.-KB/s    in 0s

2023-02-18 05:10:06 (171 MB/s) - ‘train.dat.7’ saved [11244/11244]

--2023-02-18 05:10:06--  http://huang.eng.unt.edu/CSCE-5218/test.dat
Resolving huang.eng.unt.edu (huang.eng.unt.edu)... 129.120.123.155
Connecting to huang.eng.unt.edu (huang.eng.unt.edu)|129.120.123.155|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2844 (2.8K)
Saving to: ‘test.dat.7’

test.dat.7          100%[===================>]   2.78K  --.-KB/s    in 0s

2023-02-18 05:10:06 (285 MB/s) - ‘test.dat.7’ 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     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
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
A1      A2      A3      A4      A5      A6      A7      A8      A9      A10     A11     A12     A13
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
```

CSCE 5218 / CSCE 4930 Deep Learning

HW1a The Perceptron (20 pt)

```
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):
   return sum([array2[val]*array1[val]  for val in range(len(array1))])
```

```
def sigmoid(p):
    a=math.exp(-p)+1
    return 1 / a


# 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):
    val = dot_product(weight, instance)
    z=sigmoid(val)
    return z


# 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(w, i):
  a=output(w, i)
  if  a >= 0.5:
    return 1
  else:
    return 0


def get_accuracy(weights, instances):
    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):

    # Initialize the weights to 0
    weights = [0] * (len(instances[0])-1)

    for _ in range(epochs):
        for instance in instances:
          in_value = dot_product(weights, instance) # we calculate the input value
          output_value = sigmoid(in_value) # we calculate the output value
          error = instance[-1] - output_value # we calculate the error
          for i in range(0, len(weights)): # we update the weights
            weights[i] += lr * error * output_value * (1-output_value) * instance[i]
    return weights
```

Double-click (or enter) to edit

```
# 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
```

```
--2023-02-18 05:10:06--  http://www.cse.unt.edu/~blanco/csce5218/hw1a/train.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... connected.
HTTP request sent, awaiting response... 200 OK
Length: 11244 (11K) [application/x-ns-proxy-autoconfig]
Saving to: ‘train.dat.8’

train.dat.8         100%[===================>]  10.98K  --.-KB/s    in 0s

2023-02-18 05:10:06 (169 MB/s) - ‘train.dat.8’ saved [11244/11244]

--2023-02-18 05:10:06--  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... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2844 (2.8K) [application/x-ns-proxy-autoconfig]
Saving to: ‘test.dat.8’
```

```
    test.dat.8          100%[===================>]   2.78K  --.-KB/s    in 0s

    2023-02-18 05:10:06 (244 MB/s) - ‘test.dat.8’ saved [2844/2844]
```

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

improv_instances_tr=[]
st=0
end=len(instances_tr)
for x in range(st,end-1):
  if(x%2==1):
    improv_instances_tr.append(instances_tr[x])
improv_instances_te=[]
st=0
end=len(instances_te)
for x in range(st,end-1):
  if(x%2==1):
    improv_instances_te.append(instances_te[x])

lr = 0.005
epochs = 5
weights = train_perceptron(improv_instances_tr, lr, epochs)

accuracy = get_accuracy(weights, improv_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: 800, epochs:   5, learning rate: 0.005; Accuracy (test, 200 instances): 67.7
```

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)

In the train_perceptron function, we use the dot_product and sigmoid functions because they are standard techniques for implementing a perceptron model. The dot_product function computes the weighted sum of the input features, and the resulting value is passed through the sigmoid function to generate a prediction value between 0 and 1.

While the predict function you proposed could calculate the weighted sum of the input features, it doesn't include an activation function like the sigmoid function. As a result, the output of the predict function wouldn't be in the desired range of 0 to 1, which is essential for binary classification tasks.

Therefore, to ensure that the perceptron produces an output in the appropriate range for binary classification, using the sigmoid function is essential.

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)

```
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
lr_array = [0.005, 0.01, 0.05]         # learning rate

improv_instances_tr=[]
st=0
end=len(instances_tr)
for x in range(st,end-1):
  if(x%2==1):
    improv_instances_tr.append(instances_tr[x])
improv_instances_te=[]
st=0
end=len(instances_te)
for x in range(st,end-1):
  if(x%2==1):
    improv_instances_te.append(instances_te[x])


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

```
#tr: 20, epochs: 100, learning rate: 0.005; Accuracy (test, 99 instances): 68.7
#tr: 40, epochs: 100, learning rate: 0.005; Accuracy (test, 99 instances): 67.7
#tr: 100, epochs: 100, learning rate: 0.005; Accuracy (test, 99 instances): 67.7
#tr: 200, epochs: 100, learning rate: 0.005; Accuracy (test, 99 instances): 70.7
#tr: 299, epochs: 100, learning rate: 0.005; Accuracy (test, 99 instances): 76.8
#tr: 399, epochs: 100, learning rate: 0.005; Accuracy (test, 99 instances): 75.8
#tr: 20, epochs: 100, learning rate: 0.010; Accuracy (test, 99 instances): 67.7
#tr: 40, epochs: 100, learning rate: 0.010; Accuracy (test, 99 instances): 67.7
#tr: 100, epochs: 100, learning rate: 0.010; Accuracy (test, 99 instances): 69.7
#tr: 200, epochs: 100, learning rate: 0.010; Accuracy (test, 99 instances): 77.8
#tr: 299, epochs: 100, learning rate: 0.010; Accuracy (test, 99 instances): 78.8
#tr: 399, epochs: 100, learning rate: 0.010; Accuracy (test, 99 instances): 78.8
#tr: 20, epochs: 100, learning rate: 0.050; Accuracy (test, 99 instances): 65.7
#tr: 40, epochs: 100, learning rate: 0.050; Accuracy (test, 99 instances): 68.7
#tr: 100, epochs: 100, learning rate: 0.050; Accuracy (test, 99 instances): 75.8
#tr: 200, epochs: 100, learning rate: 0.050; Accuracy (test, 99 instances): 77.8
#tr: 299, epochs: 100, learning rate: 0.050; Accuracy (test, 99 instances): 76.8
#tr: 399, epochs: 100, learning rate: 0.050; Accuracy (test, 99 instances): 79.8
```

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)

1. No, it is not always necessary to train with all the training dataset to achieve the highest accuracy with the test dataset.

   In fact, using all the training data may not always lead to the best performance. When the training dataset is too large, the model may overfit and memorize the training data instead of learning the underlying patterns that generalize to new, unseen data. This can result in poor performance on the test data, which is a common problem in machine learning.

   Therefore, it is common practice to split the training dataset into training and validation sets, and use the validation set to monitor the model's performance during training. This helps to detect and prevent overfitting, and allows for adjusting hyperparameters and model architecture to improve performance.
2. In the second case, we had a larger data set compared to the first data set but we achieved better performance in 1st case compared to the second test case. Reson of this situation is the learning rate. In the second case, even though we have a bigger data set, the learning rate is too low which results in lower performance.
3. yes we can get higher accuracy more than 80% by tuning additional hyperparameters.
4. Just increasing the number of epochs while maintaining the other hyperparameters would not always be advantageous. For instance, compared to the first experiment with only 20 epochs, the third trial with 100 training examples, 50 epochs, and a learning rate of 0.05 had a lower accuracy of 66%. This implies that training for additional epochs can result in overfitting and a reduction in the model's generalization capability. To enhance the performance of the model, it is essential to identify the ideal number of epochs that results in the best performance on the validation set.

The perceptron algorithm's performance can be significantly impacted by choosing the right hyperparameters, such as the number of training examples, learning rate, and number of epochs.

✓ 7s completed at 11:10 PM