Ex1: Consider the following training data set

<table>
<thead>
<tr>
<th>$X_1$</th>
<th>$X_2$</th>
<th>$d$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>0</td>
</tr>
</tbody>
</table>

To implement this classifier with a perceptron network:
- How many layers?
- How many neurons?
Ex1: Consider the following training data set

<table>
<thead>
<tr>
<th>X1</th>
<th>X2</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>0</td>
</tr>
</tbody>
</table>
Ex1: Consider the following training data set

<table>
<thead>
<tr>
<th>X1</th>
<th>X2</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>0</td>
</tr>
</tbody>
</table>

```python
import numpy as np
training_data = [(np.array([1, 2]), 1),
                 (np.array([-1, 2]), 0),
                 (np.array([0, -1]), 0)]
print(training_data)
```
Ex1: Consider the following training data set

<table>
<thead>
<tr>
<th>X1</th>
<th>X2</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>0</td>
</tr>
</tbody>
</table>

Initialize weights:
\[ W = [1.0, -0.8] \]
Initialize bias:
\[ b = [0] \]

weights = [1.0, -0.8]
bias = [0]

© Mai Elshehaly
Ex1: Consider the following training data set

<table>
<thead>
<tr>
<th>X1</th>
<th>X2</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>0</td>
</tr>
</tbody>
</table>

First training data point:

\[ n = W \cdot x + b = [1, 0, -0.8] \cdot [1, 2] + [0] = -0.6 \]

\[ a = f(n) = f(-0.6) = 0 \]
Ex1: Consider the following training data set

<table>
<thead>
<tr>
<th>X1</th>
<th>X2</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>0</td>
</tr>
</tbody>
</table>

Loop over training data points:
Loop over training data points and calculate network output:

```python
import numpy as np

def step_activation(x):
    if x<0: return 0;
    else: return 1;

training_data = [(np.array([1,2]), 1),
                 (np.array([-1,2]), 0),
                 (np.array([0,-1]), 0)]

weights = [1.0, -0.8]
bias = [0]

for x,d in training_data:
    n = np.dot(weights, x) + bias;
    a = step_activation(n)
    print("{}:{} -> {}").format(x,n,a)
```

Output:

- [1 2]:[-0.6] → 0
- [-1 2]:[-2.6] → 0
- [0 -1]:[0.8] → 1
Perceptron Learning: update weights

```python
import numpy as np
def step_activation(x):
    if x<0: return 0;
    else: return 1;
training_data = [(np.array([1,2]), 1),
                  (np.array([-1,2]), 0),
                  (np.array([0,-1]), 0)]
weights = [1.0, -0.8]
bias = [0]
eta = 0.2
epochs = 2
for i in range(epochs):
    for x, d in training_data:
        n = np.dot(weights, x) + bias
        a = step_activation(n)
        print("{}:{} -> {}".format(x, n, a))
        error = d - a
        weights += eta * error * x
```

Output:

```
[1 2]:[-0.6] -> 0
[-1 2]:[-2.] -> 0
[ 0 -1]:[ 0.4] -> 1
[1 2]:[ 0.8] -> 1
[-1 2]:[-1.6] -> 0
[ 0 -1]:[ 0.2] -> 1
```
Exercise:

• Run the algorithm a few more times, increasing the number of epochs by 1 in every run

• What is the number of epochs at which the network converged (i.e. learned the correct classification)?
Exercise:

• Run the algorithm a few more times, increasing the number of epochs by 1 in every run

• What is the number of epochs at which the network converged (i.e. learned the correct classification)?

• Answer = 4
Don’t forget to update the bias: Does the network converge sooner?

```python
import numpy as np
import random

def step_activation(x):
    if x<0: return 0;
    else: return 1;

training_data = [(np.array([1,2]), 1),
                 (np.array([-1,2]), 0),
                 (np.array([0,-1]), 0)]

weights = [1.0, -0.8]
bias = 0
eta = 0.2
epochs = 3

for i in range(epochs):
    for x,d in training_data:
        n = np.dot(weights, x) + bias;
        a = step_activation(n)
        print("{}:{} -> {}".format(x, n, a))
        error = d - a
        weights += eta * error * x
        bias += eta*error
```

© Mai Elshehaly
How many iterations are needed for convergence?

• Always a finite number of iterations
• Perceptron learning is guaranteed to converge
• The number of iterations is affected by how close the solution decision boundary is to the input vectors:
  • If the points are very close to the boundary, they are hard to classify so the algorithm needs more iterations
  • If the points are far from the boundary, they are easily classified and less iterations are needed
Ex2: Consider the following 3 datasets

- Solve the three simple classification problems shown by drawing a decision boundary. Find weight and bias values that result in single-neuron perceptrons with the chosen decision boundaries.

© Mai Elshehaly
def step_activation(x):
    if x<0: return 0;
    else: return 1;

training_data_a = [(np.array([-1,-1]), 1),
                   (np.array([-1,0]), 1),
                   (np.array([-1,1]), 1),
                   (np.array([0,1]), 1),
                   (np.array([1,0]), 0),
                   (np.array([0,-1]), 0),
                   (np.array([1,-1]), 0),
                   (np.array([1,1]), 0)]

weights = [1.0, -0.8]
bias = 0
eta = 0.2
epochs = 4

for i in range(epochs):
    print("epoch {}".format(i))
    for x,d in training_data_a:
        n = np.dot(weights, x) + bias;
        a = step_activation(n)
        print("{} -> {}".format(x, a))
        error = d - a
        weights += eta * error * x
        bias += eta*error
Exercise: find the number of epochs for convergence for data sets (b) and (c)
Ex3: Consider the following training dataset with 4 desired classes

To implement this classifier with a perceptron network:
- How many layers?
- How many neurons?
Ex3: Consider the following training dataset with 4 desired classes

To implement this classifier with a perceptron network:

• How many layers?
  - Linearly separable $\Rightarrow$ single layer
• How many neurons?
  - 2 decision boundaries $\Rightarrow$ 2 neurons
Exercise:
Trace the perceptron learning by hand then run the code and match your handwritten results with the printed results.

\[
W(0) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad b(0) = \begin{bmatrix} 1 \\ 1 \end{bmatrix}.
\]

Iteration 1:

\[
a = \text{hardlim} \left( W(0)p_1 + b(0) \right) = \text{hardlim} \left( \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right) = \begin{bmatrix} 1 \\ 1 \end{bmatrix},
\]

\[
e = t_1 - a = \begin{bmatrix} 0 \\ 0 \end{bmatrix} - \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \end{bmatrix},
\]

\[
W(1) = W(0) + e p^T_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + \begin{bmatrix} -1 \\ -1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix},
\]

\[
b(1) = b(0) + e = \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \begin{bmatrix} -1 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.
\]
Ex3: final decision boundary
Ex4: Consider the following multi-layer perceptron

\[ a^1 = \text{logsig}(W^1p + b^1) \]

\[ a^2 = \text{purelin}(W^2a^1 + b^2) \]
Initialize a network class with the following weights and biases

\[
W_1^{1}(0) = \begin{bmatrix} -0.27 \\ -0.41 \end{bmatrix}, \quad b_1^{1}(0) = \begin{bmatrix} -0.48 \\ -0.13 \end{bmatrix}, \quad W_2^{2}(0) = \begin{bmatrix} 0.09 & -0.17 \end{bmatrix}, \quad b_2^{2}(0) = \begin{bmatrix} 0.48 \end{bmatrix}.
\]
Initialize the network weights and biases

\[
W^1(0) = \begin{bmatrix} -0.27 \\ -0.41 \end{bmatrix}, \quad b^1(0) = \begin{bmatrix} -0.48 \\ -0.13 \end{bmatrix}, \quad W^2(0) = \begin{bmatrix} 0.09 & -0.17 \end{bmatrix}, \quad b^2(0) = \begin{bmatrix} 0.48 \end{bmatrix}.
\]

```python
import numpy as np
class Network(object):
    def __init__(self):
        w1 = np.array([-0.27, -0.41])
        w1.shape = (2, 1)  # to make it a column
        w2 = np.array([0.09, -0.17])
        self.weights = [w1, w2]
        b1 = np.array([-0.48, -0.13])
        b1.shape = (2, 1)
        b2 = np.array([0.48])
        self.biases = [b1, b2]

n = Network()
```

© Mai Elshehaly
Train the network to approximate the function \[ g(p) = 1 + \sin\left(\frac{\pi}{4}p\right) \text{ for } -2 \leq p \leq 2 \]

Create training data set:

```python
self.biases = [b1, b2]
self.training_data = self.create_training_data_set()

def create_training_data_set(self):
    td = []
    for i in np.arange(-2, 2, 0.2):
        datum = self.g(i)
        td.append(np.array([i, datum]))
    return td

def g(self, p):
    return 1+math.sin(math.pi/4*p)
```

© Mai Elshehaly
Mini Project 1:

• Write the backpropagation code to train the network using the training data set we just generated.

• Print training results at each iteration and verify your results with the following values:

• First layer:

\[
a^1 = f^1(W^1 a^0 + b^1) = \text{logsig}\left(\begin{bmatrix} -0.27 \\ -0.41 \end{bmatrix} + \begin{bmatrix} -0.48 \\ -0.13 \end{bmatrix}\right) = \text{logsig}\left(\begin{bmatrix} -0.75 \\ -0.54 \end{bmatrix}\right)
\]

\[
= \begin{bmatrix}
\frac{1}{1 + e^{-0.75}} \\
\frac{1}{1 + e^{-0.54}}
\end{bmatrix} = \begin{bmatrix} 0.321 \\ 0.368 \end{bmatrix}.
\]
Mini Project 1:

• Second layer:

\[ a^2 = f^2(W^2 \mathbf{a}^1 + \mathbf{b}^2) = \text{purelin} \left( \begin{bmatrix} 0.09 & -0.17 \\ 0.368 \end{bmatrix} \begin{bmatrix} 0.321 \\ 0.48 \end{bmatrix} \right) = \begin{bmatrix} 0.446 \end{bmatrix} \, . \]

• Error:

\[ e = t - a = \left\{ 1 + \sin \left( \frac{\pi}{4} p \right) \right\} - a^2 = \left\{ 1 + \sin \left( \frac{\pi}{4} \right) \right\} - 0.446 = 1.261 \, . \]
Implement derivatives

• The derivatives of the activation functions:

\[ f'(n) = \frac{d}{dn} \left( \frac{1}{1 + e^{-n}} \right) = \frac{e^{-n}}{(1 + e^{-n})^2} = \left( 1 - \frac{1}{1 + e^{-n}} \right) \left( \frac{1}{1 + e^{-n}} \right) = (1 - a^1)(a^1). \]

\[ f''(n) = \frac{d}{dn} (n) = 1. \]
Backpropagation Step

\[ s^2 = -2 \hat{F}^2(n^2)(t - a) = -2 \left[ f^2(n^2) \right](1.261) = -2 \left[ 1 \right](1.261) = -2.522. \]

\[ s^1 = \hat{F}^1(n^1)(W^2)^T s^2 = \begin{bmatrix} (1 - a_1^1)(a_1^1) & 0 \\ 0 & (1 - a_2^1)(a_2^1) \end{bmatrix} \begin{bmatrix} 0.09 \\ -0.17 \end{bmatrix} \begin{bmatrix} -2.522 \end{bmatrix} \]

\[ = \begin{bmatrix} (1 - 0.321)(0.321) & 0 \\ 0 & (1 - 0.368)(0.368) \end{bmatrix} \begin{bmatrix} 0.09 \\ -0.17 \end{bmatrix} \begin{bmatrix} -2.522 \end{bmatrix} \]

\[ = \begin{bmatrix} 0.218 & 0 \\ 0 & 0.233 \end{bmatrix} \begin{bmatrix} -0.227 \\ 0.429 \end{bmatrix} = \begin{bmatrix} -0.0495 \\ 0.0997 \end{bmatrix}. \]
Verify weight update for first iteration

\[
W^2(1) = W^2(0) - \alpha s^2(a^1)^T = \begin{bmatrix} 0.09 & -0.17 \end{bmatrix} - 0.1 \begin{bmatrix} -2.522 \\ 0.321 \end{bmatrix} = \begin{bmatrix} 0.171 \\ -0.0772 \end{bmatrix},
\]

\[
b^2(1) = b^2(0) - \alpha s^2 = \begin{bmatrix} 0.48 \end{bmatrix} - 0.1 \begin{bmatrix} -2.522 \end{bmatrix} = \begin{bmatrix} 0.732 \end{bmatrix},
\]

\[
W^1(1) = W^1(0) - \alpha s^1(a^0)^T = \begin{bmatrix} -0.27 \\ -0.41 \end{bmatrix} - 0.1 \begin{bmatrix} -0.0495 \\ 0.0997 \end{bmatrix} = \begin{bmatrix} -0.265 \\ -0.420 \end{bmatrix},
\]

\[
b^1(1) = b^1(0) - \alpha s^1 = \begin{bmatrix} -0.48 \\ -0.13 \end{bmatrix} - 0.1 \begin{bmatrix} -0.0495 \\ 0.0997 \end{bmatrix} = \begin{bmatrix} -0.475 \\ -0.140 \end{bmatrix}.
\]
Mini project 1: submission

• Code: yourname_filename.py
• A text file containing the output of your print() statements for the different iterations.
Mini Project 2: the Iris dataset

• Use the existing back propagation code (that you have experimented with in the sections) to classify the Iris dataset.

• You can find the Iris data at https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data

• Modify the algorithm parameters and report on different accuracy results
Mini Project 2: submission

• Code yourname_network.py
• A text file describing the output given at least 4 different parameters settings
• Report on the best parameter values for your ANN with this dataset.
Mid term exam review
Project hack session (1 hour)