92586 Computational Linguistics

8. “One” Neuron

Alberto Barrón-Cedeño

Alma Mater Studiorum-Università di Bologna
a.barron@unibo.it @_albarron_

31/03/2020

Previously

- From Words to Topics

Table of Contents

Chapter 5 of Lane et al. (2019)

There was Life Before Deep Learning

Title inspired by a speech by Hermann Ney

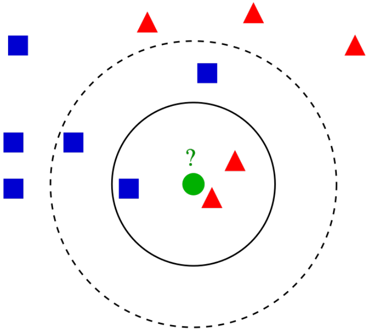
There Was Life Before Deep Learning

(And Many Non-NN in-Production Models Prevail)

- Naïve Bayes
- *k*-nearest neighbors
- Random forests
- Support vector machines
- HMM
- Logistic Regression
- ...

There Was Life Before Deep Learning

k-Nearest Neighbours



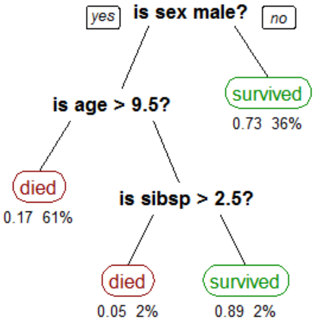
The class of • is the same as the most frequent among its *k* neighbours

https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

There Was Life Before Deep Learning

Random Forests

Titanic survivors

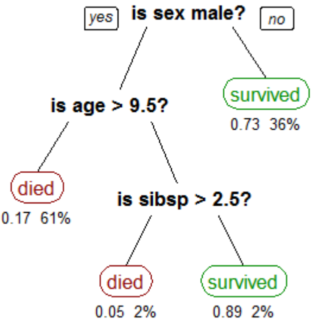


https://en.wikipedia.org/wiki/Random_forest

There Was Life Before Deep Learning

Random Forests

Titanic survivors

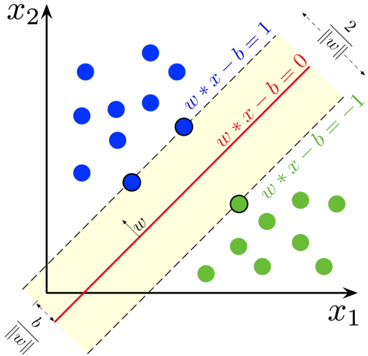


Multiple decision trees are learned and the final class is the **mode**

https://en.wikipedia.org/wiki/Random_forest

There Was Life Before Deep Learning

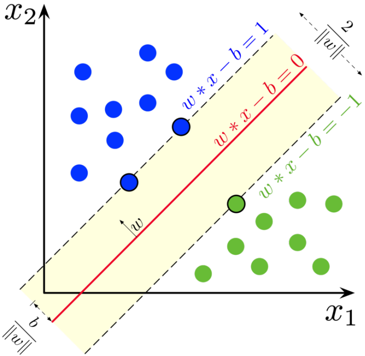
Support Vector Machines



https://en.wikipedia.org/wiki/Support-vector_machine

There Was Life Before Deep Learning

Support Vector Machines



Kernels

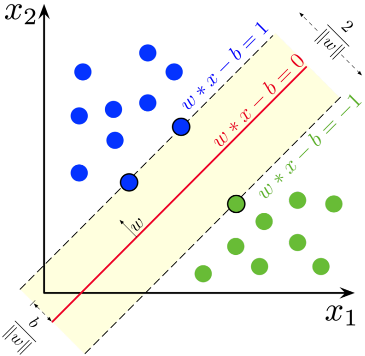
- Linear
- Polynomial
- RBF
- Tree

---

https://en.wikipedia.org/wiki/Support-vector_machine

There Was Life Before Deep Learning

Support Vector Machines



Kernels

- Linear
- Polynomial
- RBF
- Tree

---

https://en.wikipedia.org/wiki/Support-vector_machine

- SVM^{HMM} for sequences[a]
- SVM-Rank for ranking
- SVR for regression

---

[a] Also HMM

There Was Life Before Deep Learning

There are many, many others

There Was Life Before Deep Learning

There are many, many others

- Often they are SoA (or close)

There Was Life Before Deep Learning

There are many, many others

- Often they are SoA (or close)
- In general, they are *cheaper*

There Was Life Before Deep Learning

There are many, many others

- Often they are SoA (or close)
- In general, they are *cheaper*
- In general, they require *less* data

There Was Life Before Deep Learning

There are many, many others

- Often they are SoA (or close)
- In general, they are *cheaper*
- In general, they require *less* data
- Some of them give *explainable* outcomes

There Was Life Before Deep Learning

There are many, many others

- Often they are SoA (or close)
- In general, they are *cheaper*
- In general, they require *less* data
- Some of them give *explainable* outcomes
- Representations have to be *engineered*

Some History

Some History

Opening paragraph of Rosenblatt (1957)'s **The Perceptron—a perceiving and recognizing automaton**

Since the advent of electronic computers and modern servo systems, an increasing amount of attention has been focused on the feasibility of constructing a device possessing such human-like functions as perception, recognition, concept formation, and the ability to generalize from experience. In particular, interest has centered on the idea of a machine which would be capable of conceptualizing inputs impinging directly from the physical environment of light, sound, temperature, etc. -- the "phenomenal world" with which we are all familiar -- rather than requiring the intervention of a human agent to digest and code the necessary information.

AI Winters

1974–1980	First major winter
1987–1993	Second major winter

https://en.wikipedia.org/wiki/AI_winter

AI Winters

1974–1980	First major winter
1987–1993	Second major winter

1966	failure of MT
1970	abandonment of connectionism
1971–75	DARPA's frustration wrt CMU speech recognition research
1973	Lighthill report decreases AI research in the UK
1973–74	: DARPA's cutbacks to academic AI research

https://en.wikipedia.org/wiki/AI_winter

AI Winters

1974–1980	First major winter
1987–1993	Second major winter

1966	failure of MT
1970	abandonment of connectionism
1971–75	DARPA's frustration wrt CMU speech recognition research
1973	Lighthill report decreases AI research in the UK
1973–74	: DARPA's cutbacks to academic AI research
1987	collapse of the LISP machine market
1988	cancellation of new spending on AI by the Strategic Computing Initiative
1993	resistance to expert systems deployment and maintenance
1990s	end of the Fifth Generation computer project's original goals

https://en.wikipedia.org/wiki/AI_winter

The Perceptron

The Perceptron

- Intended to be a machine able of recognising images

The Perceptron

- Intended to be a machine able of recognising images
- Rough idea:

Input:	features of an image (small subsections)
Parameters:	weights for each feature (measure of importance)
Output:	Fire once all potentiometers pass a certain threshold

The Perceptron

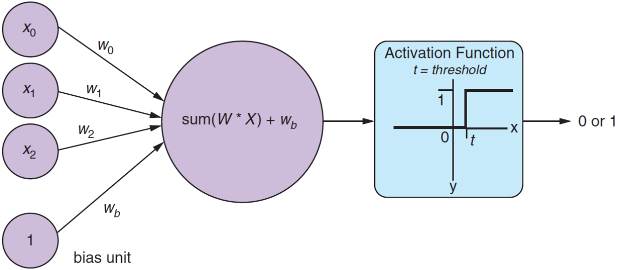
- Intended to be a machine able of recognising images
- Rough idea:

Input:	features of an image (small subsections)
Parameters:	weights for each feature (measure of importance)
Output:	Fire once all potentiometers pass a certain threshold

Fired:	positive match in the image
Did not fire:	negative class

The Perceptron

Numerical Perceptron[1]

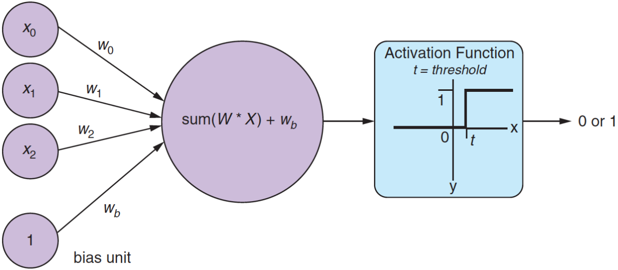


(Lane et al., 2019, p. 158)

[1] I am discarding any biological reference

The Perceptron

Numerical Perceptron[1]



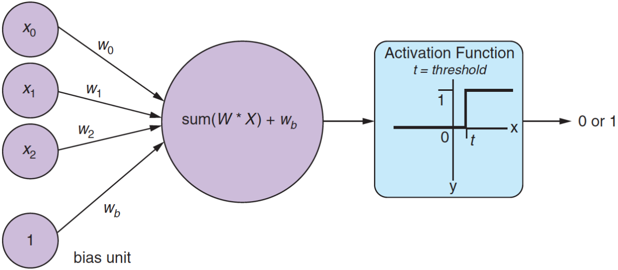
(Lane et al., 2019, p. 158)

- Feature vector: $X = [x_1, x_2, \ldots, x_i, \ldots, x_n]$

[1] I am discarding any biological reference

The Perceptron

Numerical Perceptron[1]



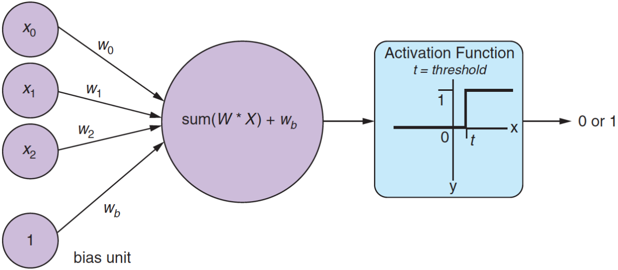
(Lane et al., 2019, p. 158)

- Feature vector: $X = [x_1, x_2, \ldots, x_i, \ldots, x_n]$
- Associated weight (per feature): $W = [w_1, w_2, \ldots, w_i, \ldots, w_n]$

[1] I am discarding any biological reference

The Perceptron

Numerical Perceptron[1]



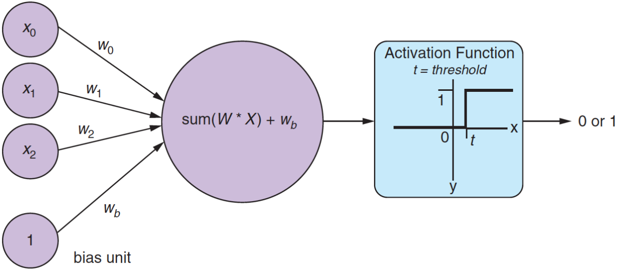
(Lane et al., 2019, p. 158)

- Feature vector: $X = [x_1, x_2, \ldots, x_i, \ldots, x_n]$
- Associated weight (per feature): $W = [w_1, w_2, \ldots, w_i, \ldots, w_n]$
- Sum up: $(x_1 * w_1) + (x_2 * w_2) + \ldots\ldots + (x_i * w_i) + \ldots$

[1] I am discarding any biological reference

The Perceptron

Numerical Perceptron[1]



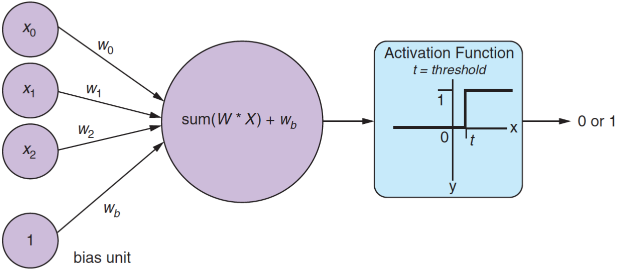
(Lane et al., 2019, p. 158)

- Feature vector: $X = [x_1, x_2, \ldots, x_i, \ldots, x_n]$
- Associated weight (per feature): $W = [w_1, w_2, \ldots, w_i, \ldots, w_n]$
- Sum up: $(x_1 * w_1) + (x_2 * w_2) + \ldots\ldots + (x_i * w_i) + \ldots$
- Activation (step) function

[1] I am discarding any biological reference

The Perceptron

Numerical Perceptron[1]



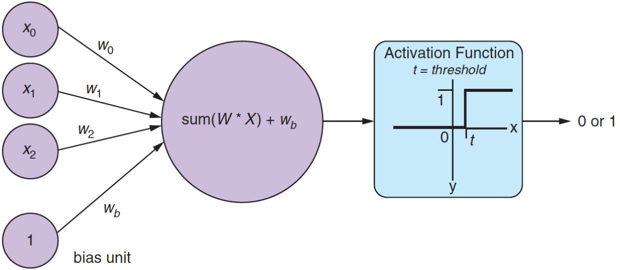
(Lane et al., 2019, p. 158)

- Feature vector: $X = [x_1, x_2, \ldots, x_i, \ldots, x_n]$
- Associated weight (per feature): $W = [w_1, w_2, \ldots, w_i, \ldots, w_n]$
- Sum up: $(x_1 * w_1) + (x_2 * w_2) + \ldots\ldots + (x_i * w_i) + \ldots$
- Activation (step) function
- Bias: always-on input (resilientcy to inputs of all zeros)

[1] I am discarding any biological reference

The Perceptron

Numerical Perceptron

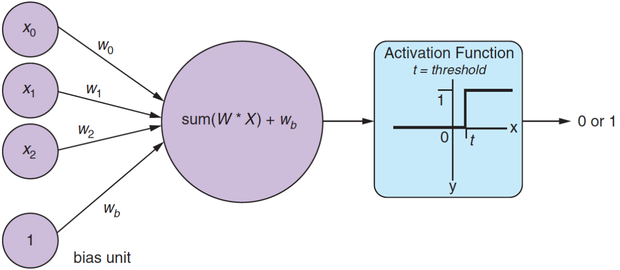


(Lane et al., 2019, p. 158)

$$f(\vec{x}) = \begin{cases} 1 & \text{if } \sum_{i=0}^{n} x_i w_i > \textit{threshold} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

The Perceptron

Numerical Perceptron



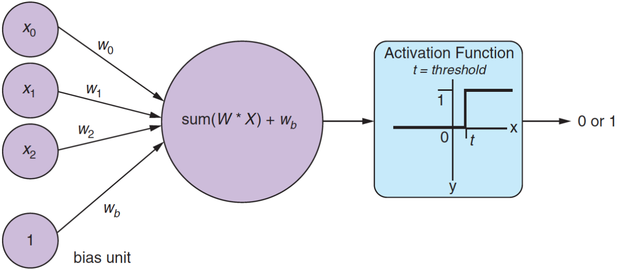
(Lane et al., 2019, p. 158)

$$f(\vec{x}) = \begin{cases} 1 & \text{if } \sum_{i=0}^{n} x_i w_i > threshold \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

This perceptron is a special case of **neuron** —the base unit of a neural network

The Perceptron

Numerical Perceptron



(Lane et al., 2019, p. 158)

$$f(\vec{x}) = \begin{cases} 1 & \text{if } \sum_{i=0}^{n} x_i w_i > threshold \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

This perceptron is a special case of **neuron** —the base unit of a neural network

Let us see

The Perceptron
Without Bias

"The output of [a perceptron] is a linear function of the input"
(Goodfellow et al., 2016, p. 105)

$$\hat{y} = w^T x \tag{2}$$

The Perceptron

Without Bias

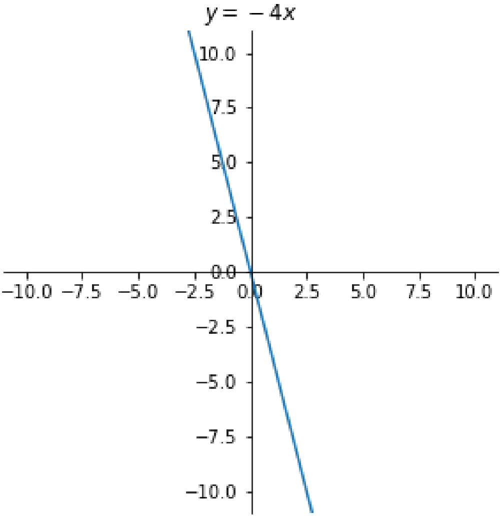
```
import matplotlib.pyplot as plt
import numpy as np
for i in range(-5, 5, 1):
    fig, ax = plt.subplots(figsize = (5,5))
    ax.spines['left'].set_position('center')
    ax.spines['bottom'].set_position('center')
    ax.spines['right'].set_color('none')
    ax.spines['top'].set_color('none')
    ax.set(title='$y=w^Tx$')
    x = np.arange(-5.0, 5.0, 0.01)
    plt.xlim((-5,+5))
    plt.ylim((-5,+5))
    ax.set(title='$y={}x$'.format(i))
    y = i*x  #1 + np.sin(2 * np.pi * x)
    ax.plot(x, y)
    fig.savefig("linear_w{}.png".format(i))
    plt.show()
```

Not the nicest way to plot

The Perceptron

Without Bias

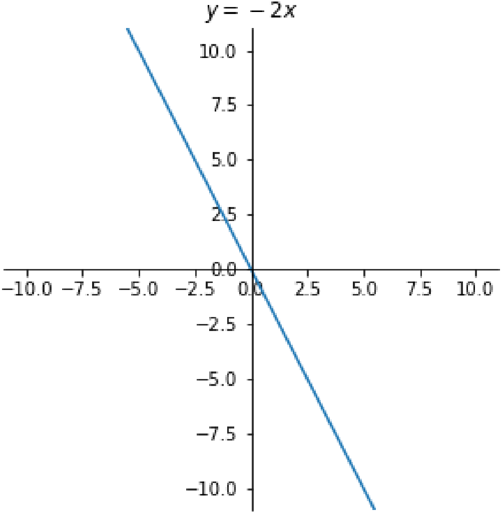
Plotting with different values of w



The Perceptron

Without Bias

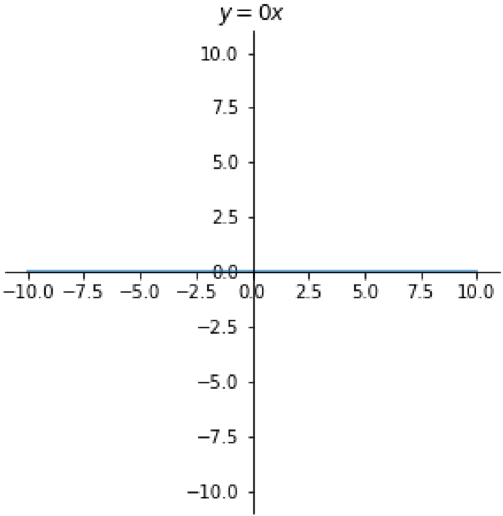
Plotting with different values of w



The Perceptron

Without Bias

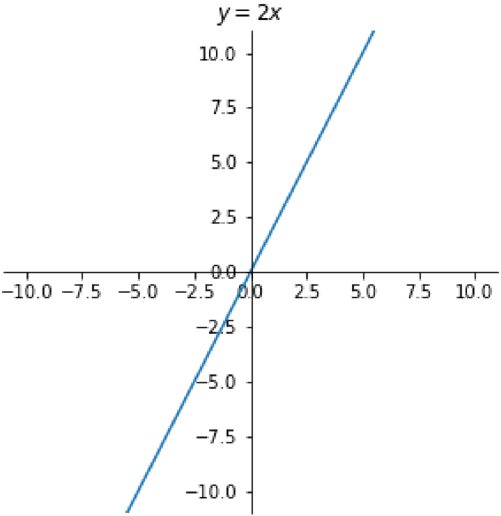
Plotting with different values of w



The Perceptron

Without Bias

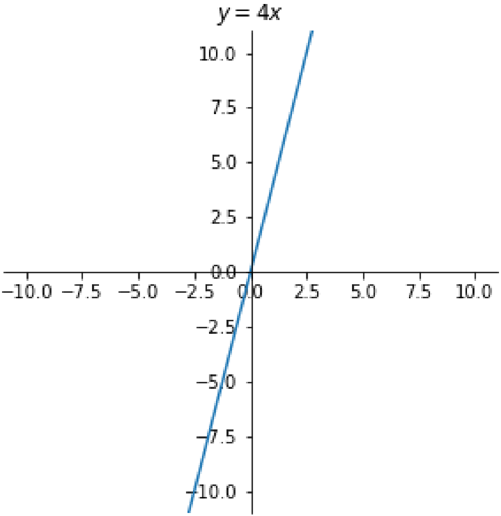
Plotting with different values of *w*



The Perceptron

Without Bias

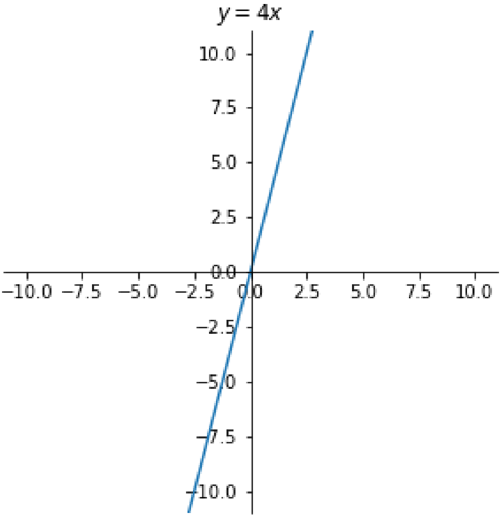
Plotting with different values of w



The Perceptron

Without Bias

Plotting with different values of w do you see an issue?



The Perceptron

With Bias

$$\hat{y} = w^T x + b \tag{3}$$

The Perceptron

With Bias

$$\hat{y} = w^T x + b \tag{3}$$

"[...] the mapping from parameters to predictions is still a linear function but the mapping from features to predictions is now an affine function" (Goodfellow et al., 2016, p. 107)

The Perceptron

With Bias

$$\hat{y} = w^T x + b \quad (3)$$

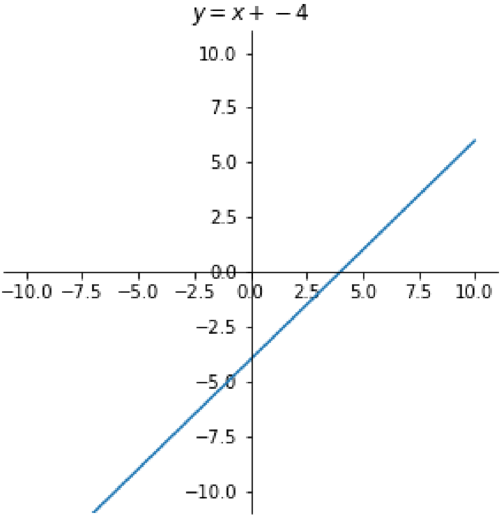
"[...] the mapping from parameters to predictions is still a linear function but the mapping from features to predictions is now an affine function" (Goodfellow et al., 2016, p. 107)

(does not need to pass by the origin)

The Perceptron

Without Bias

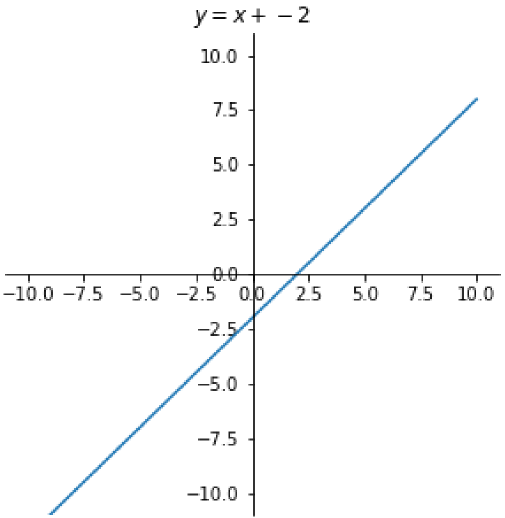
Plotting with $w = 1$ and different values of b



The Perceptron

Without Bias

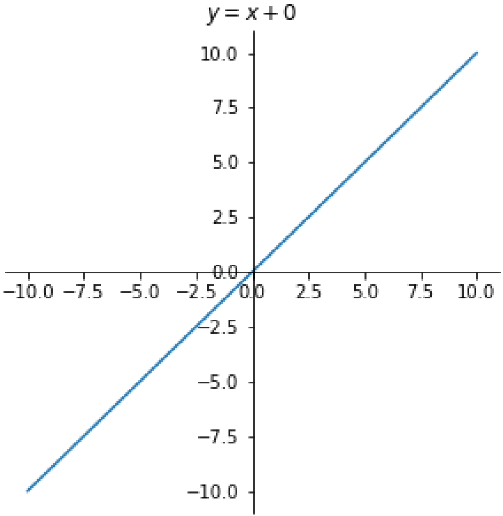
Plotting with $w = 1$ and different values of b



The Perceptron

Without Bias

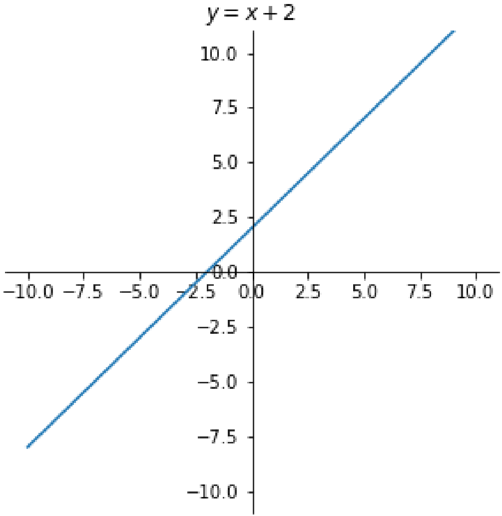
Plotting with $w = 1$ and different values of b



The Perceptron

Without Bias

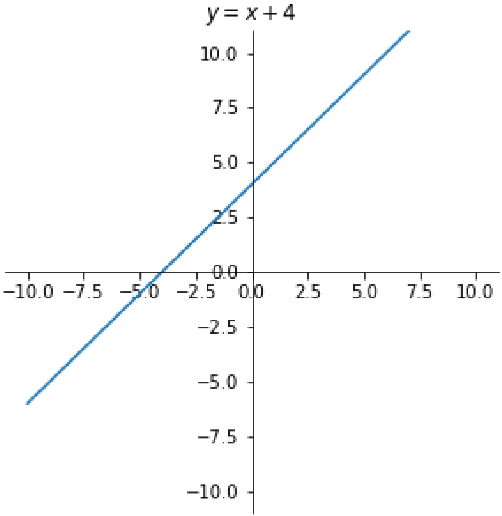
Plotting with $w = 1$ and different values of b



The Perceptron

Without Bias

Plotting with $w = 1$ and different values of b



The Perceptron

Typical Learning Process (1/2)

Given an annotated dataset. . .

- start with a random weight initialisation from a normal distribution

$$\vec{w} \sim \mathcal{N}(\mu, \sigma^2) \text{ with } \mu \sim 0 \text{ (but do not use 0!)}$$

The Perceptron

Typical Learning Process (1/2)

Given an annotated dataset. . .

- start with a random weight initialisation from a normal distribution

$$\vec{w} \sim \mathcal{N}(\mu, \sigma^2) \text{ with } \mu \sim 0 \text{ (but do not use 0!)}$$

- feed one instance and see if the resulting class is correct

The Perceptron

Typical Learning Process (1/2)

Given an annotated dataset...

- start with a random weight initialisation from a normal distribution

$$\vec{w} \sim \mathcal{N}(\mu, \sigma^2) \text{ with } \mu \sim 0 \text{ (but do not use 0!)}$$

- feed one instance and see if the resulting class is correct
- if the class is correct: do nothing
  else: readjust the weights (slightly; not until getting the class right!)

The Perceptron

Typical Learning Process (1/2)

Given an annotated dataset...

- start with a random weight initialisation from a normal distribution

$$\vec{w} \sim \mathcal{N}(\mu, \sigma^2) \text{ with } \mu \sim 0 \text{ (but do not use 0!)}$$

- feed one instance and see if the resulting class is correct
- if the class is correct: do nothing
  else: readjust the weights (slightly; not until getting the class right!)

The Perceptron

Typical Learning Process (1/2)

Given an annotated dataset...

- start with a random weight initialisation from a normal distribution

$$\vec{w} \sim \mathcal{N}(\mu, \sigma^2) \text{ with } \mu \sim 0 \text{ (but do not use 0!)}$$

- feed one instance and see if the resulting class is correct
- if the class is correct: do nothing
  else: readjust the weights (slightly; not until getting the class right!)

Each weight is adjusted by how much it contributed to the resulting error

Let us see

The Perceptron

Typical Learning Process (2/2)

- All instances in the training data are fed a number of times: **epoch**
- Typical stop criteria include
  - *error* $< \epsilon$ (convergence)
  - *error* stabilises
  - max number of epochs reached

The Perceptron

Example 1: Logical OR

input		**output**
0	0	0
0	1	1
1	0	1
1	1	1

The Perceptron

Example 1: Logical OR

input		output
0	0	0
0	1	1
1	0	1
1	1	1

Let us see

The Perceptron

Example 1: Logical OR

input		output
0	0	0
0	1	1
1	0	1
1	1	1

Let us see

Mr. Perceptron can learn!

The Perceptron

Example 1: Logical OR

input		output
0	0	0
0	1	1
1	0	1
1	1	1

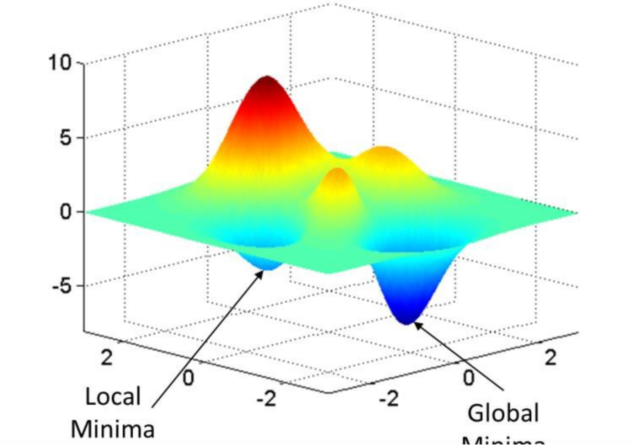
Let us see

Mr. Perceptron can learn!

This learning model is called **linear regression** (another ML alternative)

The Perceptron

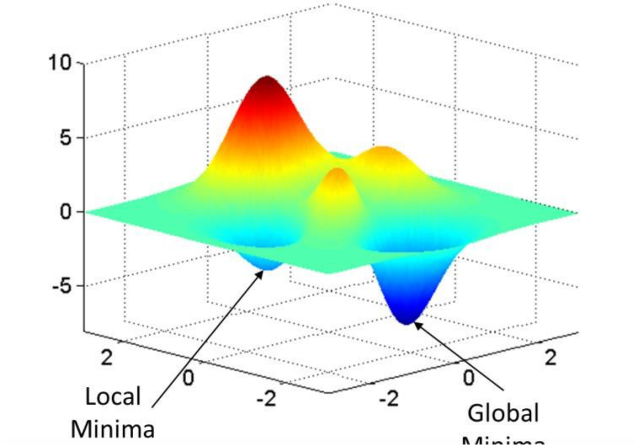
Drawback: Local vs Global Minimum



Plot from M. Ryan's thesis (http://www.isni.org/isni/000000045916099X)

The Perceptron

Drawback: Local vs Global Minimum



No guarantee that the model will reach the global optimal solution

Plot from M. Ryan's thesis (`http://www.isni.org/isni/000000045916099X`)

The Perceptron

Drawback: Linearly separable

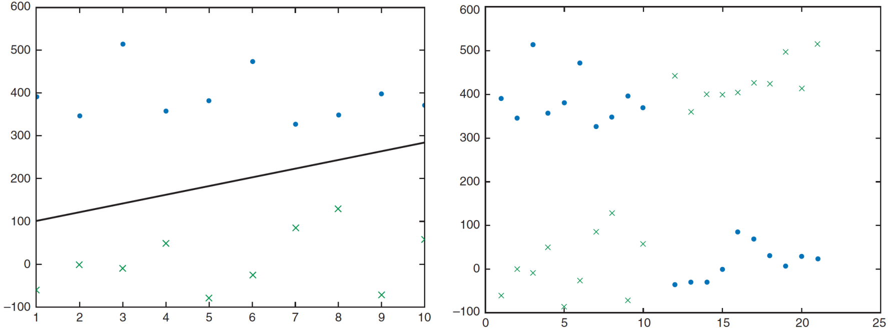
The perceptron can only deal with linearly separable data

Plots from (Lane et al., 2019, p. 164–165)

The Perceptron

Drawback: Linearly separable

The perceptron can only deal with linearly separable data

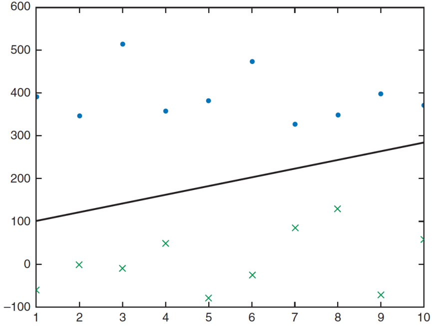


Plots from (Lane et al., 2019, p. 164–165)

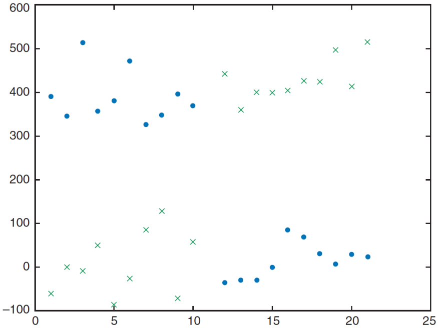
The Perceptron

Drawback: Linearly separable

The perceptron can only deal with linearly separable data



linearly separable



not linearly separable

Plots from (Lane et al., 2019, p. 164–165)

The Perceptron

Example 2: Logical XOR

We have learned a logical OR function ...

Can we learn a logical XOR?

The Perceptron

Example 2: Logical XOR

We have learned a logical OR function ...

Can we learn a logical XOR?

input		**output**
0	0	0
0	1	1
1	0	1
1	1	0

The Perceptron

Example 2: Logical XOR

We have learned a logical OR function ...

Can we learn a logical XOR?

input		**output**
0	0	0
0	1	1
1	0	1
1	1	0

Let us see

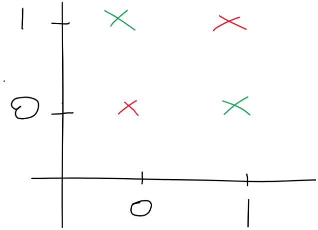
The Perceptron

Example 2: Logical XOR

We have learned a logical OR function ...

Can we learn a logical XOR?

input		output
0	0	0
0	1	1
1	0	1
1	1	0



Let us see

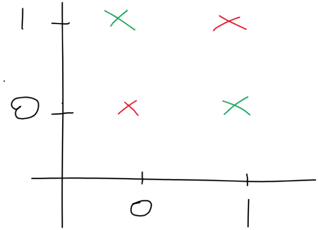
The Perceptron

Example 2: Logical XOR

We have learned a logical OR function . . .

Can we learn a logical XOR?

input		output
0	0	0
0	1	1
1	0	1
1	1	0



Let us see

Mr. Perceptron **cannot** learn! **. . . winter**

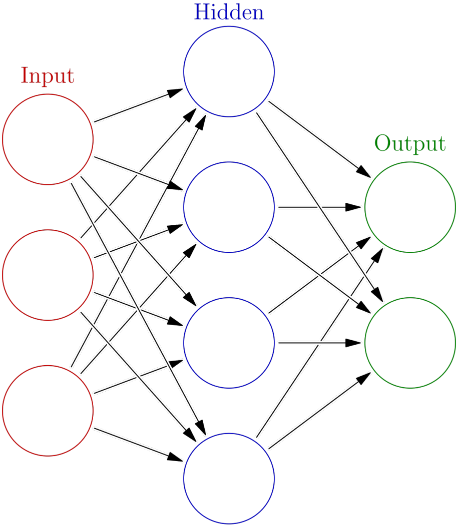
More than One Neuron

Neural Networks

A neural network is a combination of multiple perceptrons (and it can deal with more complex patterns)

Neural Networks

A neural network is a combination of multiple perceptrons (and it can deal with more complex patterns)



Some Formalisms

Input $x = [x_1, x_2, x_3, \ldots, x_k]$

Output $f(x)$[2]

Answer y

---

[2] aka y'

[3] aka loss function

Some Formalisms

Input $x = [x_1, x_2, x_3, \ldots, x_k]$

Output $f(x)$[2]

Answer y

Cost Function[3] Quantifier of the mismatch between **actual** and **predicted** output

$$err(x) = |y - f(x)| \tag{4}$$

[2] aka y'

[3] aka loss function

Some Formalisms

Input $x = [x_1, x_2, x_3, \ldots, x_k]$

Output $f(x)$[2]

Answer y

Cost Function[3] Quantifier of the mismatch between **actual** and **predicted** output

$$err(x) = |y - f(x)| \tag{4}$$

Training goal Minimising the cost function across all input samples

$$J(x) = \min \sum_{i=1}^{n} err(x_i) \tag{5}$$

[2] aka y'

[3] aka loss function

Some Formalisms

Input $x = [x_1, x_2, x_3, \ldots, x_k]$

Output $f(x)$[2]

Answer y

Cost Function[3] Quantifier of the mismatch between **actual** and **predicted** output

$$err(x) = |y - f(x)| \tag{4}$$

Training goal Minimising the cost function across all input samples

$$J(x) = \min \sum_{i=1}^{n} err(x_i) \tag{5}$$

[2]aka y'

[3]aka loss function

Next

- Backpropagation (briefly)
- Activation functions
- Keras

References

Goodfellow, I., Y. Bengio, and A. Courville
2016. *Deep Learning*. MIT Press.
http://www.deeplearningbook.org.

Lane, H., C. Howard, and H. Hapkem
2019. *Natural Language Processing in Action*. Shelter Island, NY: Manning Publication Co.

Rosenblatt, F.
1957. The perceptron—a perceiving and recognizing automaton. Technical Report 85-460-1, Cornell Aeronautical Laboratory, Buffallo, NY.