Lecture 07:

Introduction to Artificial Neural Networks (ANN)

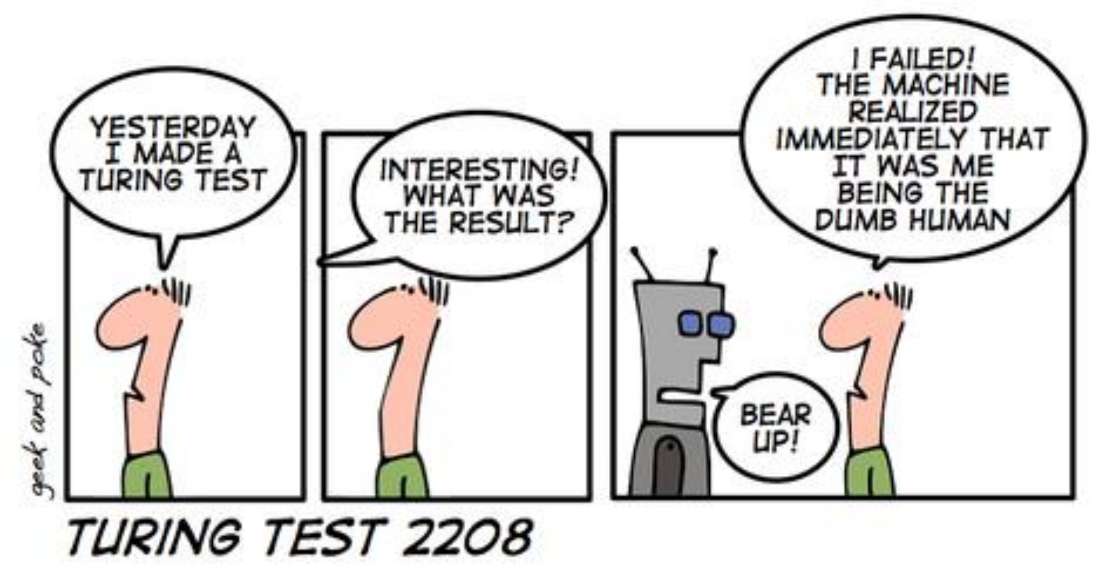
Markus Hohle

University California, Berkeley

Machine Learning Algorithms

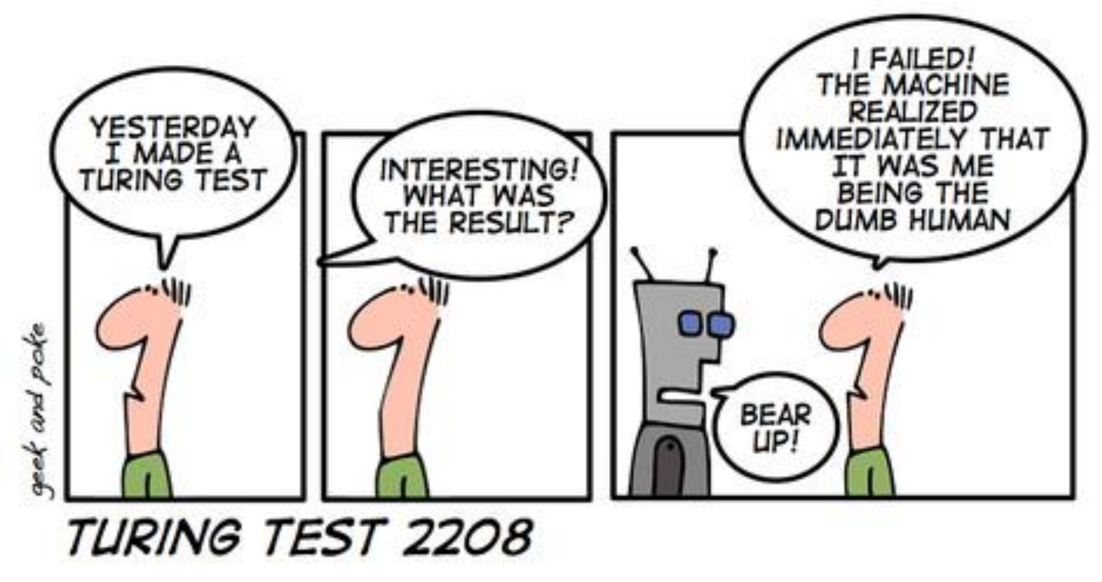
MSSE 277B, 3 Units

Spring 2025



Outline

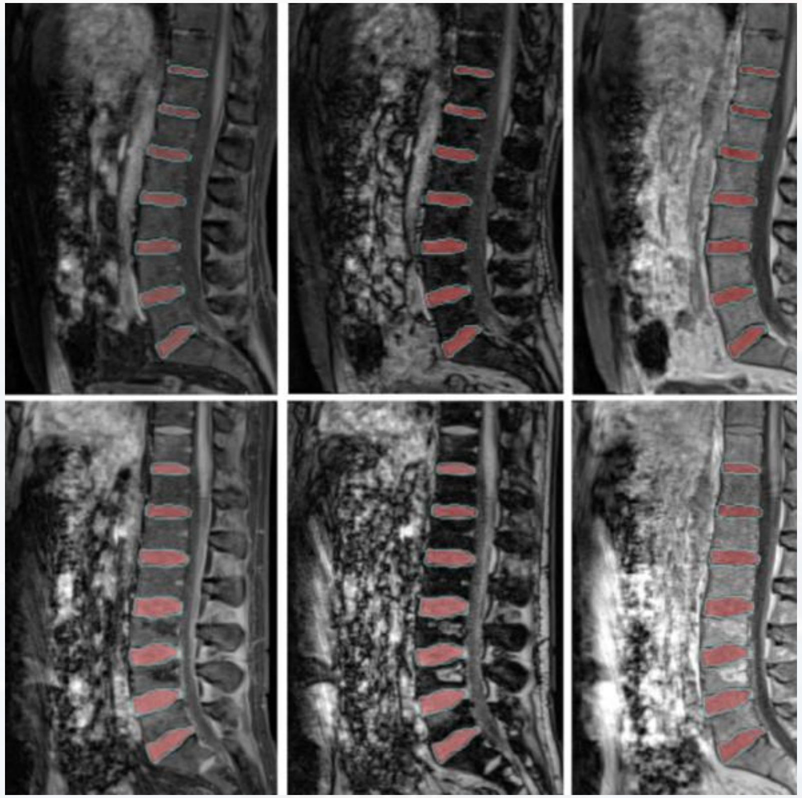
- **Motivation & Overview**
- **The Perceptron aka Neuron**
- **Building a Perceptron**

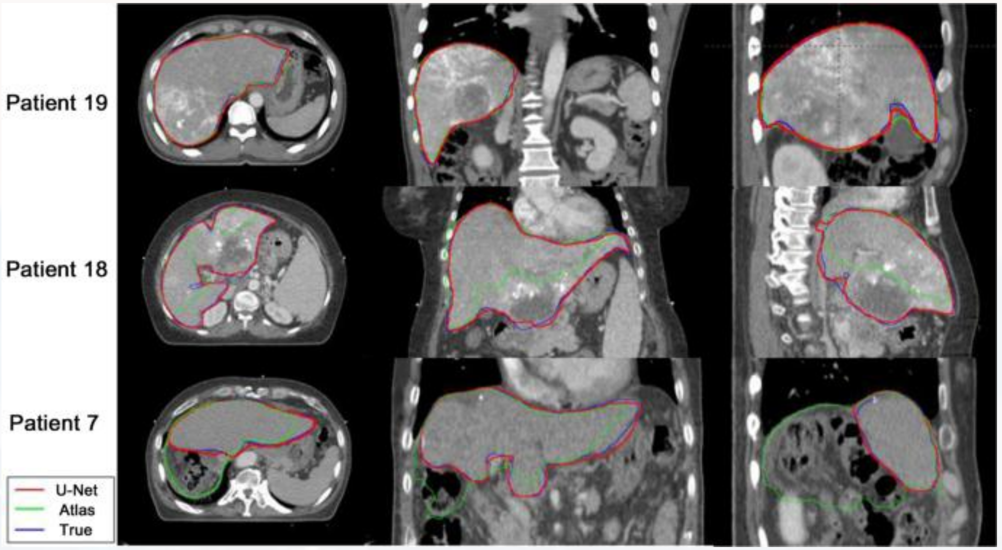


Outline

- **Motivation & Overview**
- The Perceptron aka Neuron
- Building a Perceptron

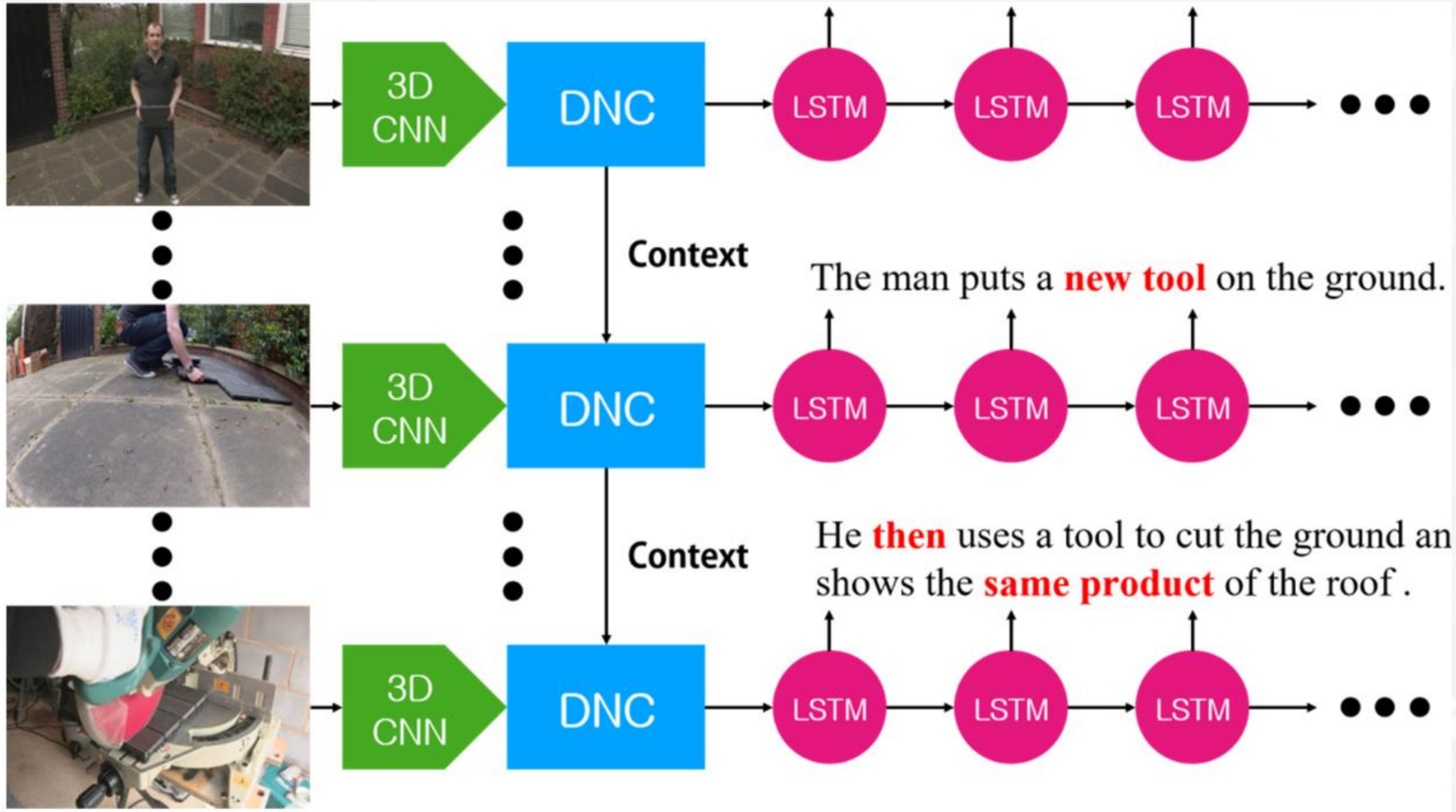
Why Artificial Neural Networks?





ANN - Intro: Motivation & Overview

Why Artificial Neural Networks?



Why Artificial Neural Networks?

Article

DeepSOCIAL: Social Distancing Monitoring and Infection Risk Assessment in COVID-19 Pandemic

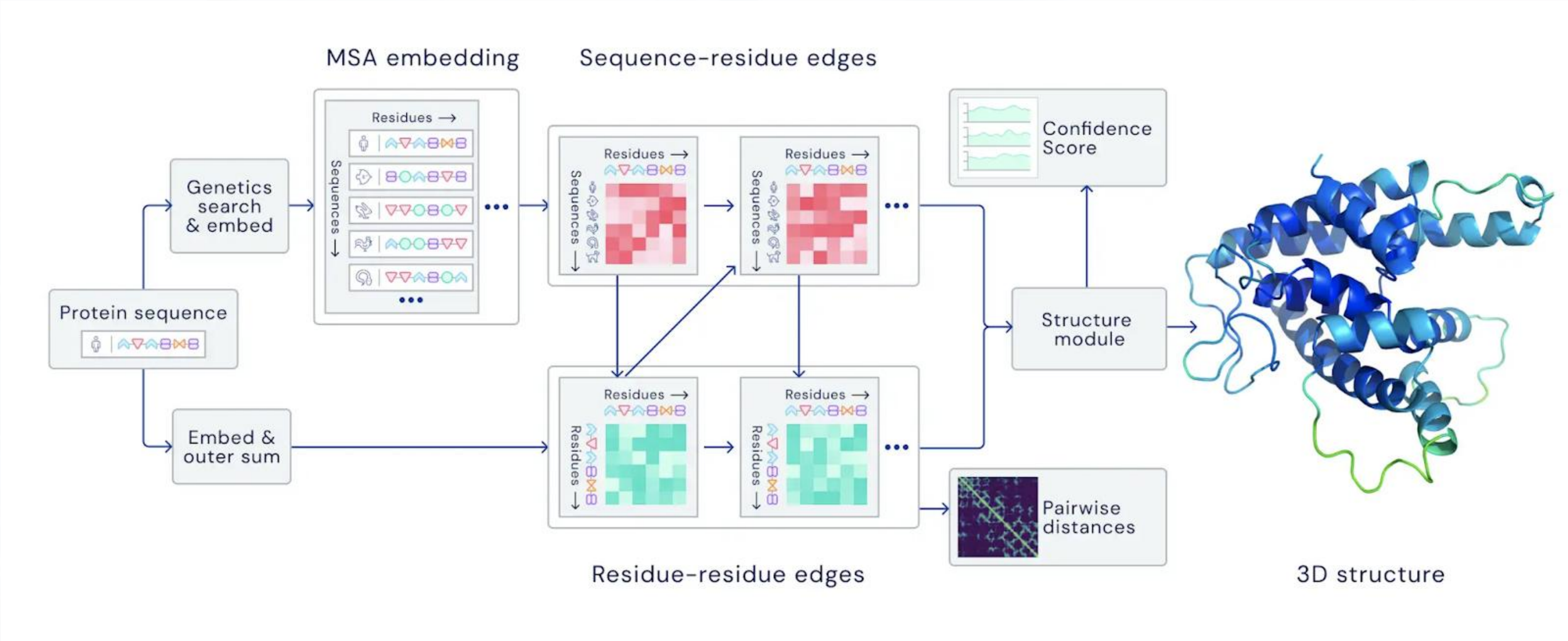
YOLOv4-based *Deep Neural Network* (DNN) model for automated people detection in the crowd in indoor and outdoor environments using common CCTV security cameras. The proposed DNN model in combination with an adapted inverse perspective mapping (IPM) technique and SORT tracking algorithm leads to a robust people detection and social distancing monitoring. The model has been trained against two most comprehensive datasets by the time of the research—the Microsoft Common Objects in Context (MS COCO) and Google Open Image datasets. The system has been

health authorities have set the 2-m physical distancing as a mandatory safety measure in shopping centres, schools and other covered areas. In this research, we develop a hybrid *Computer Vision* and YOLOv4-based *Deep Neural Network* (DNN) model for automated people detection in the crowd in indoor and outdoor environments using common CCTV security cameras. The proposed DNN model in combination with an adapted inverse perspective mapping (IPM) technique and SORT tracking algorithm leads to a robust people detection and social distancing monitoring. The model has been trained against two most comprehensive datasets by the time of the research—the Microsoft Common Objects in Context (MS COCO) and Google Open Image datasets. The system has been evaluated against the Oxford Town Centre dataset (including 150,000 instances of people detection)

Why Artificial Neural Networks?

Demis Hassabis @demishassabis

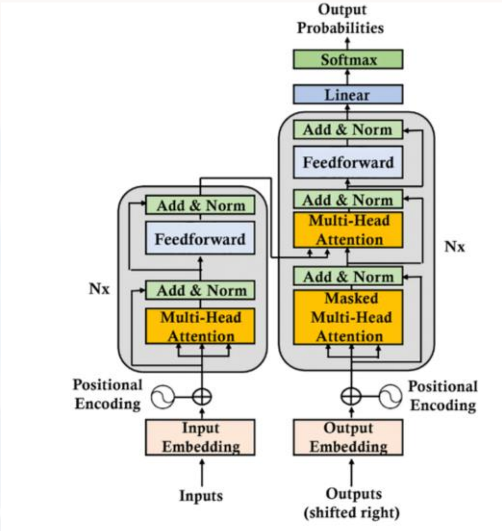
Thrilled to announce our first major breakthrough in applying AI to a grand challenge in science. #AlphaFold has been validated as a solution to the 'protein folding problem' & we hope it will have a big impact on disease understanding and drug discovery:



ANN - Intro: Motivation & Overview

Why Artificial Neural Networks?

ChatGPT



language models
“understanding” context

AI art

AI generated videos

Why Artificial Neural Networks?

Reuters World US Election Business Markets Sustainability Legal Breakingviews Technology

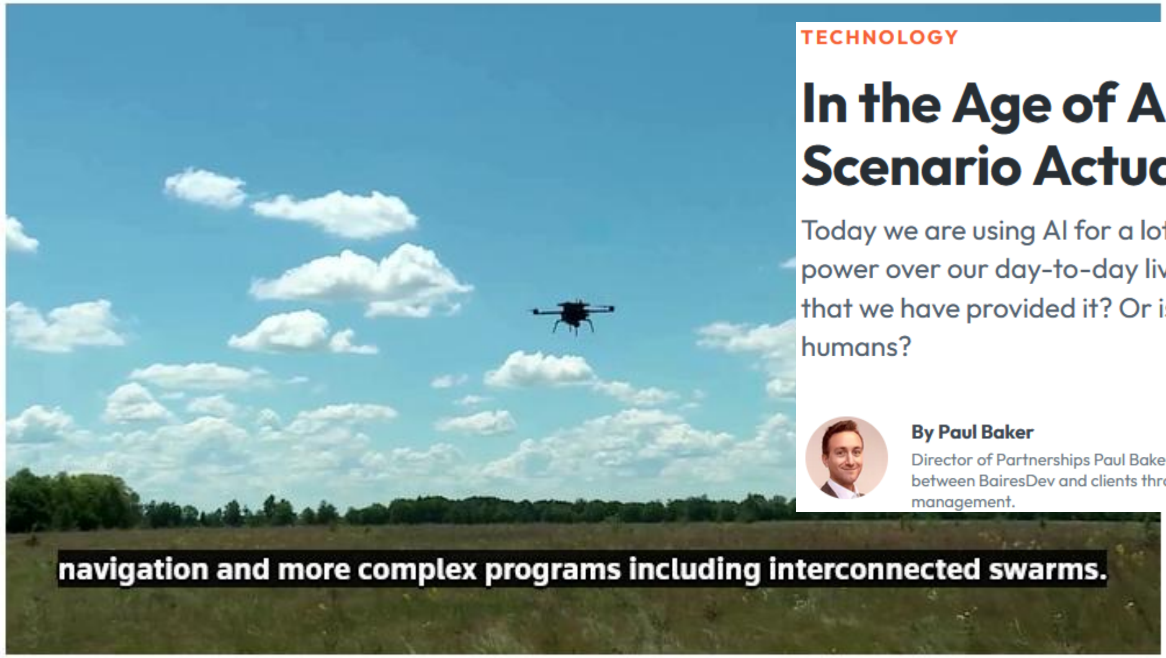
Artificial Intelligence | Ukraine and Russia at War

Ukraine rushes to create AI-enabled war drones

By Max Hunder

July 18, 2024 3:36 PM GMT+2 · Updated 2 months ago

Aa



TECHNOLOGY

In the Age of AI, Can a Terminator Scenario Actually Happen?

Today we are using AI for a lot of reasons. And unknowingly, we are giving it a lot of power over our day-to-day lives. But can AI turn upon us with the same resources that we have provided it? Or is it a fantasy to think that AI can ever go against humans?

By Paul Baker

Director of Partnerships Paul Baker builds strong business relationships between BairesDev and clients through strategy and partnership management.

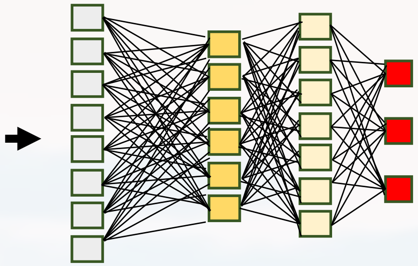
6 min read Share

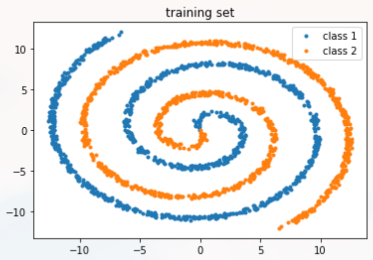
In the next lectures we will learn about:

- classical multi-layer ANN	regression, classification
- **C**onvolution **N**eural **N**etworks	image classification, segmentation, sequence analysis
- **R**ecurrent **N**eural **N**etworks	time series forecasting and classification
- **L**ong-**S**hort-**T**erm-**M**emory ANNs	time series forecasting and classification, language processing
- combining the above networks	
- **G**raph **N**eural **N**etworks	molecular 3D structures, process/workflow planning
- Transformers	time series analysis, **L**arge **L**anguage **M**odels, **N**atural **L**anguage **P**rocessing

bonus: **- running ANNs on GPUs using Cuda in PyTorch → 20 – 50 times faster!**
- Parallel Processing in Python

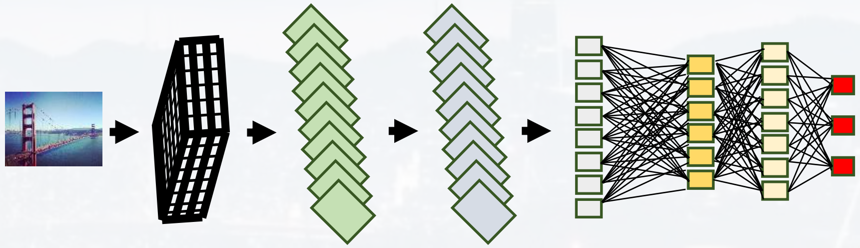
fully connected
(ANN)



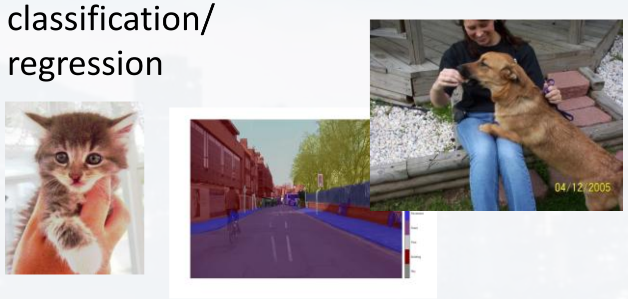


classification/
regression

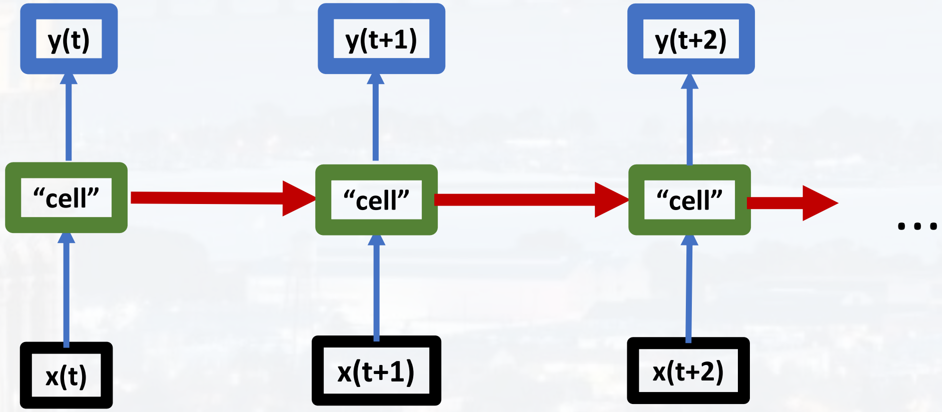
convolution
(CNN)



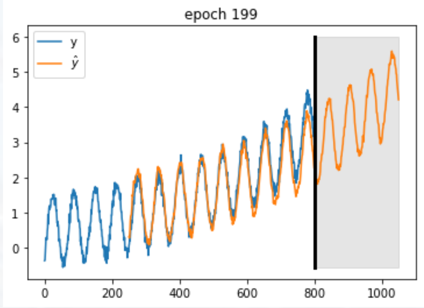
classification/
regression



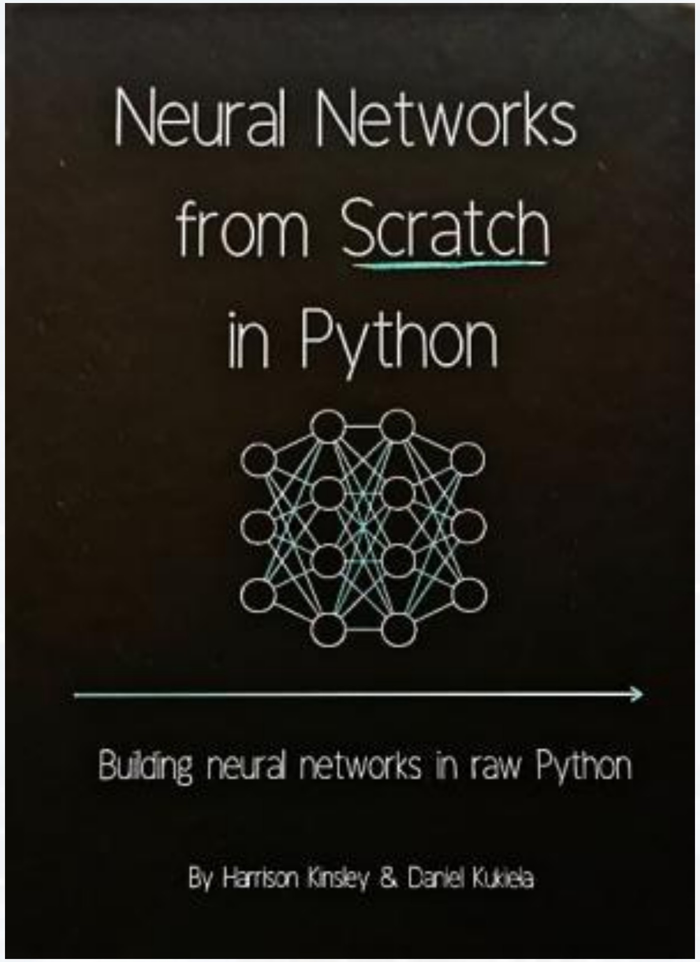
recurrent
(RNN, LSTM)

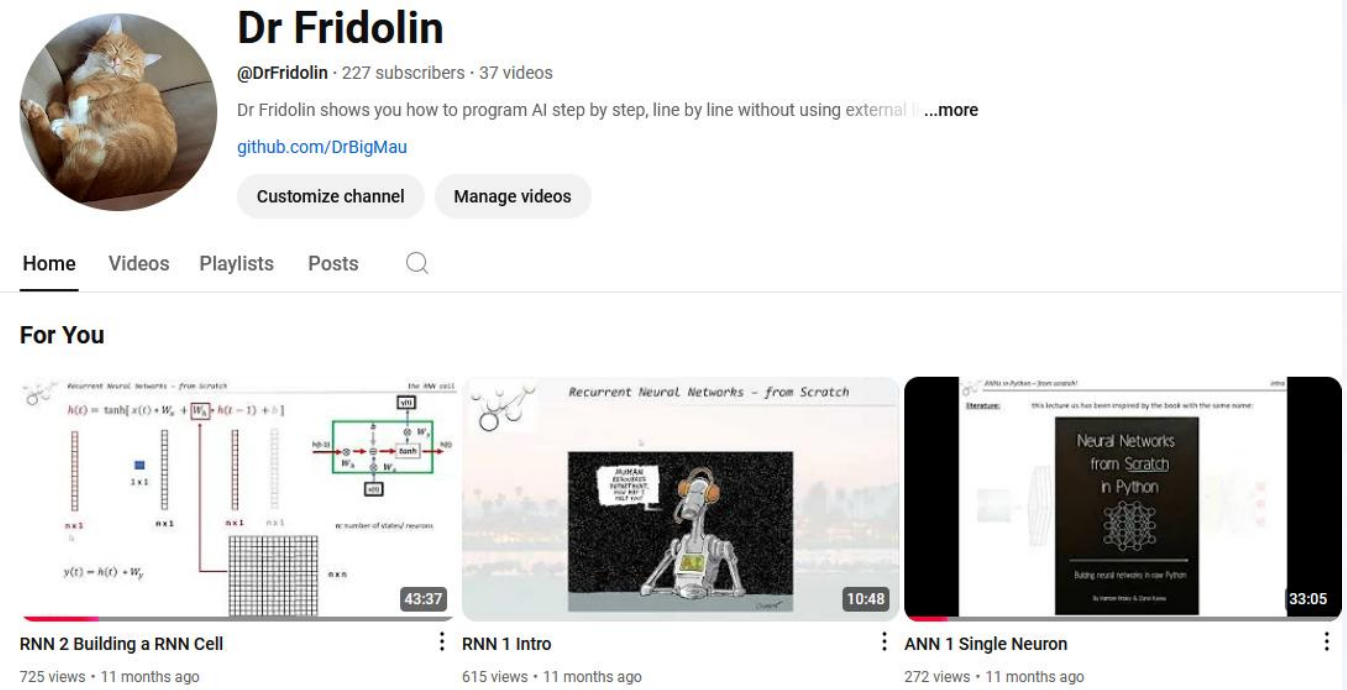


classification/
Regression, **prediction**



Tutorials and Books



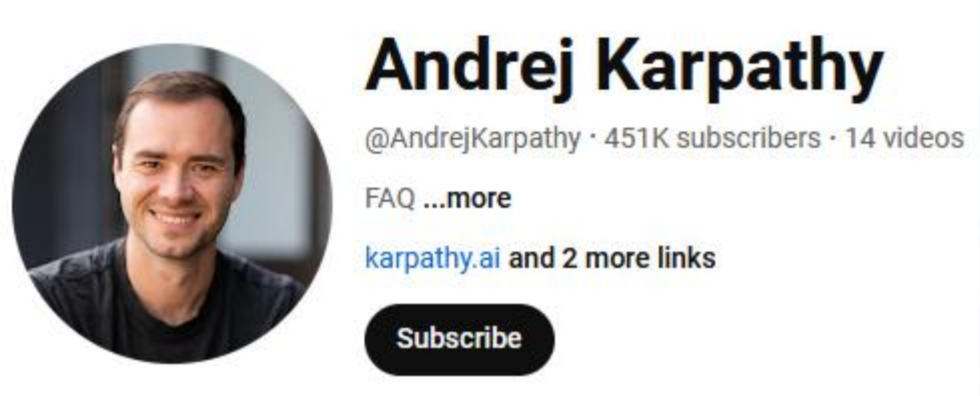


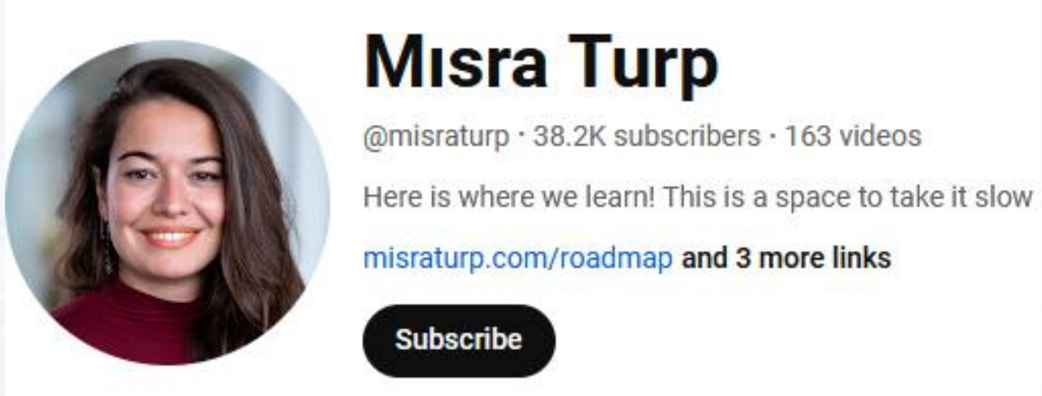
Tutorials and Books

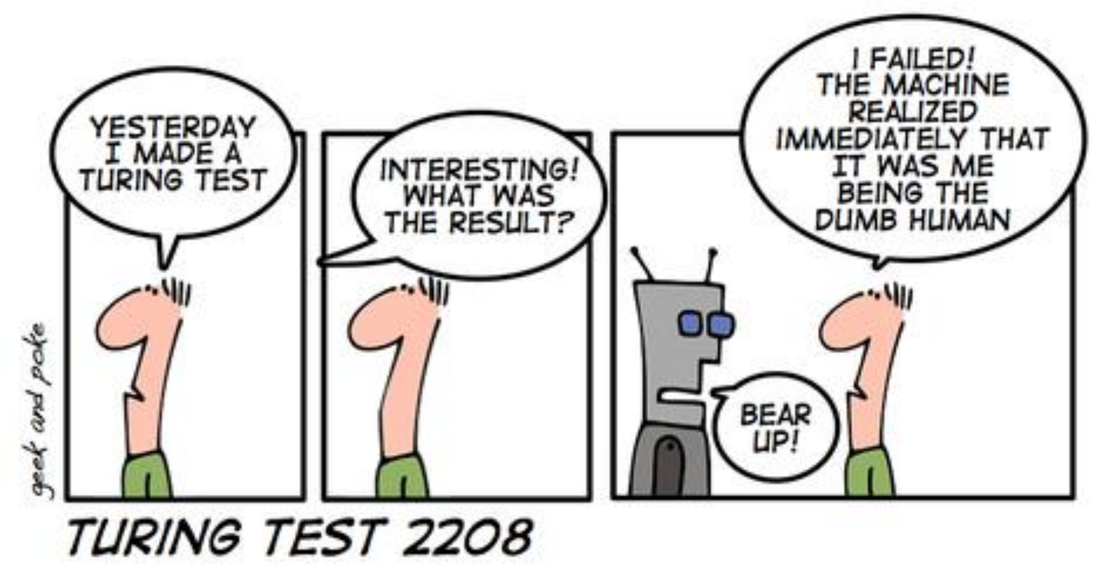
examples and **application** (CNN/RNN/LSTM and more):

Jason Brownlee
Machine Learning Mastery

all about **transformers, LLM/NLP and way more**



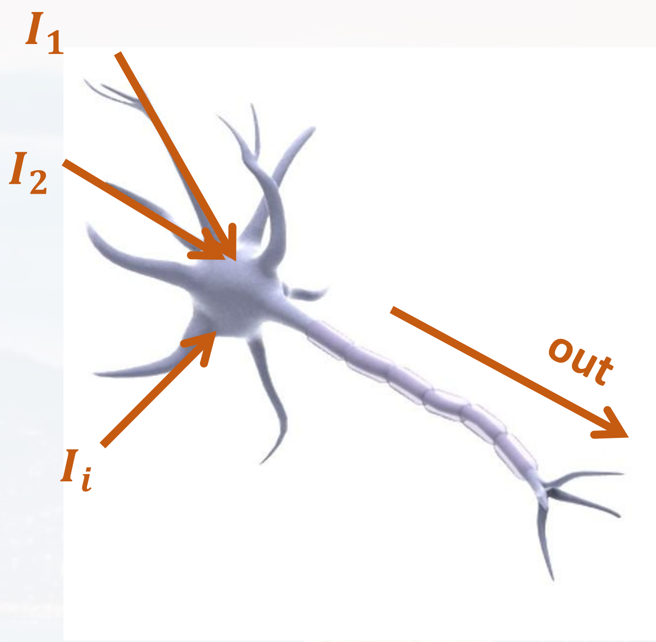




Outline

- Motivation & Overview
- **The Perceptron aka Neuron**
- Building a Perceptron

the perceptron:



I_i input *i*

w_i corresponding weight

b bias (base potential)

what we know…

- inputs enter the neuron
- something happens inside the neuron
- neuron generates an output

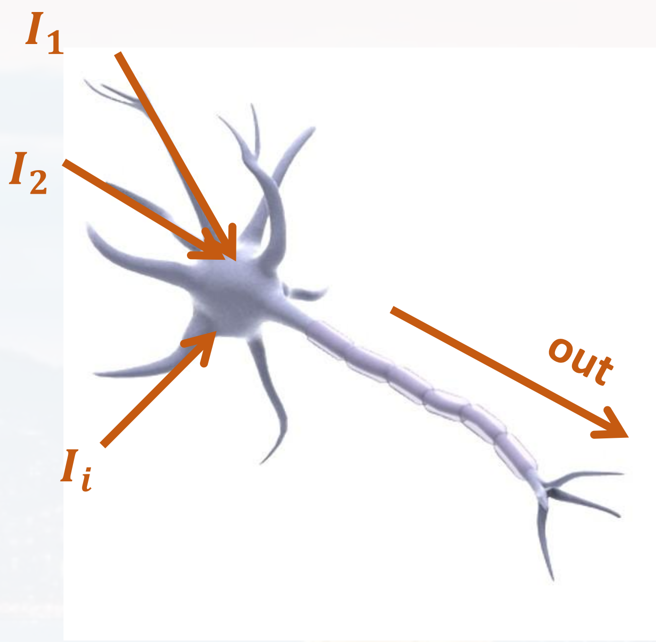
…how we could model it

- inputs must be weighted (important vs unimportant input)
- **learning process: changing weights**
- output = sum of weighted inputs

$$\boxed{net = \underbrace{\sum_i I_i \cdot w_i}_{\textit{dot product}} + b}$$

ANN - Intro: The Perceptron

the perceptron:



$$\boxed{net = \underbrace{\sum_i I_i \cdot w_i}_{\textbf{dot product}} + b}$$

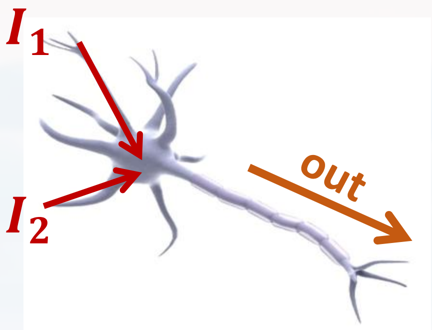
I_i **input *i***
w_i **corresponding weight**
b **bias (base potential)**

recall: linear models

$$y_k = \beta_0 + \sum_{n=1}^{N} \beta_n x_n + \epsilon$$

$$\begin{pmatrix} y_1 \\ \dots \\ y_k \\ \dots \\ \dots \\ y_K \end{pmatrix} = \begin{pmatrix} 1 & x_{11} & x_{12} & \dots & x_{1n} & \dots & x_{1N} \\ \dots & \dots & & & \dots & & \\ 1 & x_{k1} & & & x_{kn} & & \\ 1 & \dots & & & \dots & & \\ 1 & x_{K1} & x_{K2} & \dots & x_{Kn} & \dots & x_{KN} \end{pmatrix} \begin{pmatrix} \beta_0 \\ \beta_1 \\ \dots \\ \beta_n \\ \dots \\ \beta_N \end{pmatrix} + \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \dots \\ \varepsilon_k \\ \dots \\ \varepsilon_K \end{pmatrix}$$

the perceptron:



$$\boxed{net = \underbrace{\sum_i I_i \cdot w_i}_{\textit{dot product}} + b}$$

I_i	**input *i***
w_i	**corresponding weight**
b	**bias (base potential)**

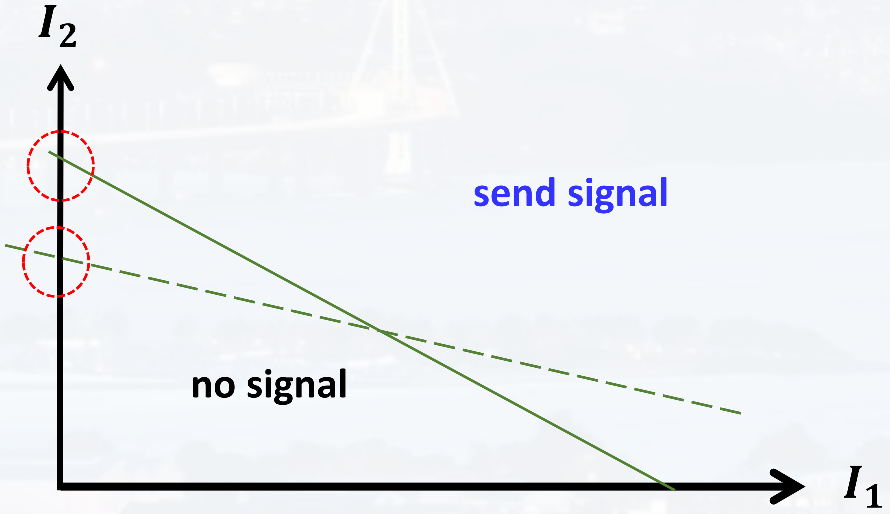
simple example:

one neuron with a **switch**, threshold **T** and **two** input channels

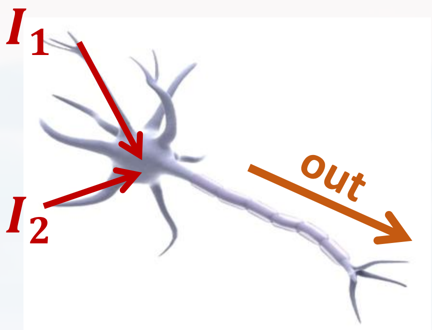
fire if: $b + I_1 w_1 + I_2 w_2 > T$

$$I_2 = -\frac{w_1}{w_2} I_1 + \frac{T-b}{w_2}$$

slope offset



the perceptron:

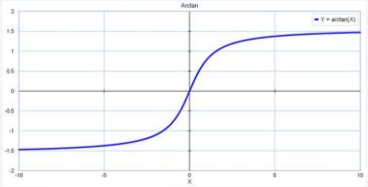


$$net = \sum_i I_i \cdot w_i + b$$

$\boldsymbol{I_i}$ **input *i***
$\boldsymbol{w_i}$ **corresponding weight**
$\boldsymbol{b}$ **bias (base potential)**

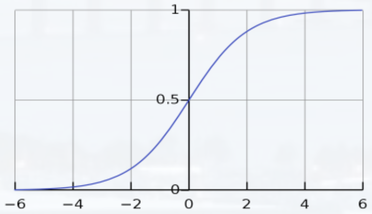
net → some activation function ***f*** → actual output

- y = arctan(net)



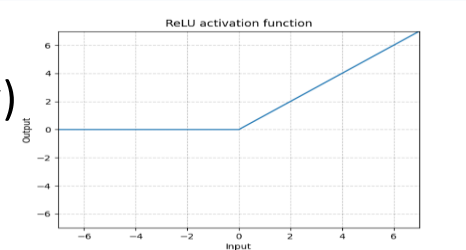
$(-\infty;\ +\infty) \rightarrow (-\pi/2;\ +\pi/2)$

- y = sigm(net)



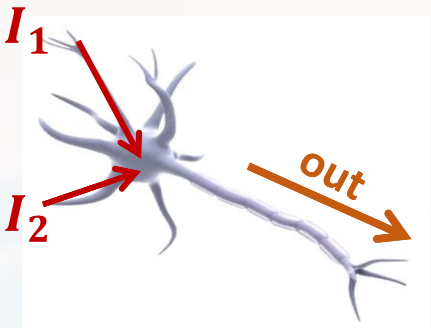
$(-\infty;\ +\infty) \rightarrow (0; 1)$

- y = ReLU(net) = *max(0, net)*



$(-\infty;\ +\infty) \rightarrow (0; +\infty)$

the perceptron:



$$\boxed{net = \sum_i I_i \cdot w_i + b}$$

$\boldsymbol{I_i}$	**input *i***
$\boldsymbol{w_i}$	**corresponding weight**
$\boldsymbol{b}$	**bias (base potential)**
$\boldsymbol{f}$	**activation function**

$$y = f(net)$$

learning:

$$net = \sum_i I_i \cdot w_i + b \xrightarrow{\text{activation function } f} y = f(net) \xrightarrow{\text{objective function } E} E = \frac{1}{2}(t - y)^2$$

target output ***t***

finding best w_i by **minimizing** *E*

$$\boldsymbol{\Delta w_i = w_i(new) - w_i(old) = -\alpha \frac{\partial E}{\partial w_i}}$$

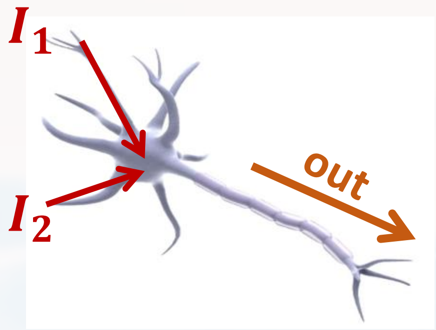
gradient descent with learning rate $\boldsymbol{\alpha}$

$$\boldsymbol{\frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial net}\frac{\partial net}{\partial w_i}}$$

chain rule

ANN - Intro: The Perceptron

the perceptron:



$$\boxed{net = \sum_i I_i \cdot w_i + b}$$

I_i	**input *i***
w_i	**corresponding weight**
b	**bias (base potential)**
f	**activation function**

$$y = f(net)$$

learning:

$$net = \sum_i I_i \cdot w_i + b \xrightarrow{\text{activation function } f} y = f(net) \xrightarrow{\text{objective function } E} E = \frac{1}{2}(t - y)^2$$

target output ***t***

finding best w_i by **minimizing** E

$$\Delta w_i = w_i(new) - w_i(old) = -\alpha \frac{\partial E}{\partial w_i}$$

gradient descent with learning rate α

$$\frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial net}\frac{\partial net}{\partial w_i} = -(t - y)\, f'(net)\, I_i$$

learning:

activation function ***f*** objective function ***E***

$$net = \sum_i I_i \cdot w_i + b \longrightarrow y = f(net) \longrightarrow E = \frac{1}{2}(t - y)^2$$

target output ***t***

finding best w_i by **minimizing** E

$$\Delta \boldsymbol{w_i} = \boldsymbol{w_i}(\boldsymbol{new}) - \boldsymbol{w_i}(\boldsymbol{old}) = -\boldsymbol{\alpha}\frac{\partial \boldsymbol{E}}{\partial \boldsymbol{w_i}}$$

gradient descent with learning rate ***α***

$$\frac{\partial \boldsymbol{E}}{\partial \boldsymbol{w_i}} = \frac{\partial \boldsymbol{E}}{\partial \boldsymbol{y}}\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{net}}\frac{\partial \boldsymbol{net}}{\partial \boldsymbol{w_i}} = \boxed{-(\boldsymbol{t} - \boldsymbol{y})\ \boldsymbol{f}'(\boldsymbol{net})}\ \boxed{\boldsymbol{I_i}}$$

inner derivative

outer derivatives

$\boldsymbol{f}'(\boldsymbol{net})$ depends on the activation function

The required change of ***E*** propagates back to the changes of $\boldsymbol{w_i}$ → **backpropagation** learning:

activation function $\boldsymbol{f}$ — objective function $\boldsymbol{E}$

$$net = \sum_i I_i \cdot w_i + b \longrightarrow y = f(net) \longrightarrow E = \frac{1}{2}(t - y)^2$$

target output $\boldsymbol{t}$

finding best w_i by **minimizing** E

$$\Delta \boldsymbol{w_i} = \boldsymbol{w_i(new) - w_i(old) = -\alpha \frac{\partial E}{\partial w_i}}$$

gradient descent with learning rate $\boldsymbol{\alpha}$

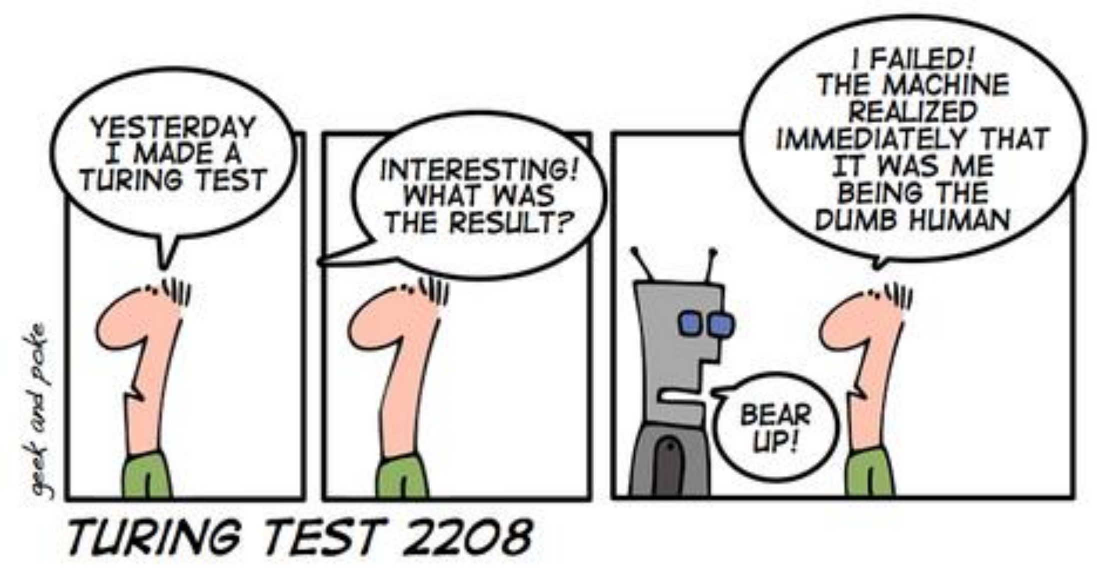
$$\boldsymbol{\frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial net}\frac{\partial net}{\partial w_i} = \boxed{-(t-y)\ f'(net)}\,\boxed{I_i}}$$

inner derivative — outer derivatives

The required change of $\boldsymbol{E}$ propagates back to the changes of $\boldsymbol{w_i}$ → **backpropagation**

$$\boxed{\Delta \boldsymbol{w_i = w_i(new) - w_i(old) = \alpha\,(t-y)\ f'(net)\ I_i}}$$

$$\boxed{\Delta \boldsymbol{b = b(new) - b(old) = \alpha\,(t-y)\ f'(net)\ \cdot 1}}$$



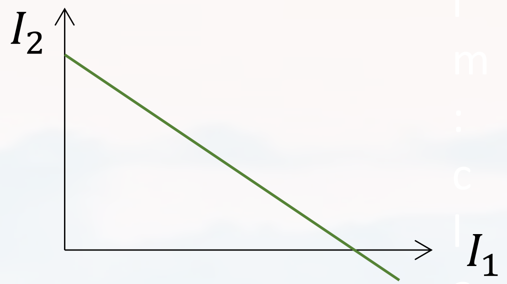
Outline

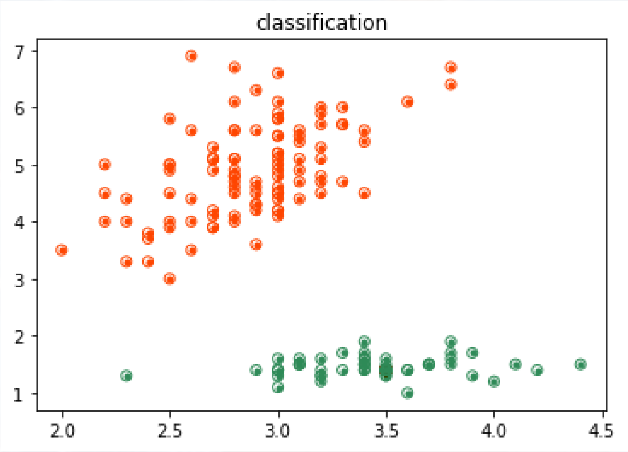
- Motivation & Overview

- The Perceptron aka Neuron

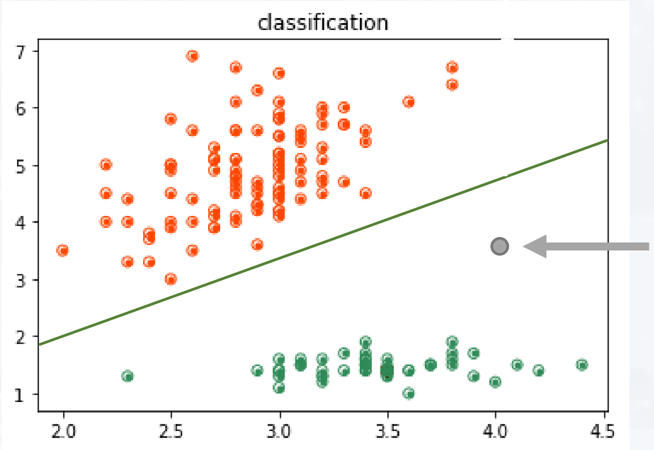
- Building a Perceptron

one neuron
two features
two classes
N data points





goal



classifying a new data point

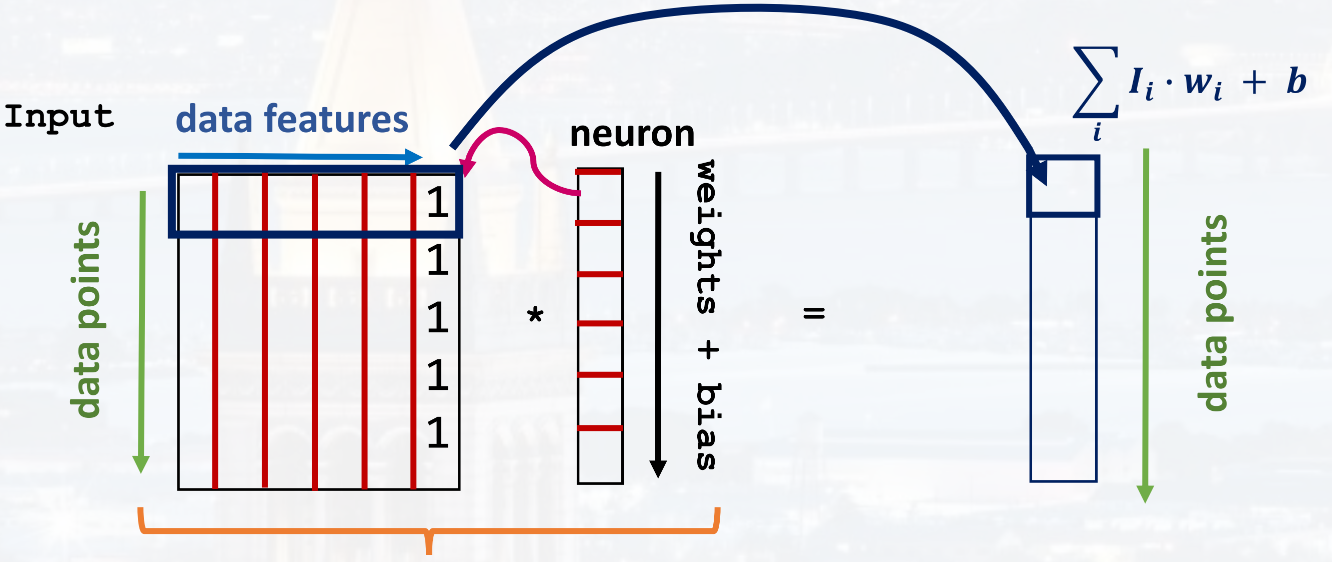
see the Jupyter Notebook `Perceptron.ipynb` for details main part of the code:

data features

data points

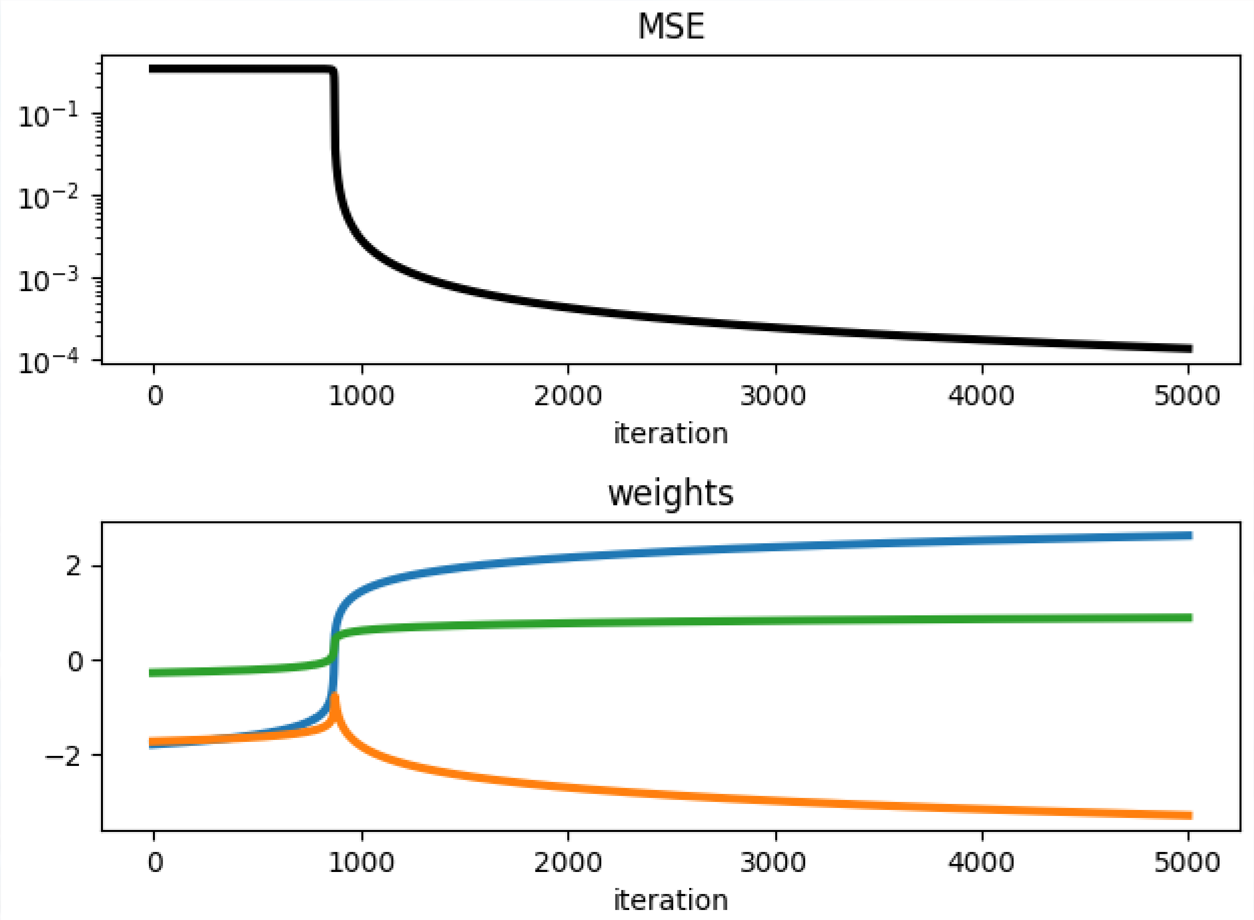
```
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
```

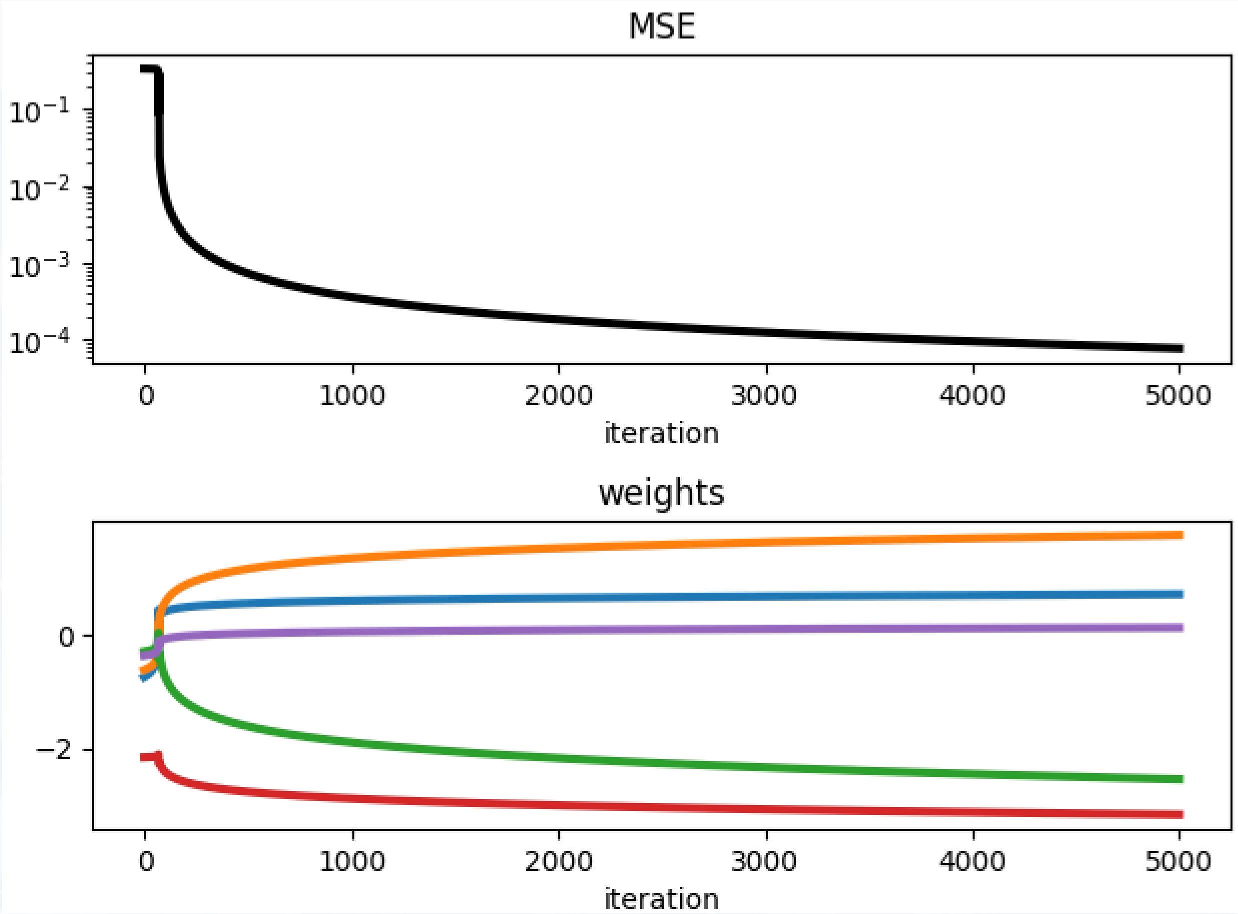
$$\boxed{net = \underbrace{\sum_i I_i \cdot w_i}_{\text{dot product}} + b}$$



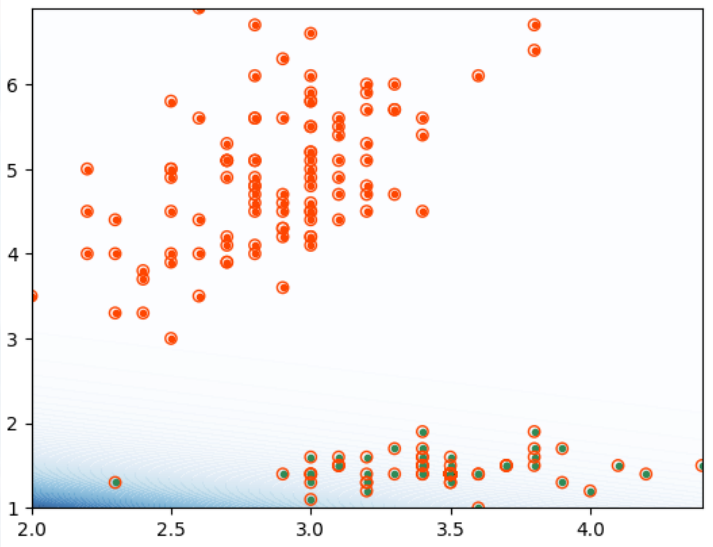
np.dot(Input, W)

The training process

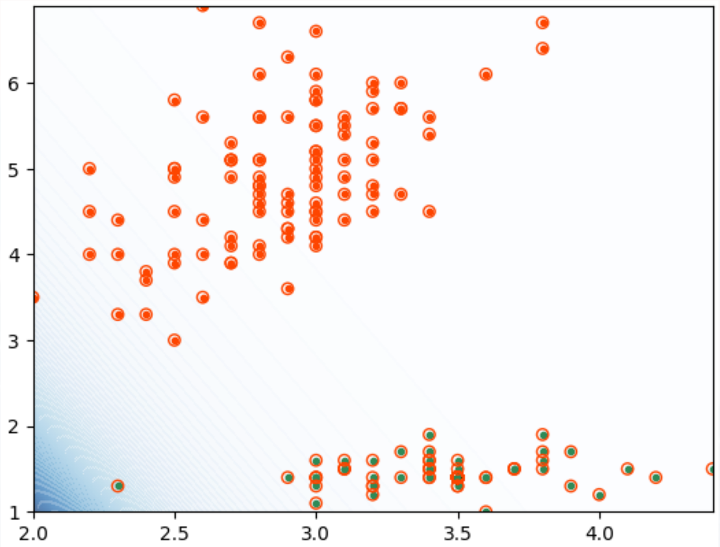




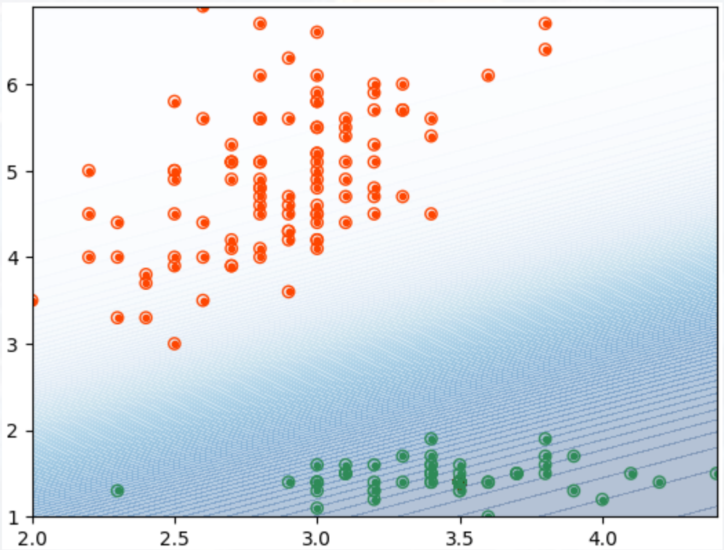
The training process



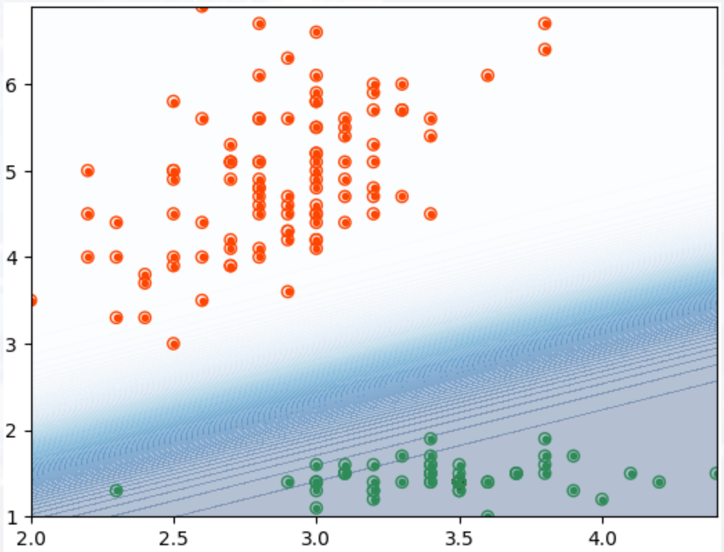
t = 3



t = 5



t = 100



t = 10,000

Thank you for your attention!