LAST CLASS

- People are so smart
- We want a computer program that can learn from data
- Artificial Neurons simulate part of the functionality of human brain
- They are connected together to form networks
- Neurons can make a decision
- Neural networks can recognize handwritten characters
- Activation function determines when a neuron is fired
- Step function is not flexible for learning
- Sigmoid function gives a smooth transition
HOMEWORK 1

- **Q1:** draw a neuron (with weights and bias for a step activation function) to implement a NOR gate

- **Q2:** Use multiple neurons to build a neural network that realizes a half adder using NOR only gates (each NOR gate is a neuron).
  - A. Label weights and threshold on the diagram
  - B. How many layers exist in your network?
  - C. How many neurons in each layer?

- **Q3:** Add a fourth layer to the network on slide 52 to transform the outputs (coming from layer 3) into binary format.
Q1: NOR GATE

A

-2

-1

B

-2

\[ \text{NOR} \]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Q2: HALF ADDER:
First let's define what half adder is:
A circuit of logic gates that can add two bits
The output of half adder is a sum bit and a carry bit

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Truth table

1 bit half adder

Schematic

Realization

© Mai Elshehaly
STEP 1: FOR SUM BIT IMPLEMENT XOR USING NOR GATES

- Sum = $\bar{A}.B + A.\bar{B}$
  
  $= (A.B + A.\bar{B})$
  $= ((\bar{A}B). (A\bar{B}))$
  $= ((A + \bar{B}).(\bar{A} + B))$
  $= (A\bar{A} + AB + \bar{A}B + B\bar{B})$
  $= (A.B + \bar{A}.B)$
  $= ((\bar{A} + \bar{B}) + (A + B))$
  $= (((A + A) + (B + B)) + (A + B))$
STEP 2: FOR CARRY BIT IMPLEMENT AND GATE USING NOR GATES

- Carry = A.B

\[
= \overline{A \cdot B}
= (\overline{A} + \overline{B})
= \left( (\overline{A} + A) + (B + \overline{B}) \right)
\]

Half adder using NOR logic
STEP 3: TRANSFORM EACH NOR GATE INTO A NEURON LIKE THE ONE YOU DREW IN Q1

Half adder using NOR logic

3 layers for sum
2 layers for carry
Q3: A NN TO RECOGNIZE HANDWRITTEN NUMBERS

© Mai Elshehaly
EXAMPLE: RECOGNIZING HANDWRITTEN 6

• Suppose the image is made up of 28 x 28 pixels
• Each pixel holds one value = intensity
• We can build a neural network to tell us whether or not the number shown is 6

# neurons in input layer = 28 x 28 = 784
# neurons in output layer = 1 (yes or no)
# neurons in hidden layer(s) depends on the design
A NETWORK FOR ALL 10 DIGITS
THE MODIFIED NETWORK SHOULD LOOK LIKE
Q3: IN THE THIRD (HIDDEN) LAYER, **ONLY ONE** NEURON WILL FIRE. THAT’S THE NEURON THAT HAS THE VALUE DETECTED BY THE NETWORK

- If the input was an image that represents the digit 0 then the first neuron in the hidden layer will output 1. All other neurons in the third layer will output 0.
- If the image was an image that represents the digit 9 then the last neuron in the hidden layer will output 1. All other neurons in the third layer will output 0.
- The same for any given digit: only the neuron that represents this digit will be 1 and all others will be 0.
EXAMPLE: IF THE DETECTED DIGIT WAS 3
Solution: We need to set the weights coming from each digit to change the neurons in the output layer in a manner consistent with this digit’s binary representation.

The weights of the desired digit are the only ones active at this point because all others are not fired from the hidden layer.
EXAMPLE: DIGIT 3
ASSUME BIAS = 0

\[ x_3 \cdot w_{30} = 1 \cdot 0 = 0 \]

\[ x_3 \cdot w_{31} = 1 \cdot 0 = 0 \]

\[ x_3 \cdot w_{32} = 1 \cdot 1 = 1 \]

\[ x_3 \cdot w_{33} = 1 \cdot 1 = 1 \]
© Mai Elshehaly
QUESTIONS ABOUT HW1

- Course website: [www.vagua.org/mai/neural/](http://www.vagua.org/mai/neural/)

- Email me: [maya70@vt.edu](mailto:maya70@vt.edu)

- Email Eng. Noura: [nouramohammed29@yahoo.com](mailto:nouramohammed29@yahoo.com)

- Office Hours: Mon. and Thu. 12:00 – 1:00

- OR: Start a discussion on [Moodle](https://moodle.vagua.org)
Moodle

- A Learning Management System (LMS)
- All class communication will happen on it
- Homeworks and class news
- Go to: www.vaqua.org/mai/neural/
STEP 1: GO TO WWW.VAQUA.ORG/MAI/MOODLE THEN CLICK “ARTIFICIAL NEURAL NETWORKS”
STEP 2: SELF ENROLMENT

Enrolment options

Introduction to Computers

Have you ever wondered how computers work? how they store and retrieve information? and how we can program them to solve our problems? This intro class will walk you through the components of a computer system, basic operations, and a programming primer.

DISCLAIMER: This course is a university requirement instituted by the Suez Canal University. The course syllabus is unified across all disciplines and MUST be covered. This material may be a little too basic for Computer Science majors so I will try to make it a little more challenging and hopefully interesting. I will do this within strict limitations in order to conform to university requirements.

Self enrolment (Student)

Guests cannot access this course. Please log in.

Continue
STEP 3: CREATE NEW ACCOUNT

Dr. Mai Elshehaly's classes

Username / email
Password

Remember username

Log in

Forgotten your username or password?
Cookies must be enabled in your browser

Some courses may allow guest access

Log in as a guest

Is this your first time here?

For full access to this site, you first need to create an account.

Create new account

You are currently using guest access

Home
STEP 4: ENTER YOUR ACCOUNT INFORMATION
Choose your username and password

Username: 
mimi

The password must have at least 8 characters, at least 1 digit(s), at least 1 lower case letter(s), at least 1 upper case letter(s), at least 1 non-alphanumeric character(s) such as *, -, or #.

Password: 

More details

Email address: 
mai@umbc.edu

Email (again): 
mai@umbc.edu

First name: 
Maya

Surname: 
El

City/town: 

Country: 
Select a country

Create my new account | Cancel

There are required fields in this form marked *.
MAKE SURE YOUR PASSWORD MEETS THE REQUIREMENTS

Dr. Mai Elshehaly's classes

New account

Choose your username and password

Username: mimi

The password must have at least 8 characters, at least 1 digit(s), at least 1 lower case letter(s), at least 1 upper case letter(s), at least 1 non-alphanumeric character(s) such as *, -, or #

Password: 

Passwords must have at least 1 digit(s).
Passwords must have at least 1 upper case letter(s).
Passwords must have at least 1 non-alphanumeric character(s) such as *, -, or #.

More details

Email address: mail@umbc.edu

Email (again): mail@umbc.edu

First name: Maya
STEP 5: CHECK YOUR EMAIL FOR CONFIRMATION MESSAGE AND CLICK THE LINK IN THE EMAIL
STEP 6: ENTER ENROLMENT KEY: FCI2017
STEP 7: ENROLL ME

Enrolment options

Artificial Neural Networks

Have you ever wondered how computers work? how they store and retrieve information? and how we can program them to solve our problems? This intro class will walk you through the components of a computer system, basic operations, and a programming primer.

DISCLAIMER: This course is a university requirement instated by the Susz Canal University. The course syllabus is unified across all disciplines and MUST be covered. This materials may be a little too basic for Computer Science majors so I will try to make it a little more challenging and hopefully interesting. I will do this within strict limitations in order to conform to university requirements.

Self enrolment (Student)

Enrolment key

FCI2017

Press enter to save changes

Enrol me
TODAY’S CLASS

- Perceptron
- Decision Boundary
- Cost Function
- Supervised Learning
- Gradient Descent
- Least Mean Squares (LMS)
- Remember Amr Diab?
  - Amr Diab concert in Ismailia! Should you go or not?
    - $x_1$: Is the weather good?
    - $x_2$: Is the ticket affordable?
    - $x_3$: Does your best friend want to go with you?

\[
\text{output} = \begin{cases} 
0 & \text{if } \sum_j w_j x_j \leq \text{threshold} \\
1 & \text{if } \sum_j w_j x_j > \text{threshold}
\end{cases}
\]
ANATOMY OF A PERCEPTRON

$\sum_{i=1}^{n} w_i x_i + b$

Inputs | Weights | Neuron

Net input | Activation function | Output

Bias
SINGLE NEURON PERCEPTRON

\[ \text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases} \]

\[ a = f(n) = f(w \cdot x + b) \]
EXAMPLE: A FRUIT CLASSIFIER

EXAMPLE: A FRUIT CLASSIFIER

- Two types of fruit: apples and oranges
- Sensors can measure a number of **features**
- The **features** act as **inputs** to the neural network
- The neural network decides whether the fruit is an apple or an orange
- Possible features:
  - Length
  - Weight
  - Roundness
  - Texture
Let's consider 2 features for input: length and weight
### Training Set

<table>
<thead>
<tr>
<th>Fruit class</th>
<th>Weight (grams)</th>
<th>Length (cm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apple (C1)</td>
<td>121</td>
<td>16.8</td>
</tr>
<tr>
<td>Orange (C2)</td>
<td>210</td>
<td>9.4</td>
</tr>
<tr>
<td>Apple (C1)</td>
<td>114</td>
<td>15.2</td>
</tr>
<tr>
<td>Orange (C2)</td>
<td>195</td>
<td>8.1</td>
</tr>
</tbody>
</table>
## TRAINING SET

<table>
<thead>
<tr>
<th>Fruit class</th>
<th>Weight (grams)</th>
<th>Length (cm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apple (C1)</td>
<td>121</td>
<td>16.8</td>
</tr>
<tr>
<td>Orange (C2)</td>
<td>210</td>
<td>9.4</td>
</tr>
<tr>
<td>Apple (C1)</td>
<td>114</td>
<td>15.2</td>
</tr>
<tr>
<td>Orange (C2)</td>
<td>195</td>
<td>8.1</td>
</tr>
</tbody>
</table>

The data above is plotted on a graph showing a linear relationship between weight and length, indicating that the classes are linearly separable.
# TRAINING SET

<table>
<thead>
<tr>
<th>Fruit class</th>
<th>Weight (grams)</th>
<th>Length (cm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apple (C1)</td>
<td>121</td>
<td>16.8</td>
</tr>
<tr>
<td>Orange (C2)</td>
<td>210</td>
<td>9.4</td>
</tr>
<tr>
<td>Apple (C1)</td>
<td>114</td>
<td>15.2</td>
</tr>
<tr>
<td>Orange (C2)</td>
<td>195</td>
<td>8.1</td>
</tr>
</tbody>
</table>

Which line? Depends on the weights.
Let’s say we have:
- Initial weight vector: \( w = [w_1, w_2] = [-30, 300] \)
- Bias: \( b = -1200 \)
**STEP 2: DECISION BOUNDARY**

\[ a = f(\mathbf{n}) = f(\mathbf{Wx} + \mathbf{b}) = \sum w_i x_i + b \]
\[ a = -30 \, x_1 + 300 \, x_2 - 1200 \]

Set \( a = 0 \)

- Set \( x_1 = 100 \)
  \[ x_2 = \frac{30 \, x_1 + 1200}{300} = \frac{3000 + 1200}{300} = 14 \]
- Set \( x_1 = 200 \)
  \[ x_2 = \frac{30 \, x_1 + 1200}{300} = \frac{6000 + 1200}{300} = 24 \]

Is this a good decision boundary?
**STEP 3: ACTIVATION FUNCTION**

- \( f(.) = \) Symmetrical Hard Limit
  - \( a = -1 \) \( n < 0 \)
  - \( a = +1 \) \( n \geq 0 \)

Orange

Apple
## Step 4: Substitute Training Data

<table>
<thead>
<tr>
<th>Fruit class</th>
<th>Weight (grams)</th>
<th>Length (cm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apple (C1)</td>
<td>121</td>
<td>16.8</td>
</tr>
<tr>
<td>Orange (C2)</td>
<td>210</td>
<td>9.4</td>
</tr>
<tr>
<td>Apple (C1)</td>
<td>114</td>
<td>15.2</td>
</tr>
<tr>
<td>Orange (C2)</td>
<td>195</td>
<td>8.1</td>
</tr>
</tbody>
</table>

\[
n = -30x_1 + 300x_2 - 1200 \\
= -30 \times 121 + 300 \times 16.8 - 1200 = 31,470 > 0 \quad a = f(n) = 1
\]

Correctly classified an apple so no weight adjustment needed at this training step.
STEP 4: SUBSTITUTE TRAINING DATA

<table>
<thead>
<tr>
<th>Fruit class</th>
<th>Weight (grams)</th>
<th>Length (cm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apple (C1)</td>
<td>121</td>
<td>16.8</td>
</tr>
<tr>
<td>Orange (C2)</td>
<td>210</td>
<td>9.4</td>
</tr>
<tr>
<td>Apple (C1)</td>
<td>114</td>
<td>15.2</td>
</tr>
<tr>
<td>Orange (C2)</td>
<td>195</td>
<td>8.1</td>
</tr>
</tbody>
</table>

\[ n = -30x_1 + 300x_2 - 1200 \]
\[ = -30 \times 210 + 300 \times 9.4 - 1200 = -4,680 < 0 \]

\[ a = f(n) = -1 \]

Correctly classified an orange so no weight adjustment needed at this training step
## STEP 4: SUBSTITUTE TRAINING DATA

<table>
<thead>
<tr>
<th>Fruit class</th>
<th>Weight (grams)</th>
<th>Length (cm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apple (C1)</td>
<td>121</td>
<td>16.8</td>
</tr>
<tr>
<td>Orange (C2)</td>
<td>210</td>
<td>9.4</td>
</tr>
<tr>
<td>Apple (C1)</td>
<td>114</td>
<td>15.2</td>
</tr>
<tr>
<td>Orange (C2)</td>
<td>195</td>
<td>8.1</td>
</tr>
</tbody>
</table>

\[
n = -30 \, x_1 + 300 \, x_2 - 1200 \\
= -30 \times 114 + 300 \times 15.2 - 1200 = -60 < 0 \\
\]

\[
a = f(n) = -1
\]

Wrong! The classifier decided that this is an orange, when in fact it is an apple.
WEIGHT ADJUSTMENT

\[ a = f(n) = f(Wx + b) = \sum w_i x_i + b \]
\[ a = -30 x_1 + 300 x_2 - 1200 \]

The weights need to be adjusted slowly until we reach a better decision boundary that clearly separates the two classes.
**WEIGHT ADJUSTMENT**

\[ a = f(n) = f(Wx + b) = \sum w_i x_i + b \]
\[ a = -30x_1 + 300x_2 - 1200 \]

A large adjustment step can create another bad decision boundary
WEIGHT ADJUSTMENT

\[ a = f(n) = f(Wx + b) = \sum w_i x_i + b \]
\[ a = -30 x_1 + 300 x_2 - 1200 \]

An adjustment in the opposite direction will only make things worse.
SUPervised Learning

- In the training dataset we know what the desired outcome is.
- In the above example, we knew which fruits are apples and which are oranges.
- This information is used to decide when and **how** the weights of the ANN need to be adjusted in order to create a better decision boundary.
- The learning process needs to be gradual with small steps.
HOW TO ADJUST

learning step = \( w(t + 1) - w(t) = \eta(d - a)x \)

\[
\begin{align*}
w(t + 1) &= w(t) + \eta(d - a)x \\
w(0) &= [-30, 300, -1200]^T \\
x &= [114, 15.2, +1]^T
\end{align*}
\]

\[
w(1) = [-30, 300, -1200]^T + 0.01 \times (1 - (-1)) \times [114, 15.2, +1]^T
\]

\[
w(1) = [-27.72, 300.304, -1199.98]
\]

\[
n = -27.72 \, x_1 + 300.304 \, x_2 - 1199.98
\]

\[
= -27.72 \times 114 + 300.304 \times 15.2 - 1199.98 = 204.92 > 0
\]

\[
a = f(n) = 1 \rightarrow \text{Correctly classified as apple}
\]
LEARNING ACCOMPLISHED

- We only needed one iteration for this example
- Usually the program will iterate through a number of steps to reach an appropriate decision boundary that correctly classifies all the samples in the training set.
The operations we discussed so far are those of an adaptive filter.

They consist of 2 main processes:

1. Filtering process: involves the computation of the actual output $a$ and an error term $e$ which is the difference between $a$ and the desired output $d$.
2. Adaptive process: the automatic adjustment of synaptic weights of the neuron according to the error signal $e$. 
COST FUNCTION

- Determines the manner in which the error signal $e$ controls the adjustments to the neuron’s synaptic weights
- We can define the cost function to be a function that tells us how large is the error signal $e$
- More error (misclassified samples) means the classifier is not doing a good job and weights need to be adjusted
- Closely related to optimization problems
UNCONSTRAINED OPTIMIZATION

- We want to modify the weights $\mathbf{w}$ in a way that minimizes the cost function.
- We want to find an optimal solution $\mathbf{w}^*$ that satisfies:

$$\mathcal{E}(\mathbf{w}^*) \leq \mathcal{E}(\mathbf{w}) \quad (3.5)$$

That is, we need to solve an *unconstrained optimization problem*, stated as follows:

*Minimize the cost function $\mathcal{E}(\mathbf{w})$ with respect to the weight vector $\mathbf{w}$* \quad (3.6)

The necessary condition for optimality is

$$\nabla \mathcal{E}(\mathbf{w}^*) = \mathbf{0} \quad (3.7)$$
GRADIENT

\[ y = x^2 - 2 \]

- **Decreasing (negative) gradient**
- **Increasing (positive) gradient**
- **Zero gradient**
ITERATIVE DESCENT

Starting with an initial guess denoted by $w(0)$, generate a sequence of weight vectors $w(1)$, $w(2)$, ..., such that the cost function $\mathcal{E}(w)$ is reduced at each iteration of the algorithm, as shown by

$$\mathcal{E}(w(n + 1)) < \mathcal{E}(w(n))$$

(3.10)

where $w(n)$ is the old value of the weight vector and $w(n + 1)$ is its updated value.
The adjustments made to the weights vector are in the direction of steepest descent of the cost function.

That is the direction opposite to $\nabla C(w)$.

Define the gradient vector: $g = \nabla C(w)$ \hspace{1cm} (3.11)

Steepest descent: $w(n + 1) = w(n) - \eta g(n)$ \hspace{1cm} (3.12)
METHOD OF STEEPEST DESCENT

- In going from iteration $n$ to iteration $n+1$ the algorithm applies the correction:

$$
\Delta w(n) = w(n+1) - w(n)
= -\eta g(n)
$$

(3.13)
TYPES OF COST FUNCTION:
(I) LEAST MEAN SQUARES (LMS)

\[ \mathcal{E}(w) = \frac{1}{2} e^2(n) \]  

(3.33)
(I) LEAST MEAN SQUARES (LMS)

\[ e(n) = d(n) - [x(1), x(2), \ldots, x(n)]^T w(n) \]
\[ = d(n) - X(n)w(n) \quad (3.25) \]

where \( d(n) \) is the \( n \)-by-1 desired response vector:

\[ d(n) = [d(1), d(2), \ldots, d(n)]^T \]

and \( X(n) \) is the \( n \)-by-\( m \) data matrix:

\[ X(n) = [x(n), x(2), \ldots, x(n)]^T \]
(I) LEAST MEAN SQUARES (LMS)

- Rate of change: Differentiating the cost function w.r.t. the weights vector $w$ yields:

$$\frac{\partial C(w)}{\partial w} = e(n) \frac{\partial e(n)}{\partial w}$$

(3.34)
(I) LEAST MEAN SQUARES (LMS)

Recall: \[ e(n) = d(n) - x^T(n)w(n) \]

Hence: \[ \frac{\partial e(n)}{\partial w(n)} = -x(n) \]

and: \[ \frac{\partial \mathcal{E}(w)}{\partial w} = e(n) \frac{\partial e(n)}{\partial w} \]

\[ \frac{\partial \mathcal{E}(w)}{\partial w(n)} = -x(n)e(n) \]
(I) LEAST MEAN SQUARES

- We use the latter as an estimate for the gradient

\[ \hat{g}(n) = -x(n)e(n) \]  \hspace{1cm} (3.36)

- Substitute in the equation of the steepest descent (3.12):

\[ \hat{w}(n + 1) = \hat{w}(n) + \eta x(n)e(n) \]  \hspace{1cm} (3.37)
RECAP: HOW TO ADJUST

\[ \text{learning step} = \mathbf{w}(t + 1) - \mathbf{w}(t) = \eta(d - a)x \]

\[
\begin{align*}
\mathbf{w}(t + 1) &= \mathbf{w}(t) + \eta(d - a)x \\
\mathbf{w}(0) &= [-30, 300, -1200]^T \\
x &= [114, 15.2, +1]^T
\end{align*}
\]

\[
\begin{align*}
\mathbf{w}(1) &= [-30, 300, -1200]^T + 0.01 \times (1 - (-1)) \times [114, 15.2, +1]^T \\
\mathbf{w}(1) &= [-27.72, 300.304, -1199.98]
\end{align*}
\]

\[
\begin{align*}
n &= -27.72 \times x_1 + 300.304 \times x_2 - 1199.98 \\
&= -27.72 \times 114 + 300.304 \times 15.2 - 1199.98 = 204.92 > 0
\end{align*}
\]

\[ a = f(n) = 1 \rightarrow \text{Correctly classified as apple} \]
LET'S CODE

- Getting started tutorial (optional): https://confluence.jetbrains.com/display/PYH/Getting+Started+with+PyCharm
- Perceptron code: https://datasciencelab.wordpress.com/2014/01/10/machine-learning-classics-the-perceptron/
PERCEPTRON LEARNING ALGORITHM

```python
import numpy as np
import random
import matplotlib.pyplot as plt

class Perceptron:
    def __init__(self, N):
        # Random linearly separated data
        xA, yA, xB, yB = [random.uniform(-1, 1) for i in range(4)]
        self.V = np.array([xB * yA - xA * yB, yB - yA, xA - xB])
        self.X = self.generate_points(N)

    def generate_points(self, N):
        X = []
        for i in range(N):
            x1, x2 = [random.uniform(-1, 1) for i in range(2)]
            x = np.array([1, x1, x2])
            s = int(np.sign(self.V.T.dot(x)))
            X.append((x, s))
        return X
```

Add this to your code

© Mai Elshehaly
def plot(self, mispts=None, vec=None, save=False):
    fig = plt.figure(figsize=(5, 5))
    plt.xlim(-1, 1)
    plt.ylim(-1, 1)
    V = self.V
    l = np.linspace(-1, 1)
    plt.plot(l, a * l + b, 'k-')
    cols = {1: 'r', -1: 'b'}
    for x, s in self.X:
        plt.plot(x[1], x[2], cols[s] + 'o')
    if mispts:
        for x, s in mispts:
            plt.plot(x[1], x[2], cols[s] + '.')
    if vec != None:
        plt.plot(l, aa * l + bb, 'g-', lw=2)
    if save:
        if not mispts:
            plt.title('N = %s' % (str(len(self.X))))
        else:
            plt.title('N = %s with %s test points' % (str(len(self.X)), str(len(mispts))))
    plt.savefig('p_N%s' % (str(len(self.X))), bbox_inches='tight', dpi=200)
    plt.show(block=True)
```python
def classification_error(self, vec, pts=None):
    # Error defined as fraction of misclassified points
    if not pts:
        pts = self.X
    M = len(pts)
    n_mispts = 0
    for x, s in pts:
        if int(np.sign(vec.T.dot(x))) != s:
            n_mispts += 1
    error = n_mispts / float(M)
    return error

def choose_miscl_point(self, vec):
    # Choose a random point among the misclassified
    pts = self.X
    mispts = []
    for x, s in pts:
        if int(np.sign(vec.T.dot(x))) != s:
            mispts.append((x, s))
    return mispts[random.randrange(0, len(mispts))]
```
```python
def pla(self, save=False):
    # Initialize the weights to zeros
    w = np.zeros(3)
    X, N = self.X, len(self.X)
    it = 0
    # Iterate until all points are correctly classified
    while self.classification_error(w) != 0:
        it += 1
        # Pick random misclassified point
        x, s = self.choose_miscl_point(w)
        # Update weights
        w += s * x
        if save:
            self.plot(vec=w)
            plt.title('N = %s, Iteration %s
            % (str(N), str(it)))
            plt.savefig('p_N%s_it%s' % (str(N), str(it)),
                        dpi=200, bbox_inches='tight')
    self.w = w

def check_error(self, M, vec):
    check_pts = self.generate_points(M)
    return self.classification_error(vec, pts=check_pts)
```
CALL

95  p = Perceptron(20)
96  p.plot()
Reimplement the perceptron class in the example given in this class to use the Mean Least Squares (MLS) cost function and the steepest descent adjustment algorithm.

### TABLE 3.1 Summary of the LMS Algorithm

<table>
<thead>
<tr>
<th>Training Sample:</th>
<th>Input signal vector = x(n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Desired response</td>
<td>d(n)</td>
</tr>
</tbody>
</table>

User-selected parameter: \( \eta \)

**Initialization.** Set \( \hat{w}(0) = 0 \).

**Computation.** For \( n = 1, 2, \ldots \), compute

\[
e(n) = d(n) - \hat{w}^T(n)x(n)
\]

\[
\hat{w}(n + 1) = \hat{w}(n) + \eta x(n)e(n)
\]
LAB WORK (SECTION) WILL COVER

- Details of the code example
- Assistance for PyCharm installation
- Questions on steepest descent and LMS
- How to submit HW on Moodle
SEE YOU NEXT WEEK