Mechatronic Engineering program

Basics of AI and Deep Learning:
4: Artificial Neural Networks – in three flavors

Ziemowit Dworakowski
AGH University of Krakow

Understanding is measured by how many different perspectives you can internalize for a given subject

1

So far, ve've learned:

- What is classification and regression – from the data perspective
- How to use optimization methods for classifier or regressor training
- What is a feature space and what can we find in it…

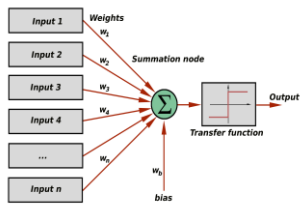
Today, we'll learn about neural networks

Understanding is measured by how many different perspectives you can internalize for a given subject…

2

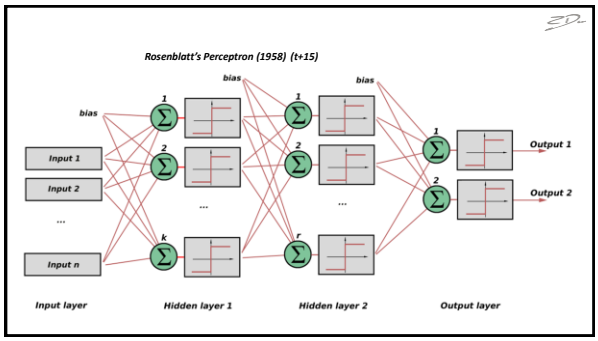
Lets model a neuron:

McCulloch – Pitts neuron (1943)

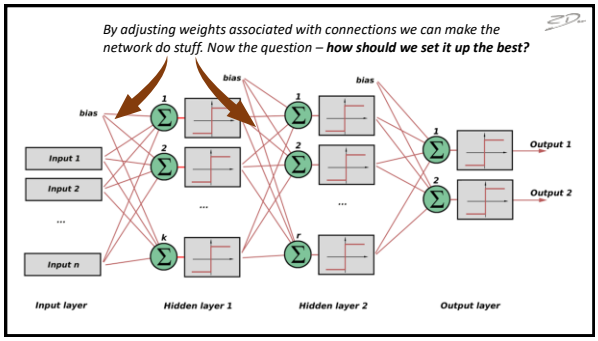


One neuron is not really that brilliant… But maybe we can connect it with others?

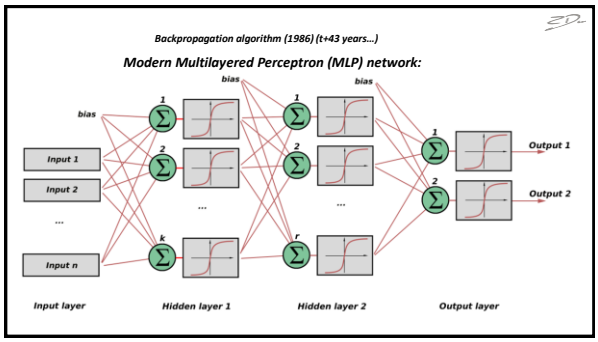
3


Rosenblatt's Perceptron (1958) (t+15)
bias
bias
bias
Input 1
Input 2
Input n
Output 1
Output 2
Input layer
Hidden layer 1
Hidden layer 2
Output layer

4

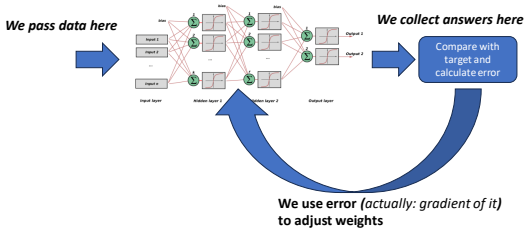

By adjusting weights associated with connections we can make the network do stuff. Now the question – how should we set it up the best?
bias
bias
bias
Input 1
Input 2
Input n
Output 1
Output 2
Input layer
Hidden layer 1
Hidden layer 2
Output layer

5


Backpropagation algorithm (1986) (t+43 years...)
Modern Multilayered Perceptron (MLP) network:
bias
bias
bias
Input 1
Input 2
Input n
Output 1
Output 2
Input layer
Hidden layer 1
Hidden layer 2
Output layer

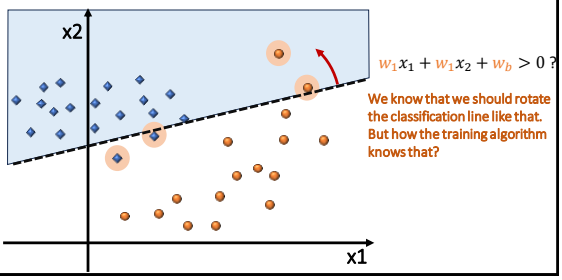
6

How we make MLP do the stuff we want?



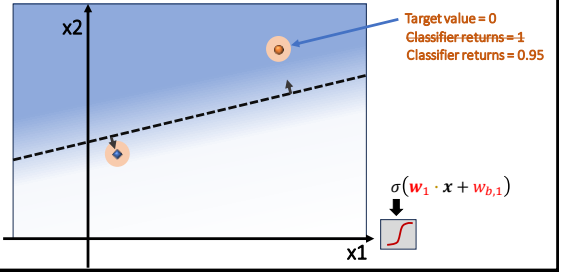
7

Training interpretation from classification perspective



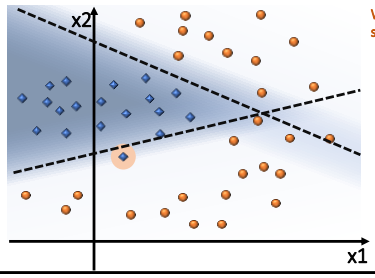
8

Training interpretation from classification perspective



9

Training interpretation from classification perspective

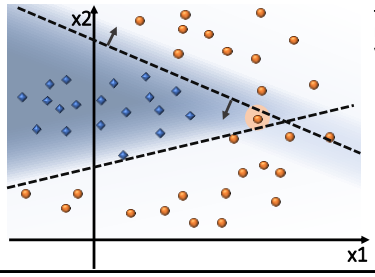


We can add several surfaces together!

$$\sigma(\boldsymbol{w}_1 \cdot \boldsymbol{x} + w_{b,1}) + \sigma(\boldsymbol{w}_2 \cdot \boldsymbol{x} + w_{b,2}) + \dots$$

10

Training interpretation from classification perspective



The more „lines" we have, the more flexible we are

And each of these is just a neuron!

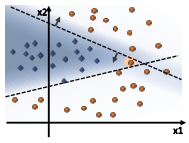
Look:

$$\sigma(\boldsymbol{w}_1 \cdot \boldsymbol{x} + w_{b,1})$$

11

ANN training: (Vanilla) gradient descent

(standard, without additions)



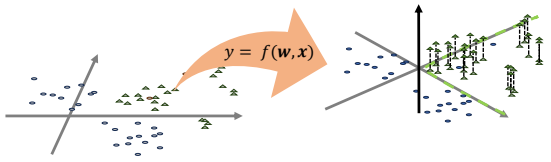
(Start from a random set of weights)

Calculate gradient of error with respect to all the weights

Modifiy the weights in the direction of the steepest gradient descent

12

Data space mapping function



We'll need lots of flexibility and lots of non-linearity…
And we want it to be adjustable according to needs…

13

$$y = f(\boldsymbol{w}, \boldsymbol{x})$$

We might want to make it compounding from different functions:

$$y = f_1(\boldsymbol{w}_1, \boldsymbol{x}) + f_2(\boldsymbol{w}_2, \boldsymbol{x}) + \ldots + f_n(\boldsymbol{w}_n, \boldsymbol{x})$$

We might want to govern these functions purely by their parameters:

$$\begin{bmatrix} x'_1 \\ x'_2 \\ \ldots \end{bmatrix} = \begin{matrix} \sigma(\boldsymbol{w}_{1,1}, \boldsymbol{x}) + \sigma(\boldsymbol{w}_{2,1}, \boldsymbol{x}) + \ldots + \sigma(\boldsymbol{w}_{n,1}, \boldsymbol{x}) \\ \sigma(\boldsymbol{w}_{1,2}, \boldsymbol{x}) + \sigma(\boldsymbol{w}_{2,2}, \boldsymbol{x}) + \ldots + \sigma(\boldsymbol{w}_{n,2}, \boldsymbol{x}) \\ \ldots \end{matrix}$$

This is a vector of our new coordinates!

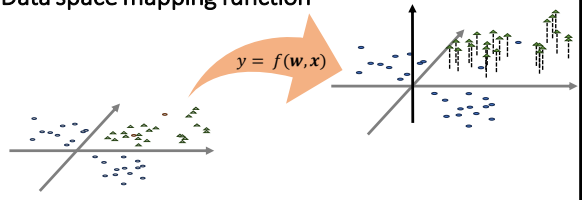
And now a „final decision" – another nonlinear transformation:

$$y = \sigma(\boldsymbol{w}_{n+1,1}, \boldsymbol{x}')$$

If we want, we can nest it even further (so y becomes $\boldsymbol{x}''$ and so on)

14

Data space mapping function



And this tool for transformation of feature space (x) into output space (y) we'll call a ***neural network:***

$$y = \sigma(\boldsymbol{w}_{n+1,1}, \boldsymbol{x}')$$

We adjust it to data using a gradient-descent backpropagation algorithm

15

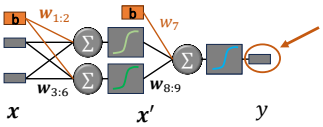
Three flavors of neural networks:

neural network is:

- **A compound function** that does complex non-linear mapping from input space into output space morphing data to gradually simplify its structure
- A **set of hyperplanes** in feature space (whose location we can adjust) that form layered decision surfaces
- A **biologically-inspired network of nodes** that can pass information through one another with **emergent feature** of data understanding

16

Consider a simple Multilayer Perceptron (MLP) network:



$$y = \sigma(w_8 \cdot x'_1 + w_9 \cdot x'_2 + w_7)$$

$$x'_1 = \sigma(w_3 \cdot x_1 + w_5 \cdot x_2 + w_1)$$

$$x'_2 = \sigma(w_2 \cdot x_1 + w_6 \cdot x_2 + w_2)$$

You can combine this to get a $f(\boldsymbol{w}, \boldsymbol{x})$ function equivalent for a network

We also have a target t associated with each input vector $\boldsymbol{x}$. Using that, we can calculate error d:

$$d = \frac{1}{2}(y - t)^2$$

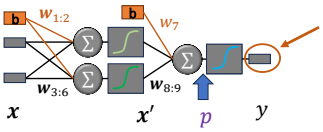
Now we can calculate gradient of d with respect to weights $\boldsymbol{w}$:

$$\nabla d = \left[\frac{\partial d}{\partial w_1}, \frac{\partial d}{\partial w_2}, \frac{\partial d}{\partial w_3}, \ldots, \frac{\partial d}{\partial w_9}\right]$$

The gradient now tells us how to adjust weights

17

Consider a simple Multilayer Perceptron (MLP) network:



$$y = \sigma(w_8 \cdot x'_1 + w_9 \cdot x'_2 + w_7)$$

$$\frac{\partial d}{\partial w_9} = \frac{\partial d}{\partial y}\frac{\partial y}{\partial p}\frac{\partial p}{\partial w_9} = (y - t)y(1 - y) \cdot x'_2$$

$$\frac{\partial d}{\partial y} = y - t$$

We also have a target t associated with each input vector $\boldsymbol{x}$. Using that, we can calculate error d:

$$d = \frac{1}{2}(y - t)^2$$

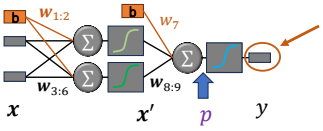
Now we can calculate gradient of d with respect to weights $\boldsymbol{w}$:

$$\nabla d = \left[\frac{\partial d}{\partial w_1}, \frac{\partial d}{\partial w_2}, \frac{\partial d}{\partial w_3}, \ldots, \frac{\partial d}{\partial w_9}\right]$$

The gradient now tells us how to adjust weights

18

Consider a simple Multilayer Perceptron (MLP) network:



$$y = \sigma(w_8 \cdot x'_1 + w_9 \cdot x'_2 + w_7)$$

$$\frac{\partial d}{\partial w_9} = \frac{\partial d}{\partial y}\frac{\partial y}{\partial p} = \frac{\partial d}{\partial y}\frac{\partial y}{\partial p}\frac{\partial p}{\partial w_9} = (y-t)y(1-y)\cdot x'_2$$

$$y(1-y) \leftarrow y = \sigma(p) = \frac{1}{1+e^{-(p)}}$$

We also have a target t associated with each input vector x. Using that, we can calculate error d:

$$d = \frac{1}{2}(y-t)^2$$

Now we can calculate gradient of d with respect to weights w:

$$\nabla d = \left[\frac{\partial d}{\partial w_1}, \frac{\partial d}{\partial w_2}, \frac{\partial d}{\partial w_3}, \ldots, \frac{\partial d}{\partial w_9}\right]$$

The gradient now tells us how to adjust weights

19

$$y = \sigma(p) = \frac{1}{1+e^{-(p)}}$$

$$\frac{\partial \sigma}{\partial p} = \frac{\partial}{\partial p}(1+e^{-p})^{-1} = \cancel{(1+e^{-p})^{-2}} \cdot e^{-p} \cdot \cancel{1}$$

$$\frac{\partial \sigma}{\partial p} = \frac{1}{1+e^{-(p)}} \cdot \frac{1}{1+e^{-(p)}} \cdot e^{-p}$$

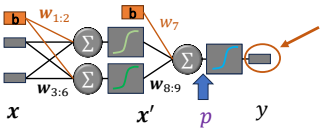
$$\frac{\partial \sigma}{\partial p} = \frac{e^{-p}}{1+e^{-(p)}} \cdot \frac{1}{1+e^{-(p)}}$$

$$\frac{\partial \sigma}{\partial p} = \frac{1+e^{-p}-1}{1+e^{-(p)}} \cdot \frac{1}{1+e^{-(p)}}$$

$$\frac{\partial \sigma}{\partial p} = \left(1 - \frac{1}{1+e^{-(p)}}\right) \cdot \frac{1}{1+e^{-(p)}} = (1-y)\cdot y$$

20

Consider a simple Multilayer Perceptron (MLP) network:



$$y = \sigma(w_8 \cdot x'_1 + w_9 \cdot x'_2 + w_7)$$

$$\frac{\partial d}{\partial w_9} = \frac{\partial d}{\partial y}\frac{\partial y}{\partial p} = \frac{\partial d}{\partial y}\frac{\partial y}{\partial p}\frac{\partial p}{\partial w_9}$$

$$\frac{\partial p}{\partial w_9} = x'_2$$

We also have a target t associated with each input vector x. Using that, we can calculate error d:

$$d = \frac{1}{2}(y-t)^2$$

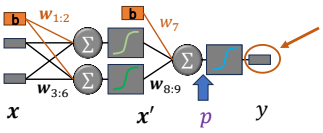
Now we can calculate gradient of d with respect to weights w:

$$\nabla d = \left[\frac{\partial d}{\partial w_1}, \frac{\partial d}{\partial w_2}, \frac{\partial d}{\partial w_3}, \ldots, \frac{\partial d}{\partial w_9}\right]$$

The gradient now tells us how to adjust weights

21

Consider a simple Multilayer Perceptron (MLP) network:



$$y = \sigma(w_8 \cdot x'_1 + w_9 \cdot x'_2 + w_7)$$

$$\frac{\partial d}{\partial w_9} = \frac{\partial d}{\partial y}\frac{\partial y}{\partial p}\frac{\partial p}{\partial w_9} = (y-t)y(1-y)\cdot x'_2$$

$$\frac{\partial p}{\partial w_9} = x'_2$$

We also have a target t associated with each input vector x. Using that, we can calculate error d:

$$d = \frac{1}{2}(y-t)^2$$

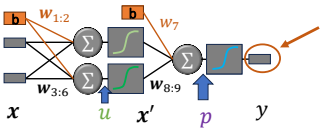
Now we can calculate gradient of d with respect to weights w:

$$\nabla d = \left[\frac{\partial d}{\partial w_1}, \frac{\partial d}{\partial w_2}, \frac{\partial d}{\partial w_3}, \ldots, \frac{\partial d}{\partial w_9}\right]$$

The gradient now tells us how to adjust weights

22

Consider a simple Multilayer Perceptron (MLP) network:



$$y = \sigma(w_8 \cdot x'_1 + w_9 \cdot x'_2 + w_7)$$

$$x'_2 = \sigma(w_4 \cdot x_1 + w_6 \cdot x_2 + w_1)$$

$$\frac{\partial d}{\partial w_9} = \frac{\partial d}{\partial y}\frac{\partial y}{\partial p}\frac{\partial p}{\partial w_9} = \frac{\partial d}{\partial y}\frac{\partial y}{\partial p}\frac{\partial p}{\partial w_9} = (y-t)y(1-y)\cdot x'_2$$

$$\frac{\partial d}{\partial w_6} = \frac{\partial d}{\partial y}\frac{\partial y}{\partial p}\frac{\partial p}{\partial x'_2}\frac{\partial x'_2}{\partial u}\frac{\partial u}{\partial w_6}$$

We also have a target t associated with each input vector x. Using that, we can calculate error d:

$$d = \frac{1}{2}(y-t)^2$$

Now we can calculate gradient of d with respect to weights w:

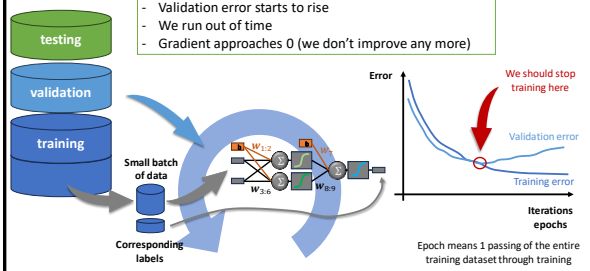
$$\nabla d = \left[\frac{\partial d}{\partial w_1}, \frac{\partial d}{\partial w_2}, \frac{\partial d}{\partial w_3}, \ldots, \frac{\partial d}{\partial w_9}\right]$$

The gradient now tells us how to adjust weights

As we go through the network backwards, we propagate our error calculations – it is **the backpropagation algorithm**

23

We stop training when:
- Validation error starts to rise
- We run out of time
- Gradient approaches 0 (we don't improve any more)

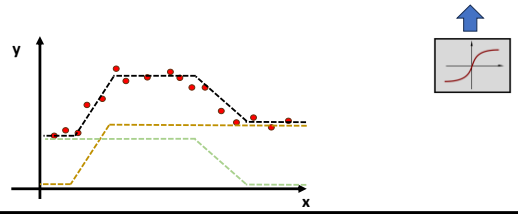


24

Regression with multilayered perceptron

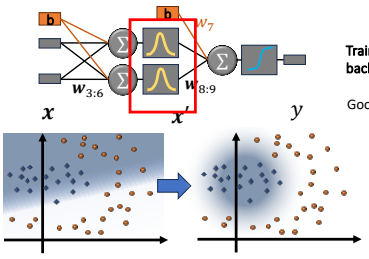
The idea here is to take a lot of simple nonlinear functions and add them together to get a more complex one

$$f(\boldsymbol{w},\boldsymbol{x}) = f_1(\boldsymbol{w}_1,\boldsymbol{x}) + f_2(\boldsymbol{w}_2,\boldsymbol{x}) + \cdots + f_3(\boldsymbol{w}_3,\boldsymbol{x})$$



25

Radial networks

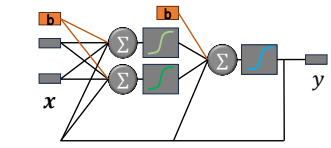


Trained (again!) using gradient backpropagation

Good for local inference

26

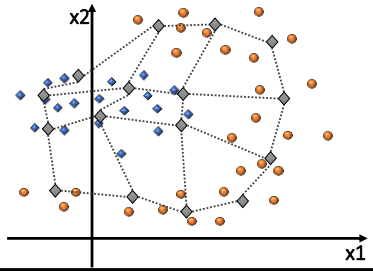
Recurrent networks



Adding recurrent connections makes for good short-time memory capabilities – allowing for system identification and prediction of time-domain signals

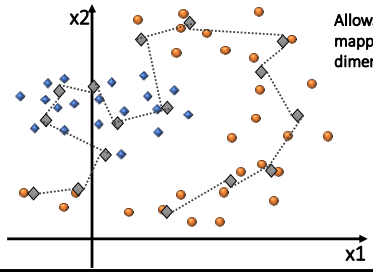
27

Kohonen networks (Self-organizing maps)



28

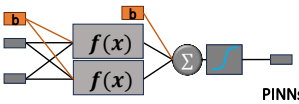
Kohonen networks (Self-organizing maps)



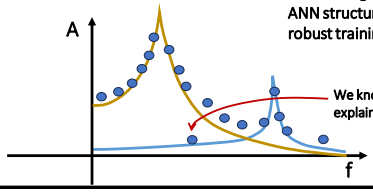
Allows for clusterization and mapping of data to lower dimensionality

29

Physics-informed neural networks



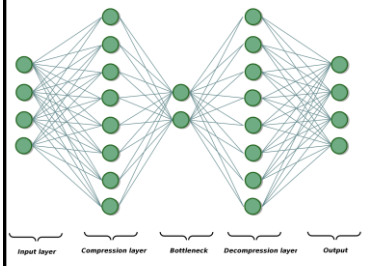
PINNs integrate expert knowledge into ANN structure – which allows more robust training with fewer examples



We know this is not physically explainable – probably an outlier

30

Autoencoders



They are taught to recreate input at output

They compress data and are used in detection of anomalies

31

Things to remember:

1. Explain Multilayered Perceptron Networks using one of the three explanations
2. Draw and annotate McCulloch-Pitts neuron model
3. Explain why McCulloch-Pitts neuron is represented by a straight line in a feature space
4. Draw and annotate full scheme of a Multilayer Perceptron Network
5. Explain the idea of a gradient backpropagation algorithm (no need to memorize calculations)
6. Write equation for a sigmoid activation function
7. Explain basic procedure that allows to prevent MLP network from overfitting (how do we use training and validation data to this end?)
8. Explain general idea behind selected network structures different than MLP (Kohonen, PINN, Radial, Autoencoders, Recurrent)

32