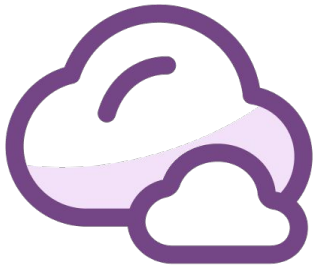
Data Science Survival Skills

Artificial Intelligence, Machine Learning, Deep Learning

slido



What is artificial intelligence for you?

ⓘ Click **Present with Slido** or install our Chrome extension to activate this poll while presenting.

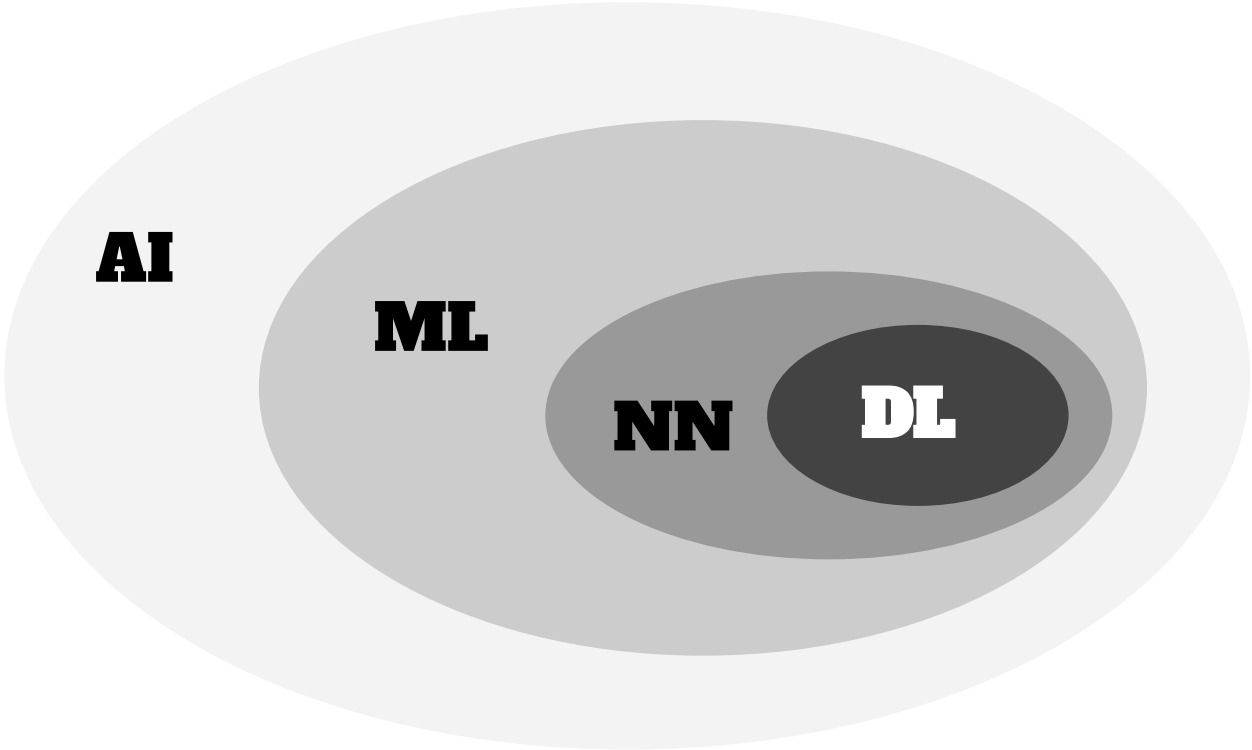
Definition(s) of AI

“Artificial intelligence (AI) is intelligence—perceiving, synthesizing, and infering information—demonstrated by machines, as opposed to intelligence displayed by animals and humans.” (Wikipedia)

“the theory and development of computer systems able to perform tasks that normally require human intelligence, such as visual perception, speech recognition, decision-making, and translation between languages.” (Oxford English Dictionary)

“Artificial intelligence (AI) refers to systems that display intelligent behaviour by analysing their environment and taking actions – with some degree of autonomy – to achieve specific goals. AI-based systems can be purely software-based, acting in the virtual world (e.g. voice assistants, image analysis software, search engines, speech and face recognition systems) or AI can be embedded in hardware devices (e.g. advanced robots, autonomous cars, drones or Internet of Things applications).” (INDEPENDENT HIGH-LEVEL EXPERT GROUP ON ARTIFICIAL INTELLIGENCE, European Commission)

Wait! Aren't we doing AI? Or Deep Learning?



What is intelligence?

Computer
(game) bots

pages. The one exception was in a list of frequently asked questions, one of which was, "Does Deep Blue use artificial intelligence?" Even more surprisingly, IBM's answer to this question was "no"! (IBM, 1997)

ChatGPT

Artificial Intelligence

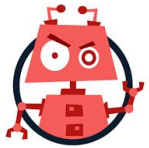
Narrow (ANI)

Weak

ChatGPT

SPECIFIC TASK

-- NOW --



General (AGI)

Strong

HUMAN-LI

-- FUTURE

Q* Learning?
Next OpenAI
breakthrough
?!?!?

Super

Strong

TRANSCENDENT

?????

ChatGPT

slido

When do we achieve AGI?

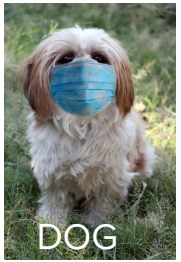
ⓘ Click **Present with Slido** or install our Chrome extension to activate this poll while presenting.

How can machines learn?
Here the most common ones.

Learning = not explicitly programed !!!

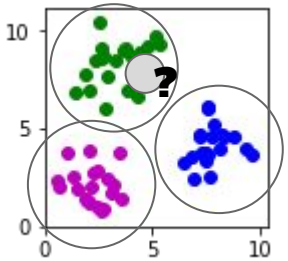
Supervised

e.g. classification, labelled data



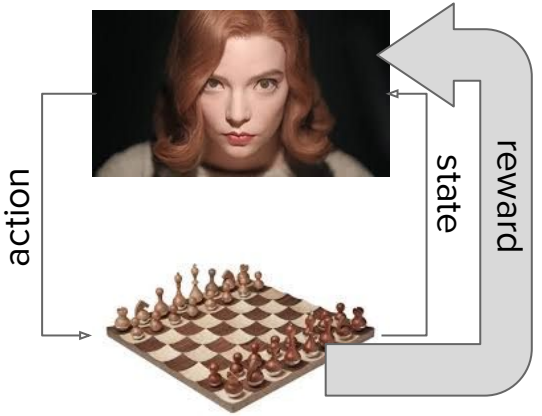
Unsupervised

e.g. clustering, unlabelled data

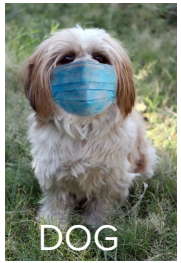


Reinforcement

e.g. complex games, like chess



Supervised learning



Training data + labels

Machine Learning Algorithm

New data

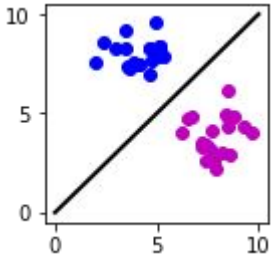
Predicitive Model

Prediction

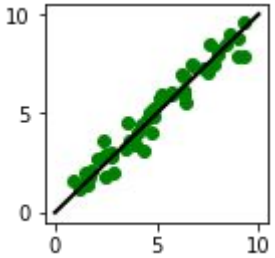
DOG

Two main types of problems (supervised learning)

CLASSIFICATION



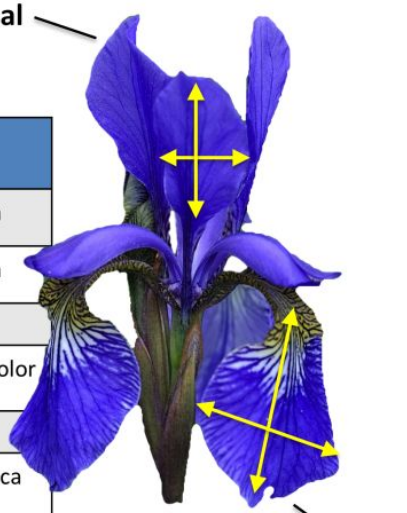
REGRESSION



Terminology

Samples
(instances, observations)

	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
...					
50	6.4	3.5	4.5	1.2	Versicolor
...					
150	5.9	3.0	5.0	1.8	Virginica

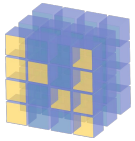


Features
(attributes, measurements, dimensions)

Class labels
(targets)

Math and I/O packages

Processing



NumPy

Multi-dimensional image processing (scipy.n ge)¶

pandas

Reading/Saving

imageio - Python library for reading and writing image data

scikit-image
image processing in python

flammkuchen

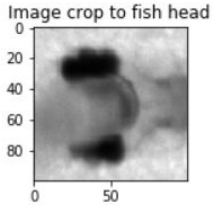
pandas

Plotting

matplotlib

seaborn: statistic data visualization

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  %matplotlib inline
4  import imageio as io
5  # ImageIO to load an image (e.g. training data)
6  im = io.imread("zebrafish.png")
7  # Numpy to process image (e.g. crop)
8  im_crop = im[220:320, 100:200]
9  # Matplotlib to e.g. show data or training progress
10 plt.figure(figsize=(2,2))
11 plt.imshow(im_crop, cmap='gray')
12 plt.title("Image crop to fish head");
```



Machine learning packages (the big ones)

PCA
ICA
Linear Regression
SVMs
...

Optimizer
Filtering
Stats
...

Neural networks etc

Machine learning packages (the new kids on the block)

JAX: Autograd and XLA

Continuous integration passing | pypi v0.2.9

Quickstart | Transformations | Install guide | Neural net libraries | Change logs | Reference docs | Code search

News: JAX tops largest-scale MLPerf Training 0.7 benchmarks!

What is JAX?

JAX is Autograd and XLA, brought together for high-performance machine learning research.

Flax: A neural network library and ecosystem for JAX designed for flexibility

Build passing | coverage 81.6%

Overview | Quick install | What does Flax look like? | Documentation

See our full documentation to learn everything you need to know about Flax.

Flax was originally started by engineers and researchers within the Brain Team in Google Research (in close collaboration with the JAX team), and is now developed jointly with the open source community.

Haiku: Sonnet for JAX

Overview | Why Haiku? | Quickstart | Installation | Examples | User manual | Documentation | Citing Haiku

pytest passing

What is Haiku?

> Haiku is a tool
> For building neural networks
> Think: "Sonnet for JAX"

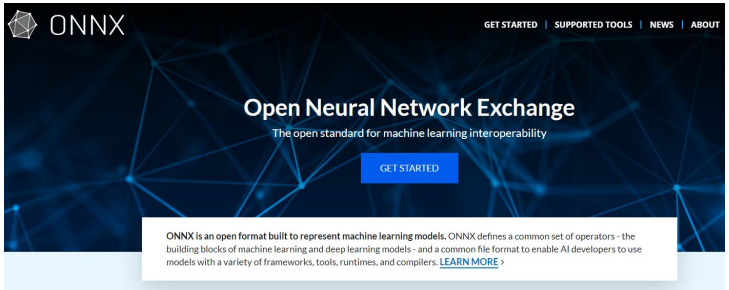
Haiku is a simple neural network library for JAX developed by some of the authors of Sonnet, a neural network library for TensorFlow.

Acme: a research framework for reinforcement learning

Overview | Installation | Documentation | Agents | Examples | Paper | Blog post

python 3.6 | 3.7 | pypi package 0.2.0 | pytest failing

Acme is a library of reinforcement learning (RL) agents and agent building blocks. Acme strives to expose simple, efficient, and readable agents, that serve both as reference implementations of popular algorithms and as strong baselines, while still providing enough flexibility to do novel research. The design of Acme also attempts to provide multiple points of entry to the RL problem at differing levels of complexity.



Published silently this week...

ARTIFICIAL INTELLIGENCE / TECH / APPLE

Apple joins AI fray with release of model framework

Photo by Amelia Holowaty Krales / The Verge

/ Apple's machine learning research team quietly releas framework called MLX to bu foundation models.

By Emilia David, a reporter who covers AI. Prior to joining The Verg the intersection between technology, finance, and the economy.

Dec 6, 2023, 5:51 PM GMT+1 | 13 Comments / 13 New

https://www.theverge.com/2023/12/6/23990678/apple-foundation-models-generative-ai-mlx

README.md

MLX

Quickstart | Installation | Documentation | Examples

PASSED

MLX is an array framework for machine learning on Apple silicon, brought to you by Apple machine learning research.

Some key features of MLX include:

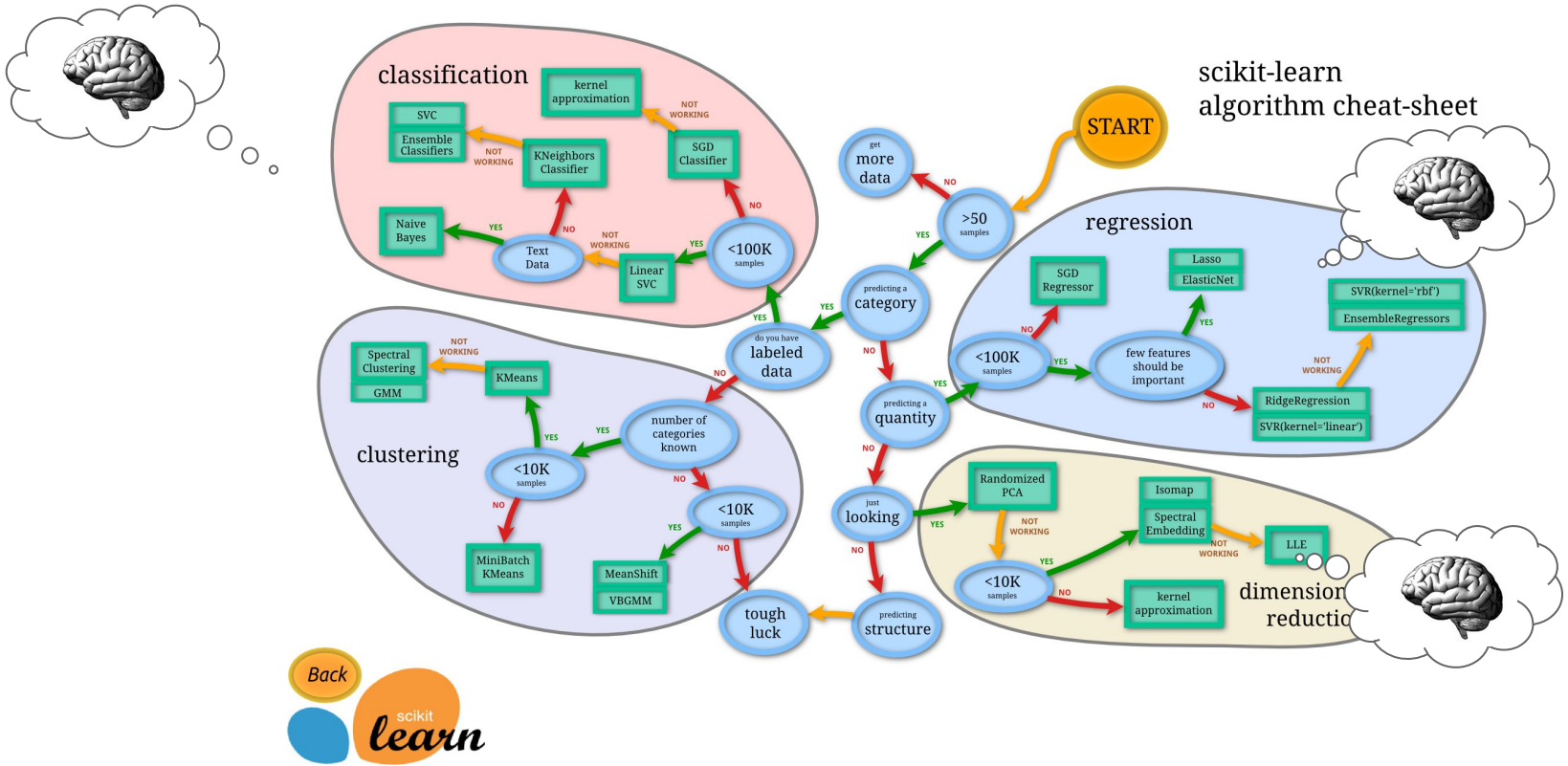
- **Familiar APIs**: MLX has a Python API that closely follows NumPy. MLX also has a fully featured C++ API, which closely mirrors the Python API. MLX has higher-level packages like `mlx.nn` and `mlx.optimizers` with APIs that closely follow PyTorch to simplify building more complex models.
- **Composable function transformations**: MLX has composable function transformations for automatic differentiation, automatic vectorization, and computation graph optimization.
- **Lazy computation**: Computations in MLX are lazy. Arrays are only materialized when needed.
- **Dynamic graph construction**: Computation graphs in MLX are built dynamically. Changing the shapes of function arguments does not trigger slow compilations, and debugging is simple and intuitive.
- **Multi-device**: Operations can run on any of the supported devices (currently, the CPU and GPU).
- **Unified memory**: A notable difference from MLX and other frameworks is the *unified memory model*. Arrays in MLX live in shared memory. Operations on MLX arrays can be performed on any of the supported device types without moving data.

MLX is designed by machine learning researchers for machine learning researchers. The framework is intended to be user-friendly, but still efficient to train and deploy models. The design of the framework itself is also conceptually simple. We intend to make it easy for researchers to extend and improve MLX with the goal of quickly exploring new ideas.

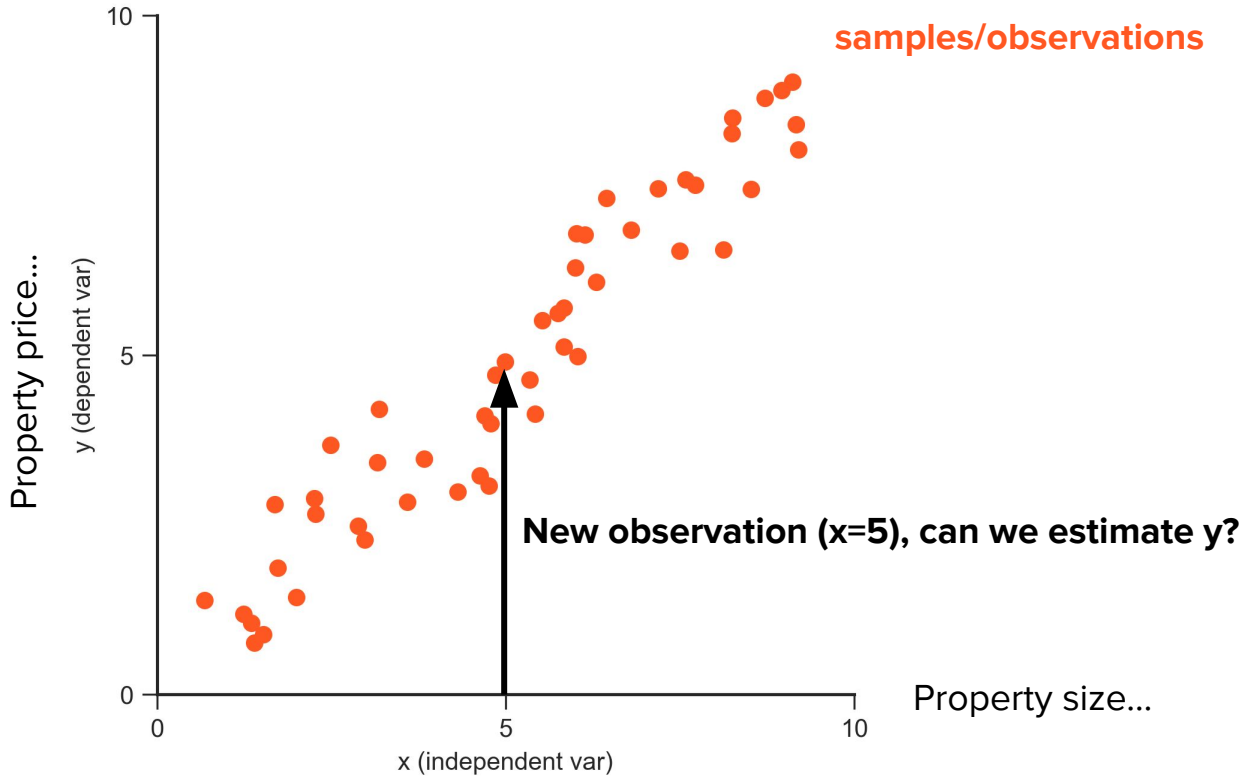
The design of MLX is inspired by frameworks like NumPy, PyTorch, Jax, and ArrayFire.

https://github.com/ml-explore/mlx

Roadmap for machine learning

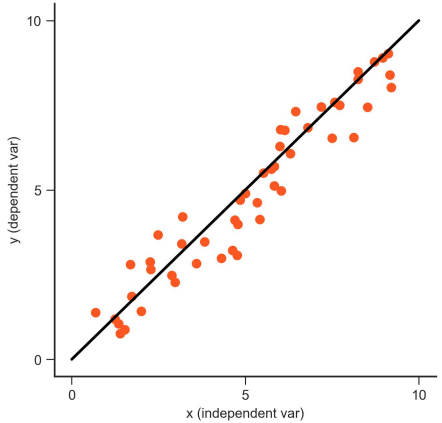


Linear regression

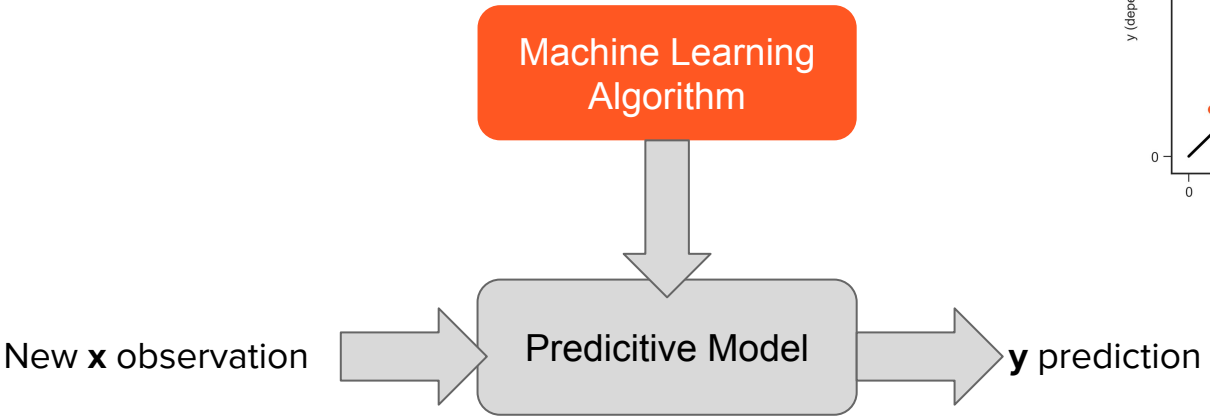


We need a model!

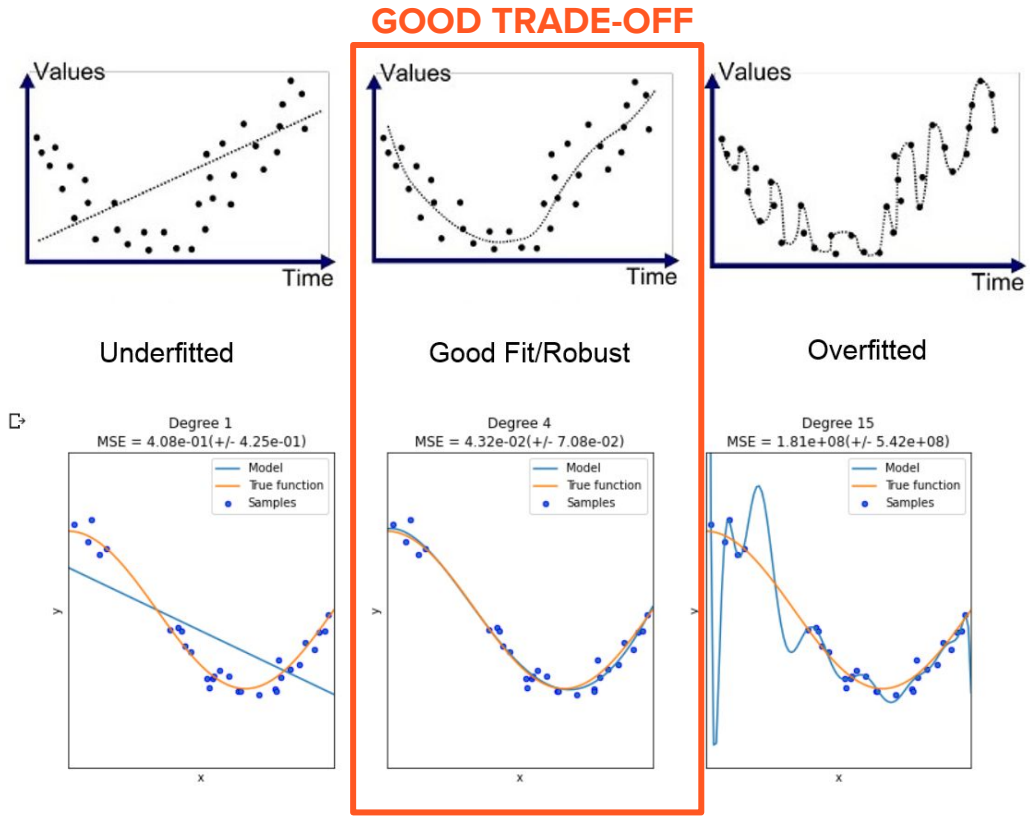
We could use a line as estimation!



All (**x**,**y**) observations



But... why a line?!



Linear regression

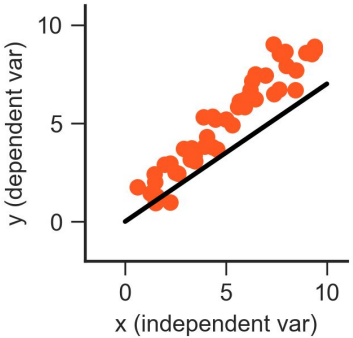
slope intercept

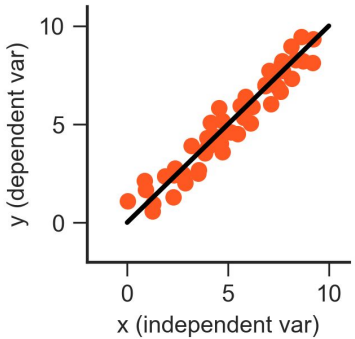
$$y = m * x + b$$

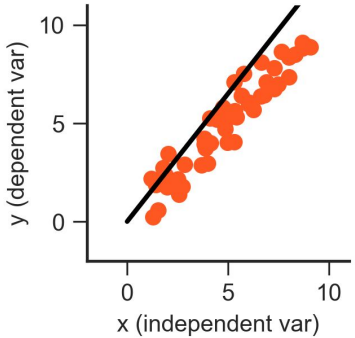
How do we find **m** and **b**?

Or, first of all, what are we aiming for?!?!

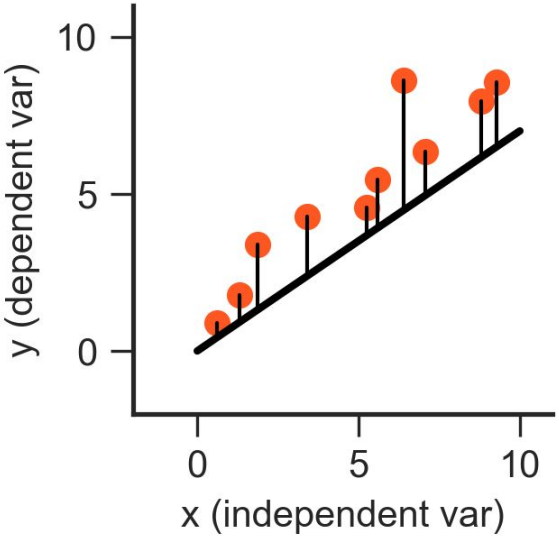
What is a good line?!





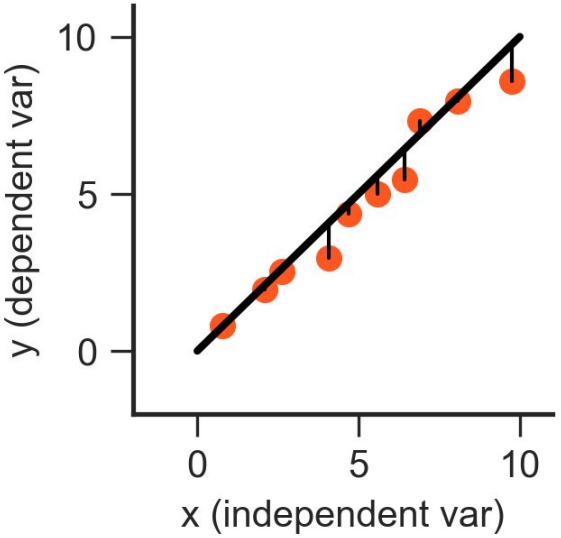


Formalizing our aim



Minimize residuals!

$$\min \sum_{i}^{n} (y_i - \hat{y}_i)^2$$



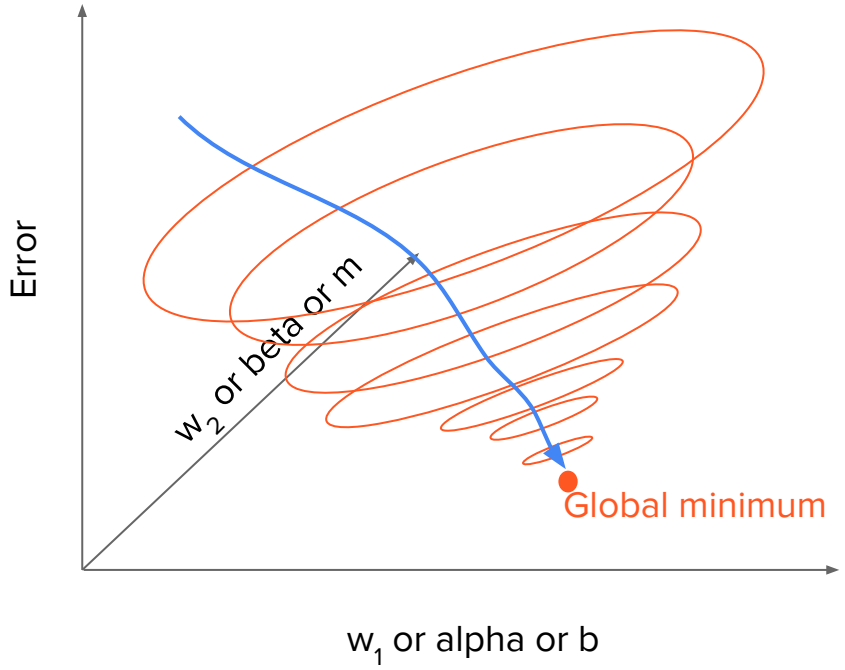
Who is minimizing your residuals?

- Optimization algorithm.
  These ones you should now:
  - Levenberg-Marquardt (damped least-squares, interpolation of Gauss/Newton method + grad. desc)
  - Nelder-Mead (downhill simplex, heuristic for non-linear opt. w/o knowing the derivative)
  - Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm
    (improves a guess of the Hessian matrix, and does not use matrix inversion ➡ O(n^2) vs. O(n^3)

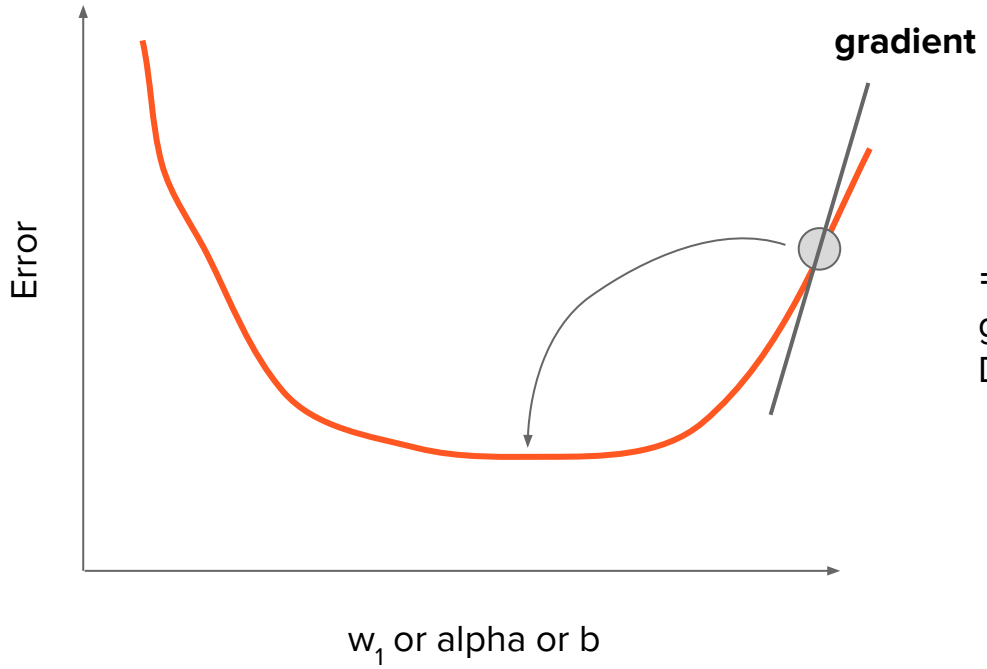
- A general method is **GRADIENT DESCENT.**

Optimization through gradient descent

$$y = m * x + b$$

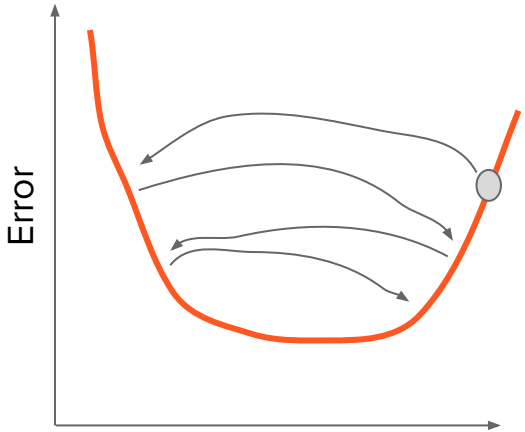


Optimization with gradient descent

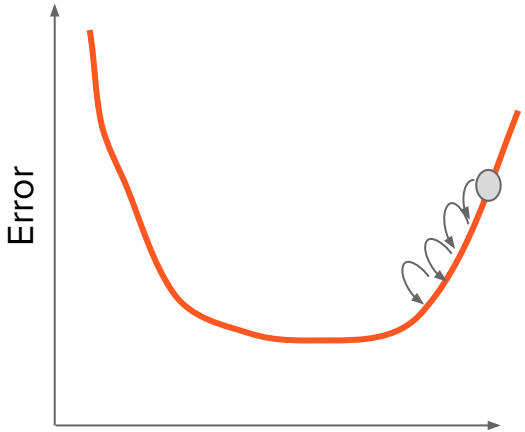


⇒ walk AGAINST the gradient ➔ gradient DESCENT :-)

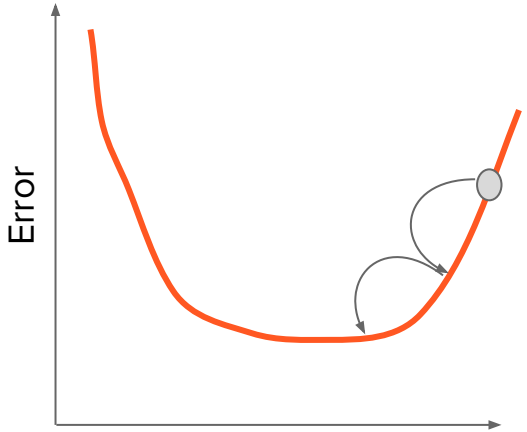
Optimization with gradient descent



Step size TOO LARGE



TOO SMALL



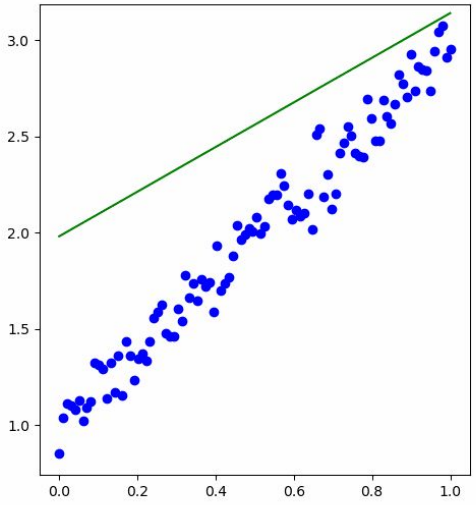
JUST RIGHT

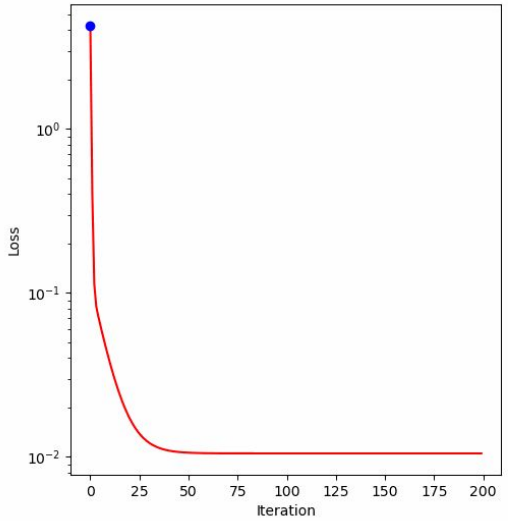
Step size = LEARNING RATE

Computing the gradient

In action

```python
def gradient_descent_linear(x, y, lr=0.1, epochs=100):
    m, b = 0.0, 0.0  # initial parameters
    losses = []
    params = []

    for _ in range(epochs):
        y_pred = m * x + b
        error = y - y_pred
        loss = np.mean(error ** 2)
        losses.append(loss)

        m_grad = -2 * np.mean(x * error)
        b_grad = -2 * np.mean(error)

        m -= lr * m_grad
        b -= lr * b_grad

        params.append((m, b))

    return m, b, losses, params
```

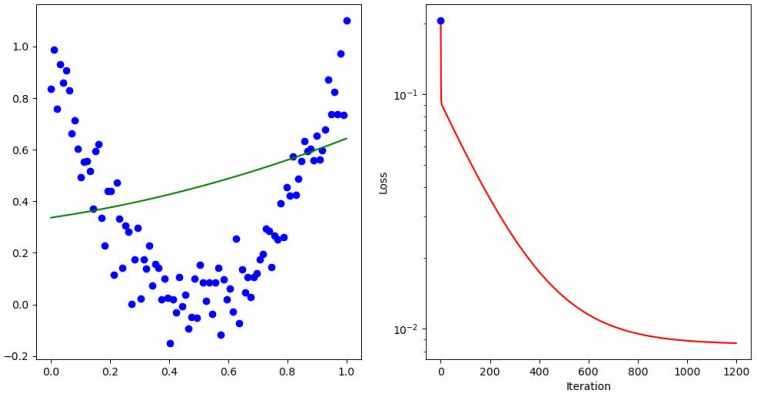




Other data

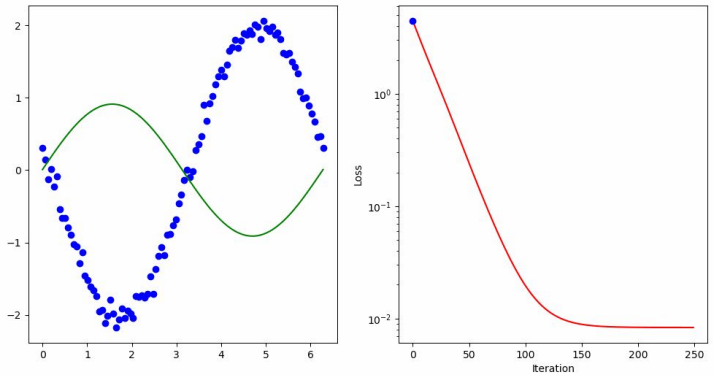
Quadratic

```
y_pred = a * x**2 + b * x + c
error = y - y_pred
loss = np.mean(error ** 2)
losses.append(loss)

a_grad = -2 * np.mean(x**2 * error)
b_grad = -2 * np.mean(x * error)
c_grad = -2 * np.mean(error)
```



Sinusoidal

```
y_pred = a * np.sin(x + b)
error = y - y_pred
loss = np.mean(error ** 2)
losses.append(loss)

a_grad = -2 * np.mean(np.sin(x + b) * error)
b_grad = -2 * np.mean(a * np.cos(x + b) * error)
```



Very short intro to P and NP

P = “polynomial time” - can be **solved** quickly

NP = “non-deterministic polynomial time” - can be **verified** quickly

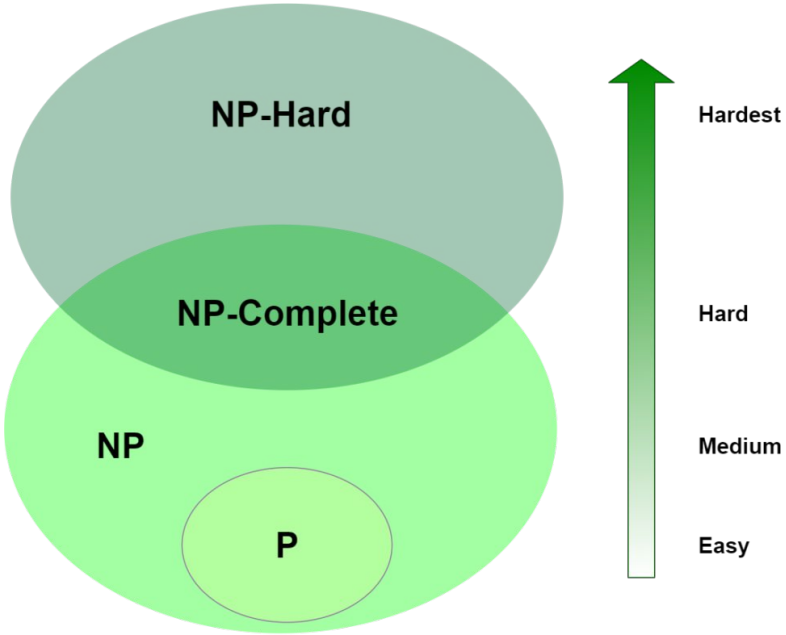
6			3		8			7
			6		9			
		9				4		
8	6						9	1
1	7						8	5
		8				2		
			2		5			
5			9		1			6

It is fast to verify if a solution to Sudoku is correct, however, rather slow to find a proper solution to it. (=> NP, != P)

NP-complete problems are a class of decision problems which contains the hardest problems in NP. Each NP-complete problem has to be in NP. They are fast to verify, and **a bruteforce search can provide you the best result. (Knapsack problem, Traveling salesman)**

NP-hard problems:
Hardest problems can be reduced to sub-problems that can be potentially solved in NP space. Maybe even not decidable.

Short overview



How to solve “NP-hard” problems?

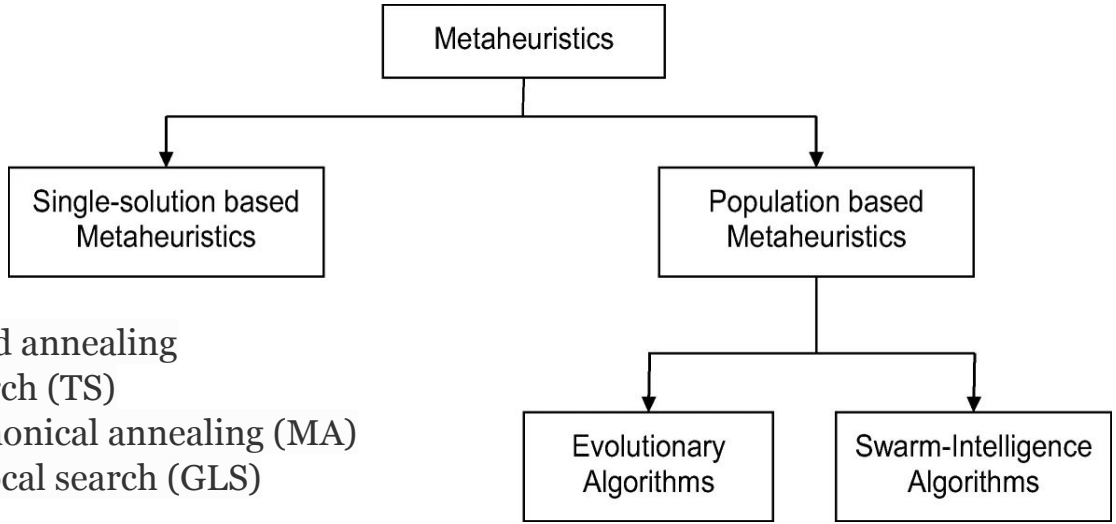
➜ Heuristic search

Trading off

- Optimality
- Completeness
- Accuracy and precision
- Execution time

<u>For example for traveling salesman:</u>
Greedy algorithms could form the following heuristic: "At each step of the journey, visit the nearest unvisited city."
➜ finds a fast, but not necessary the optimal solution (finishes in reasonable amount of steps)

Meta-heuristic algorithms

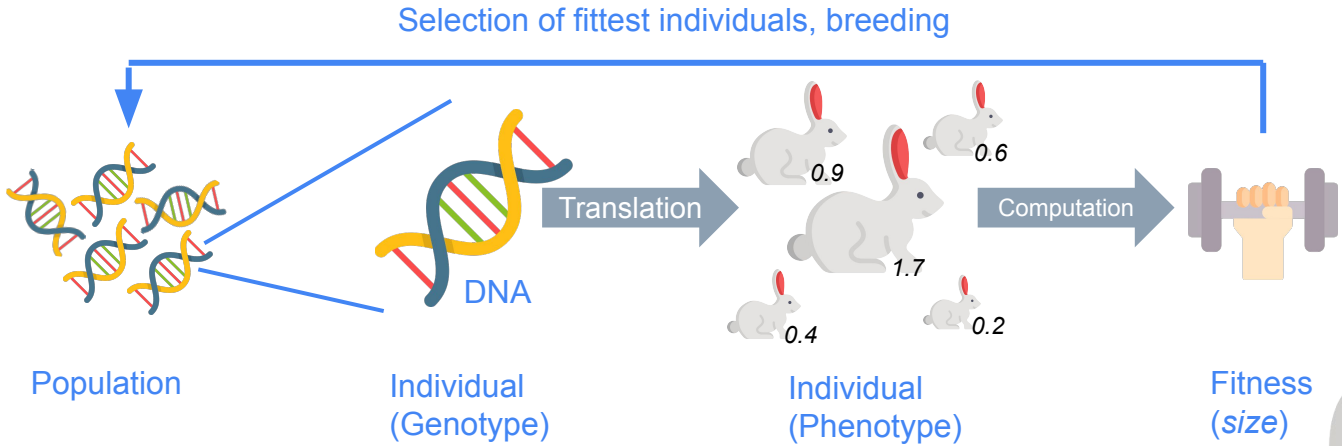


- simulated annealing
- tabu search (TS)
- microcanonical annealing (MA)
- guided local search (GLS)

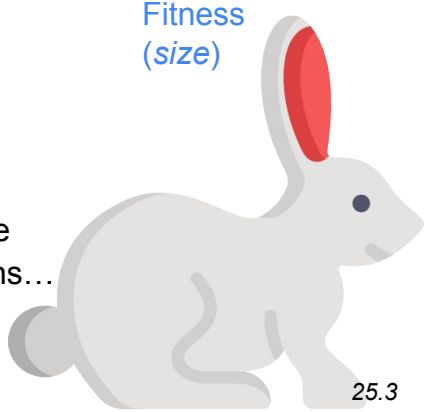
- particle swarm optimization (PSO)
- ant colony optimization (ACO)

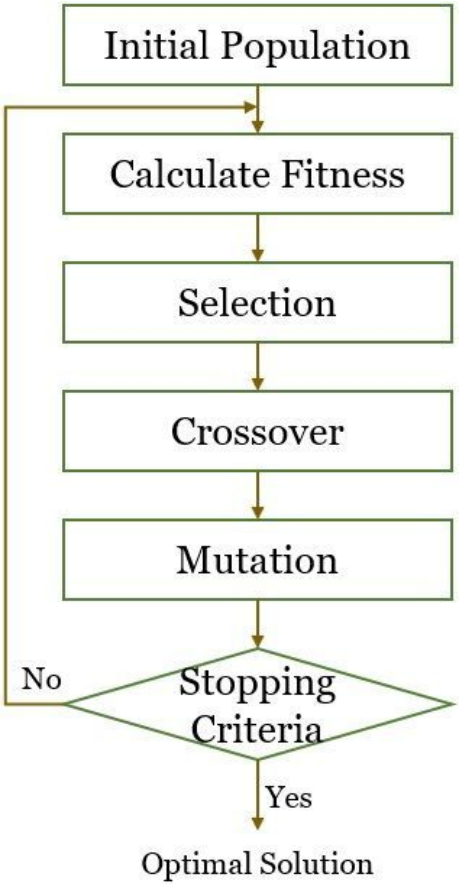
Evolutionary optimization / Genetic algorithms

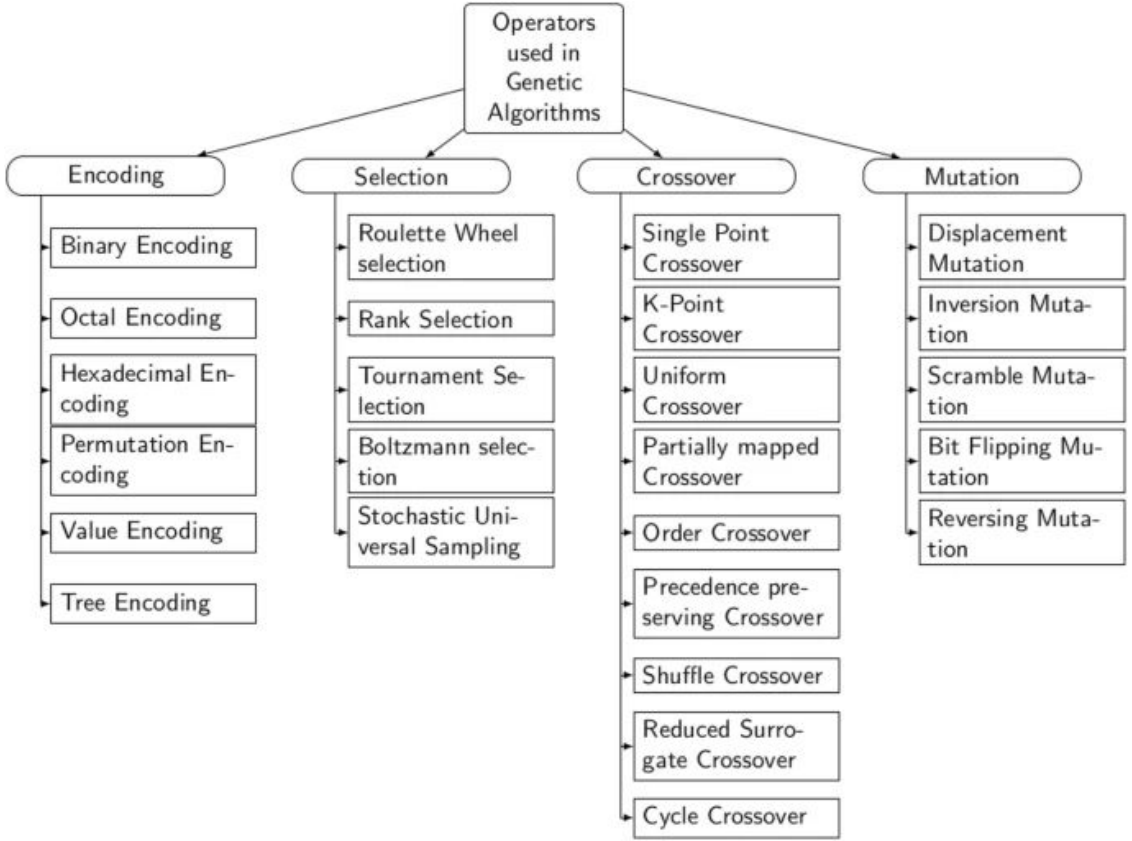
Aim: „Large rabbits“



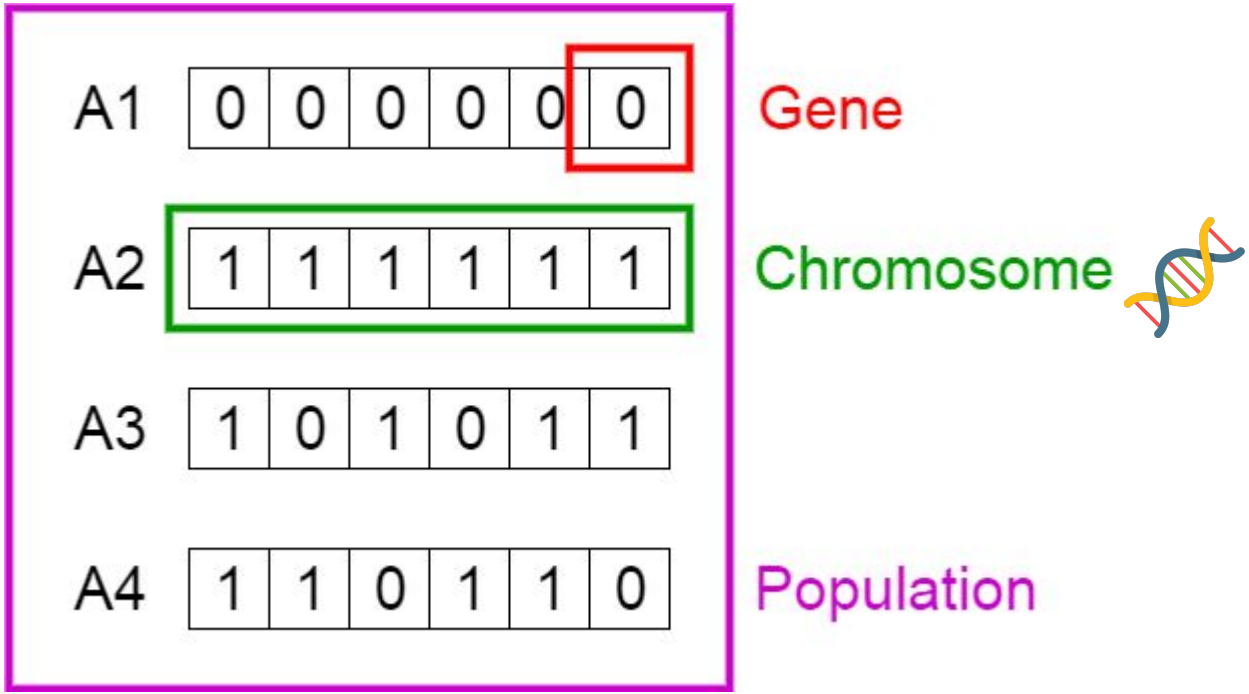
After some generations…




Initial Population
Calculate Fitness
Selection
Crossover
Mutation
Stopping Criteria
No
Yes
Optimal Solution


Operators used in Genetic Algorithms
Encoding
Binary Encoding
Octal Encoding
Hexadecimal Encoding
Permutation Encoding
Value Encoding
Tree Encoding
Selection
Roulette Wheel selection
Rank Selection
Tournament Selection
Boltzmann selection
Stochastic Universal Sampling
Crossover
Single Point Crossover
K-Point Crossover
Uniform Crossover
Partially mapped Crossover
Order Crossover
Precedence preserving Crossover
Shuffle Crossover
Reduced Surrogate Crossover
Cycle Crossover
Mutation
Displacement Mutation
Inversion Mutation
Scramble Mutation
Bit Flipping Mutation
Reversing Mutation

Defining an individual / Encoding



Encoding strategies

1. Genetic Algorithm for Traveling Salesman Problem (Binary Encoding)

- Problem: Find the shortest possible route that visits a set of cities and returns to the origin city.
- Encoding: Each city is assigned a unique binary string. A solution (route) is represented as a sequence of these binary strings.
- Application: This approach is used in logistics and route planning.

2. Genetic Programming for Symbolic Regression (Tree Encoding)

- Problem: Find a mathematical model that best fits a set of data points.
- Encoding: Solutions are encoded as tree structures, where nodes represent mathematical functions and leaves represent variables or constants.
- Application: Used in data analysis and scientific research for model discovery.

3. Interactive Genetic Algorithm for Art and Design (Mixed Encoding)

- Problem: Create appealing designs or artworks, where aesthetic preference is subjective.
- Encoding: Solutions might be encoded as a mix of discrete and continuous parameters defining colors, shapes, and layout.
- Application: Used in fields of art and design where human preferences play a significant role.

4. Co-evolutionary Algorithms for Game Strategy Development (Vector Encoding)

- Problem: Develop winning strategies for games or simulations.
- Encoding: Solutions are encoded as vectors representing different strategy or gameplay elements.
- Application: Used in game development and AI research, especially for developing AI players in complex games.

Code examples

python Copy code

```python
import numpy as np

# Representing a city as a binary string
def city_to_binary(city_index, num_cities):
    return np.binary_repr(city_index, width=int(np.ceil(np.log2(num_cit

# Example for 4 cities
num_cities = 4
cities = [city_to_binary(i, num_cities) for i in range(num_cities)]
print("Binary Encoded Cities:", cities)
```

python Copy code

```python
class Node:
    def __init__(self, value, left=None, right=None):
        self.value = value
        self.left = left
        self.right = right

# Example tree (represents an expression like 'x + 2')
root = Node('+', Node('x'), Node('2'))
```

python Copy code

```python
# Mixed encoding for a design element (color, shape, size)
design_element = {'color': '#ff00ff', 'shape': 'circle', 'size': 10.0}
print("Design Element:", design_element)
```

python Copy code

```python
# Strategy vector for a game
strategy = [1, 0, 1, 1, 0]  # Could represent different tactical choice
print("Game Strategy:", strategy)
```

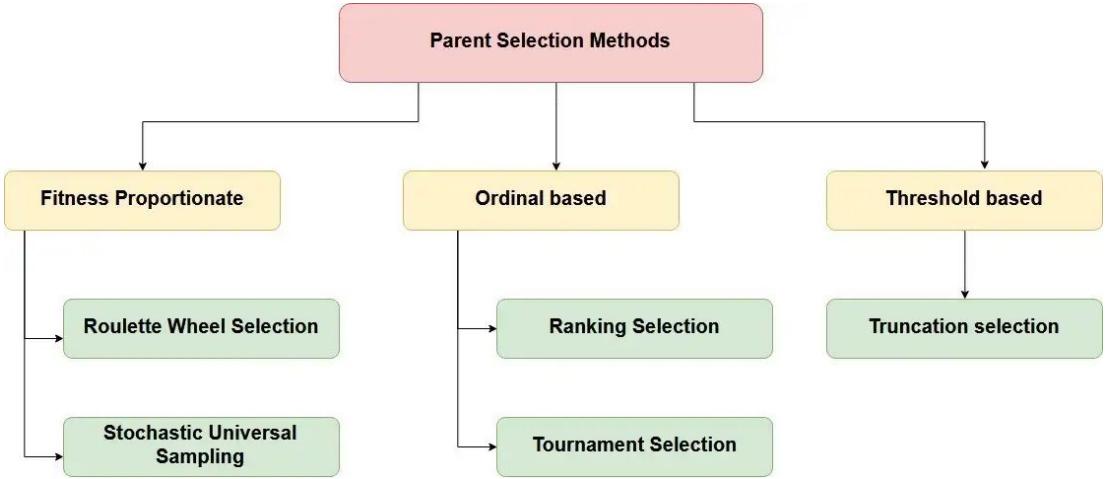
Next generation

Selection - survival of the fittest (Darwin!)

How “fit” is an individual?

- **Unconstrained**: this is the simplest case where the fitness function corresponds to your objective function
- **Constrained**: in this case, your fitness function could be composed of two terms: the original objective and a penalty term (that penalizes solutions that do not satisfy the constraints)
- **Multi-objective**: where you have multiple fitness functions (one for each objective) and the final fitness function is a combination of these multiple fitness functions; (**Pareto-optimality)**
- **Dynamic** (or noisy), where the fitness of solutions can change over time or depend on some noise component (e.g. Gaussian noise)

Selection

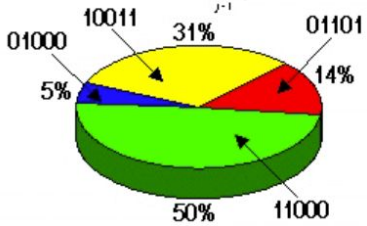


The Problem is to Maximize $f(x) = x^2$

Number	String	Fitness	% of the Total
1	01101	169	14.4
2	11000	576	49.2
3	01000	64	5.5
4	10011	361	30.9
Total		1170	100.0

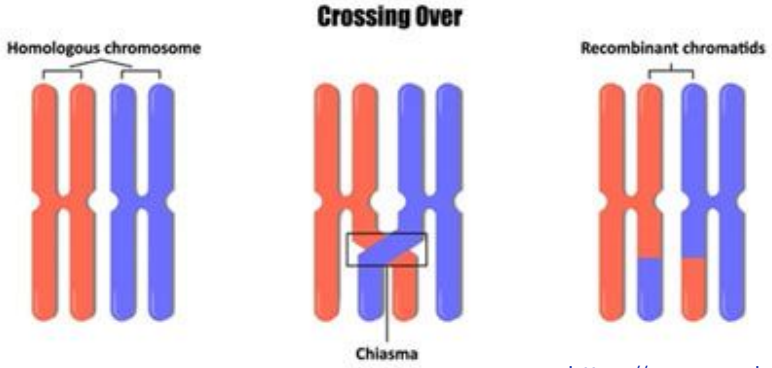
1.- *Roulette Wheel Selection*

$$P_s = \frac{f_i}{\sum_{j=1}^{n} f_j}$$

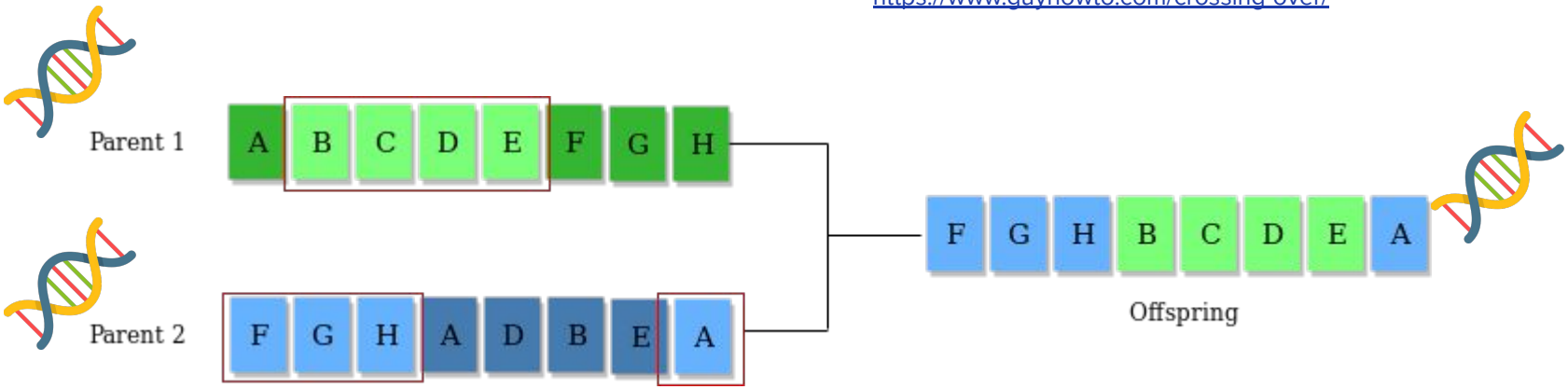


https://neo.lcc.uma.es/TutorialEA/semEC/cap02/cap_2.html

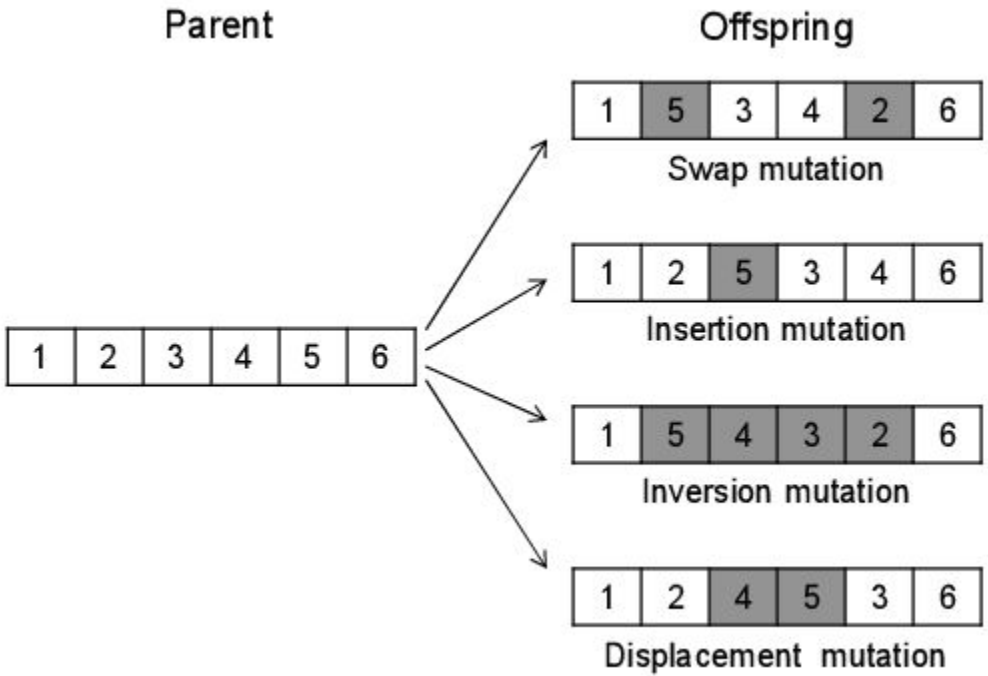
How to generate the next generation



https://www.guyhowto.com/crossing-over/

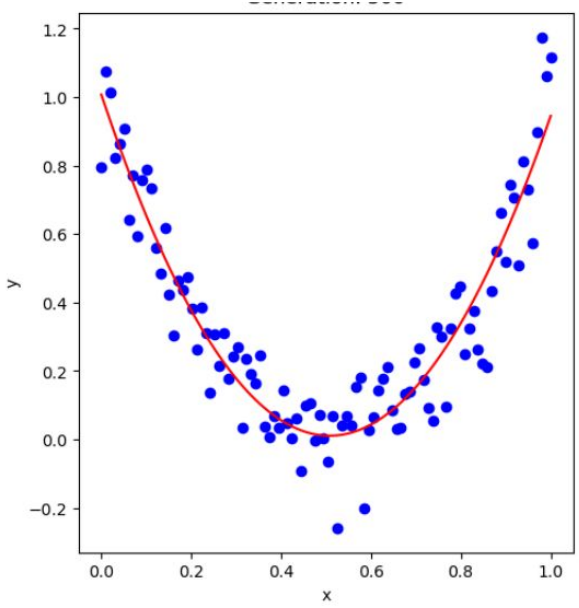


Mutation strategies (examples)



Chun Liu, 2014

Example



I am looking for a function, like the red one, described by $y = a * x^2 + b * x + c$

Fitness: **F** = 1 / mean($(y_{true}-y_{pred})^2$)

What I am looking for? ➜ a, b, c

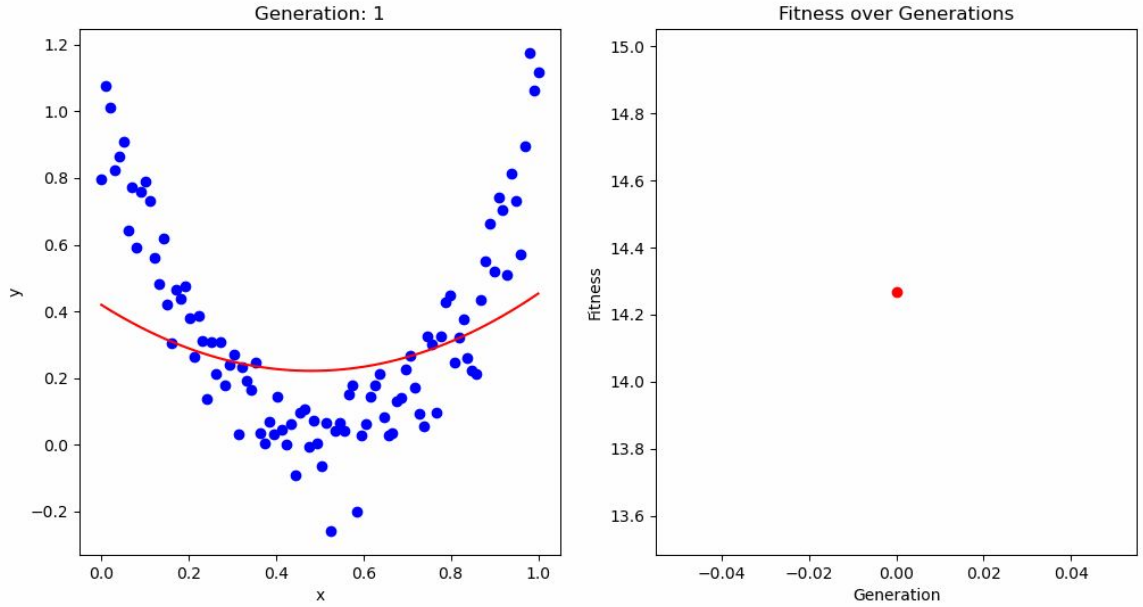
Genotype: [a, b, c]
Phenotype: $a * x^2 + b * x + c$

Genetic pool: $a, b, c \in \mathbb{R}$

Cross-over: e.g. mean of coefficients
Mutation: add random noise, e.g. drawn from normal distribution

Example

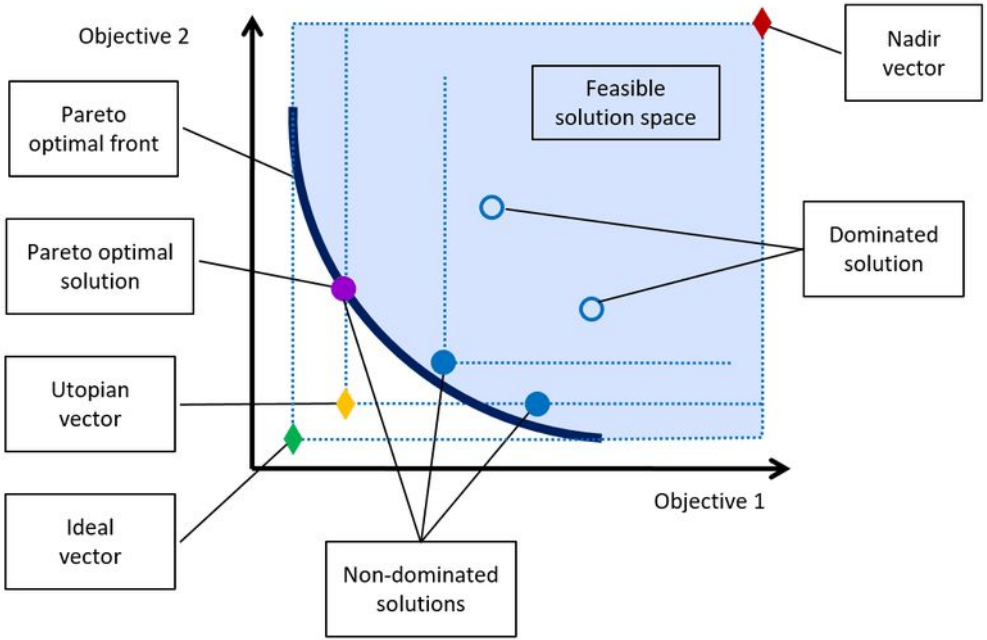
```python
import random

def fitness_function(coeffs, x, y):
    """Calculate the fitness of a solution based on the mean squared error."""
    a, b, c = coeffs
    y_pred = a * x**2 + b * x + c
    return 1 / np.mean((y - y_pred) ** 2)

def crossover(parent1, parent2):
    """Simple crossover by averaging coefficients."""
    return [(p1 + p2) / 2 for p1, p2 in zip(parent1, parent2)]

def mutate(solution, mutation_rate=0.1):
    """Randomly mutate a solution."""
    return [s + random.uniform(-mutation_rate, mutation_rate) if random.random() < 0.5 else s for s in solution]


```



Hyperparameters:
Population size,
Generations

Pareto optimality

Pareto efficiency or **Pareto optimality** is a situation where no individual or preference criterion can be made better off without making at least one individual or preference criterion worse off.



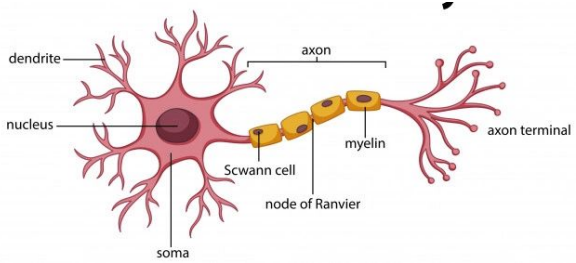
Siruana et al., 2017

Homework

In this homework, you will work with several data sets that we provide via StudOn. Your task is to fit regression lines and a decision boundary. In the exercise we will show you some methods, but you are free to use other methods as well.

Part II:
Deep Learning

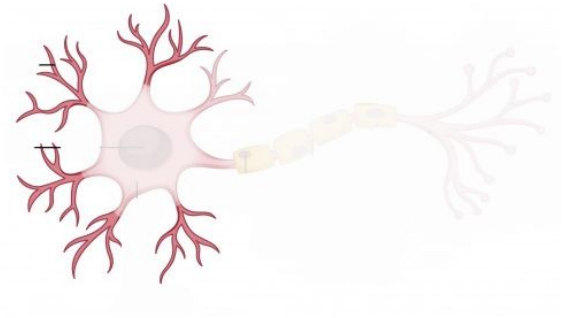
Perceptrons

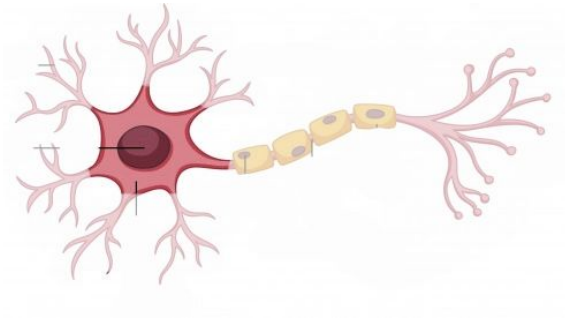


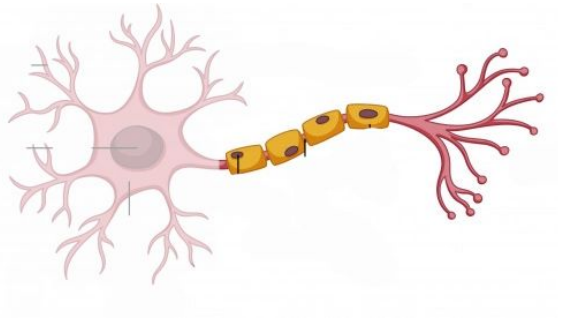
INPUT

COMPUTATION

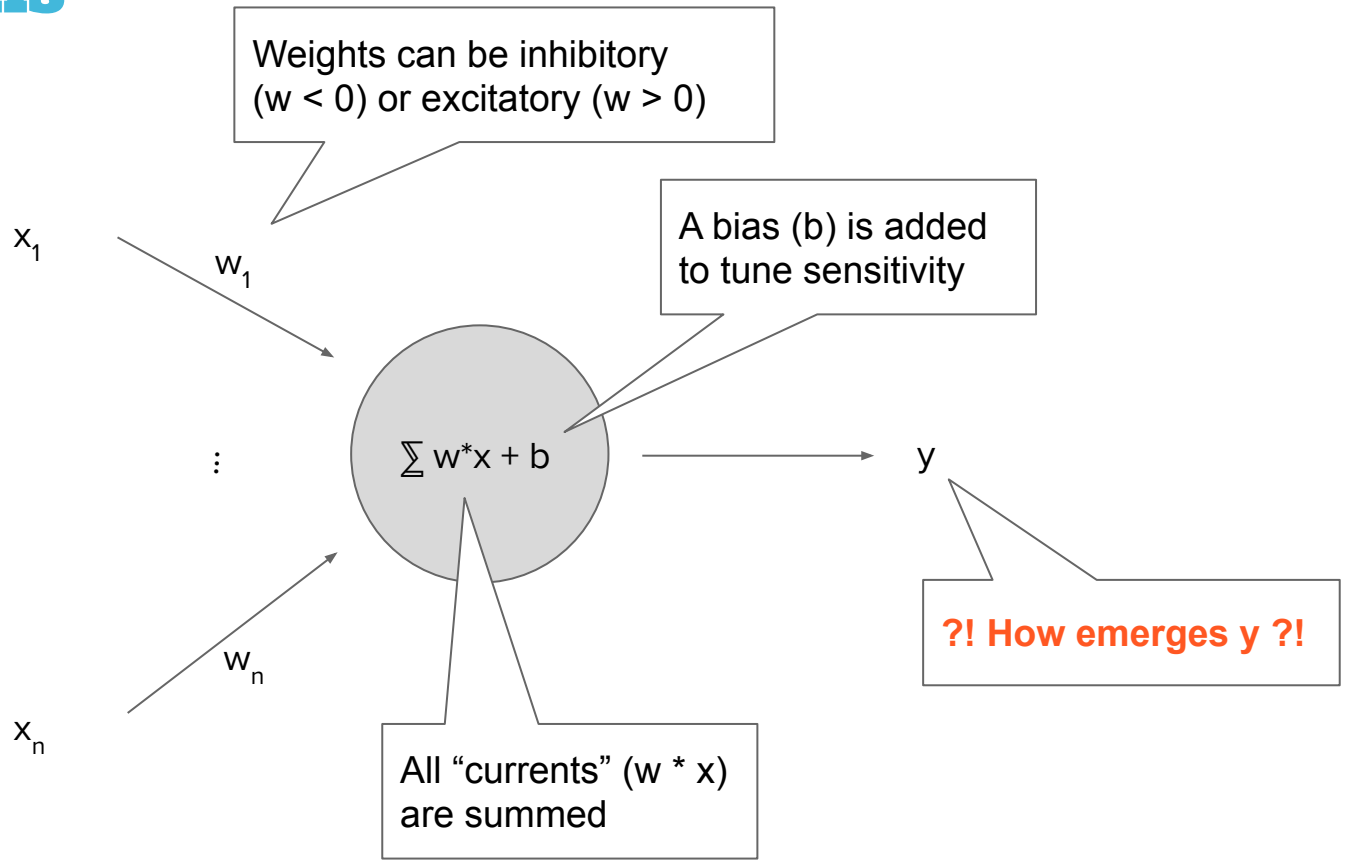
OUTPUT







Perceptrons



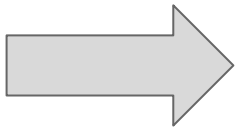
Perceptrons

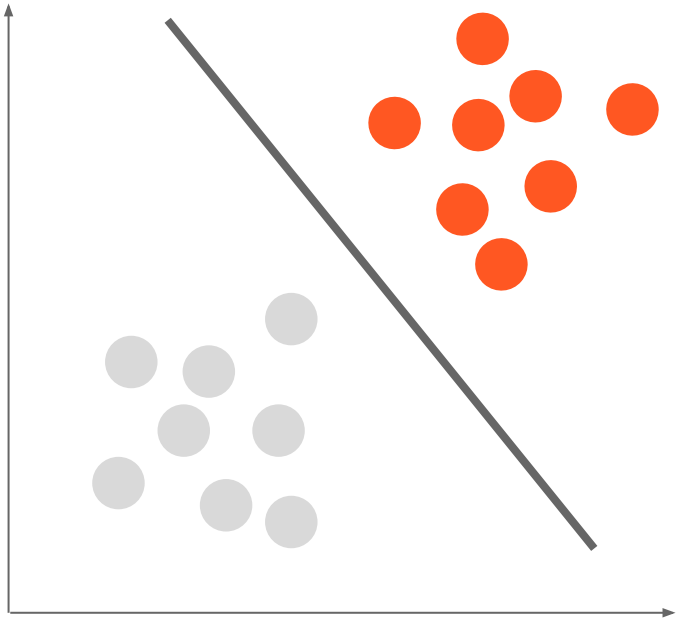
Perceptrons are **LINEAR CLASSIFIER**!

Threshold
step function:

$$\sum_{i=1}^{m} w_i x_i$$

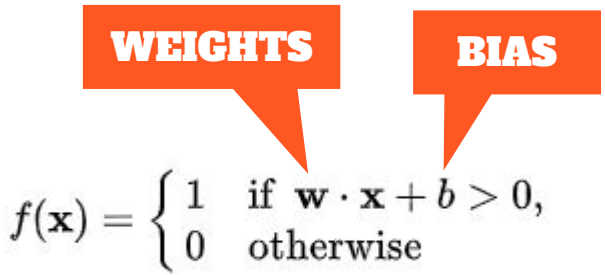
$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0, \\ 0 & \text{otherwise} \end{cases}$$



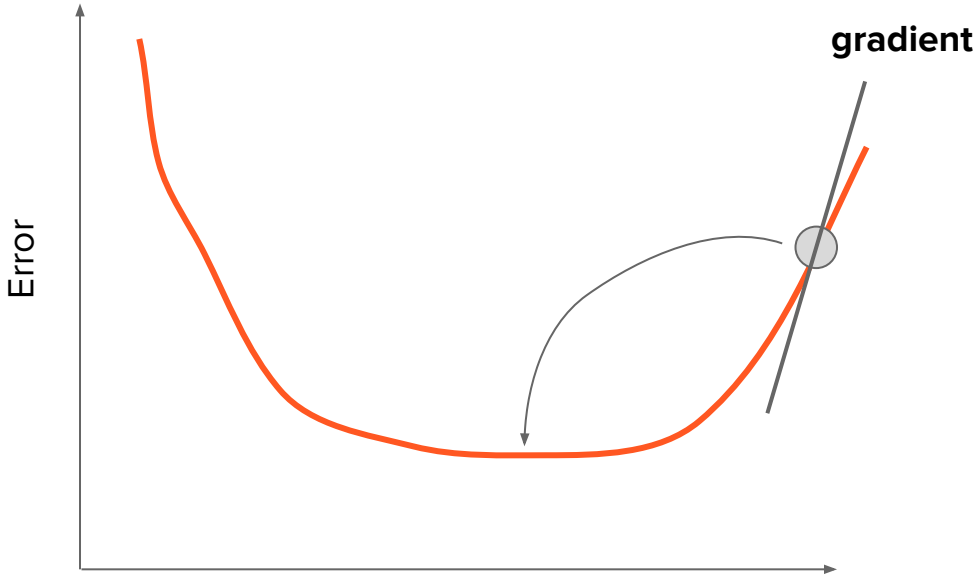


Teaching and Learning

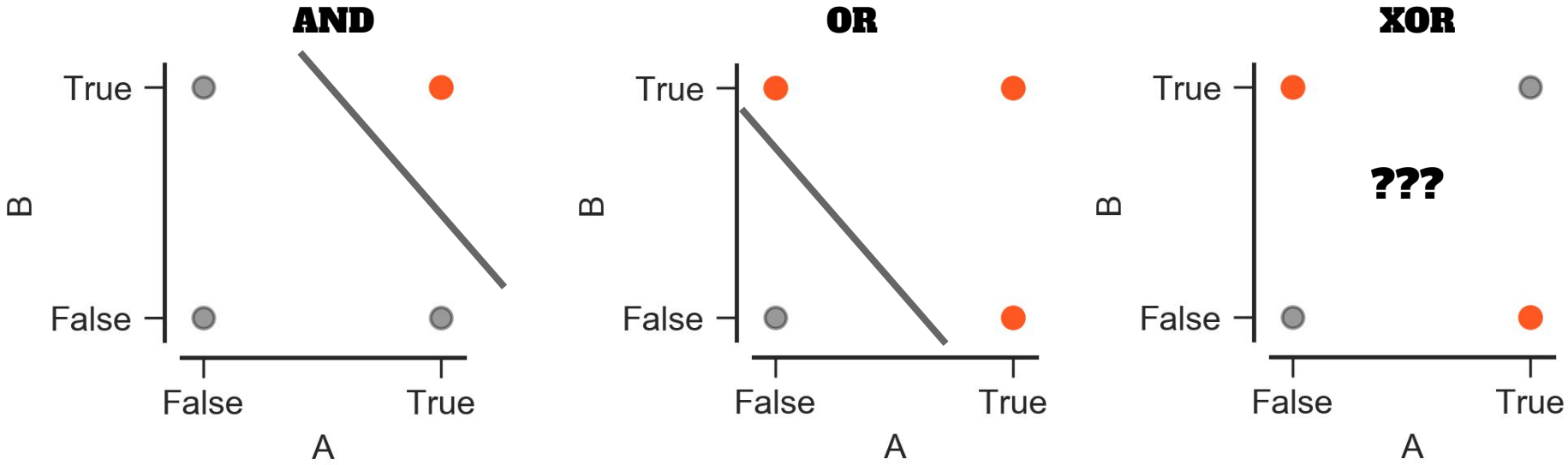
What are the parameters one could tune?



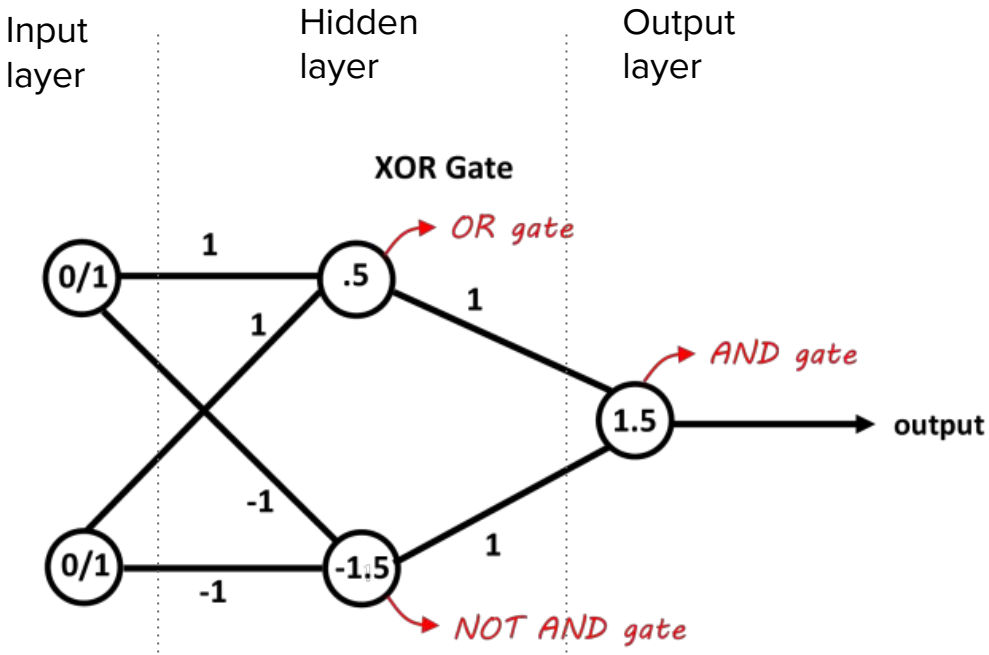
Learning weights with gradient descent

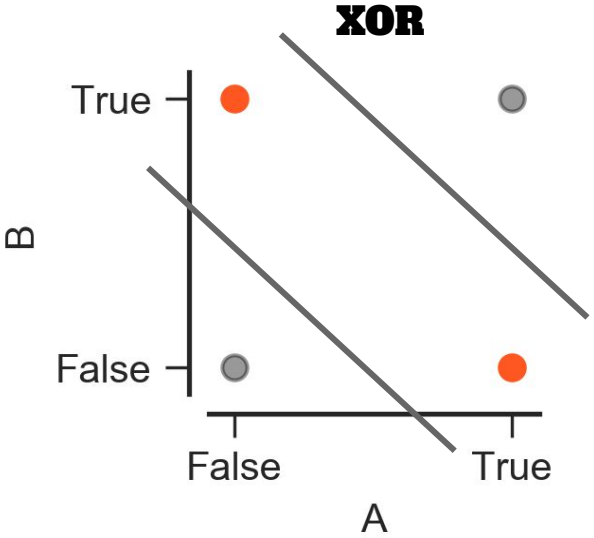


Why perceptrons were abandoned...

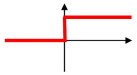
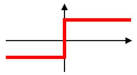
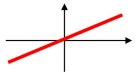
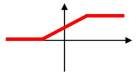
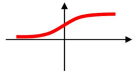
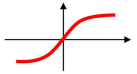
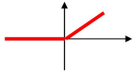
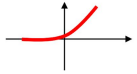


But there is a solution: The multilayer perceptron!



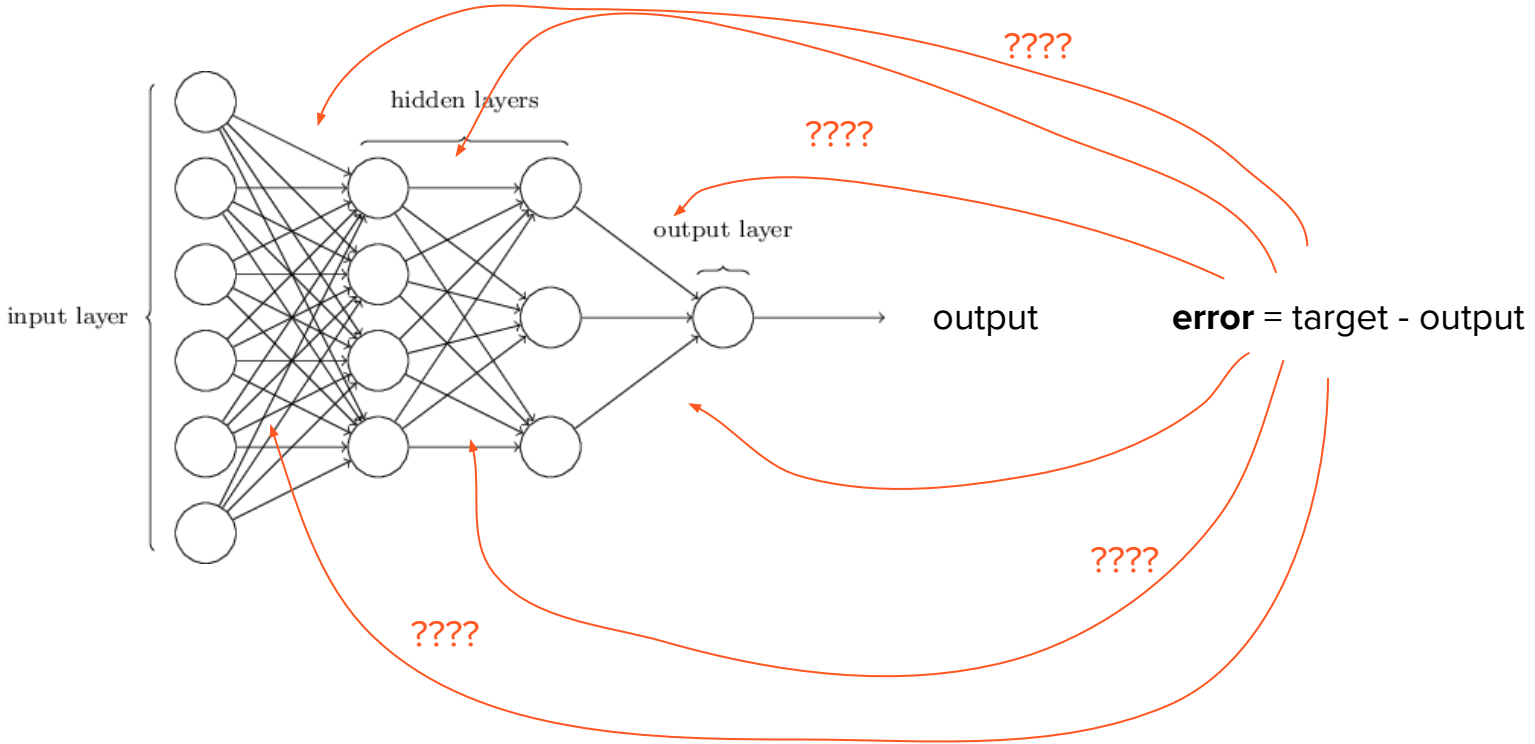


Activation functions

Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$	Multi-layer Neural Networks	
Rectifier, ReLU (Rectified Linear Unit)	$\phi(z) = max(0, z)$	Multi-layer Neural Networks	
Rectifier, softplus	$\phi(z) = \ln(1 + e^{z})$	Multi-layer Neural Networks	

Assigning the error

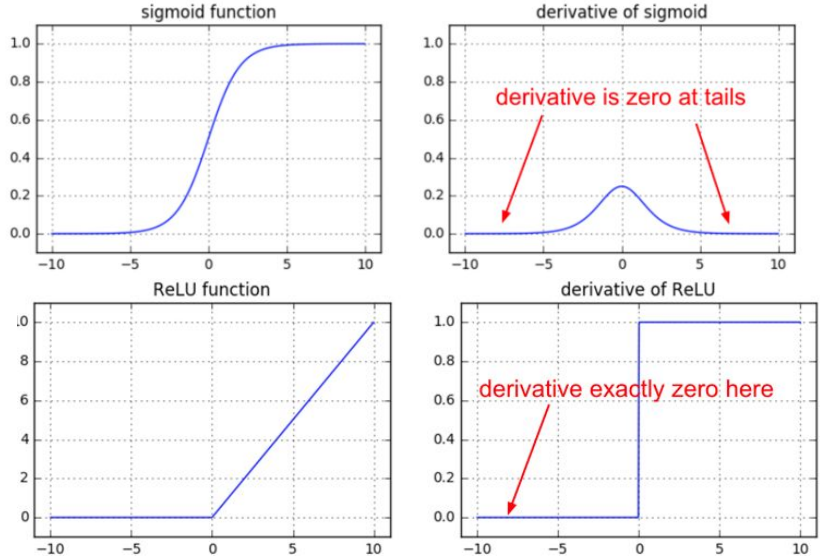
➜ Something called backpropagation.
It is done automatically, but you should KNOW WHAT IT IS DOING!



Yes you should understand backprop

Andrej Karpathy Dec 19, 2016 · 7 min read

17K 46



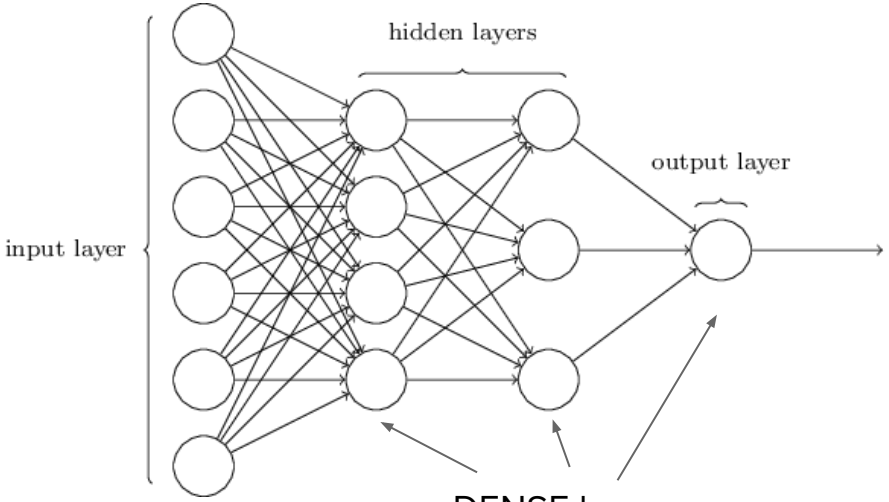
In conclusion

Backpropagation is a leaky abstraction; it is a credit assignment scheme with non-trivial consequences. If you try to ignore how it works under the hood because "TensorFlow automagically makes my networks learn", you will not be ready to wrestle with the dangers it presents, and you will be much less effective at building and debugging neural networks.

https://medium.com/@karpathy/yes-you-should-understand-backprop-e2f06eab496b

TensorFlow/Keras

TensorFlow



e.g. CAT/DOG

DENSE layers

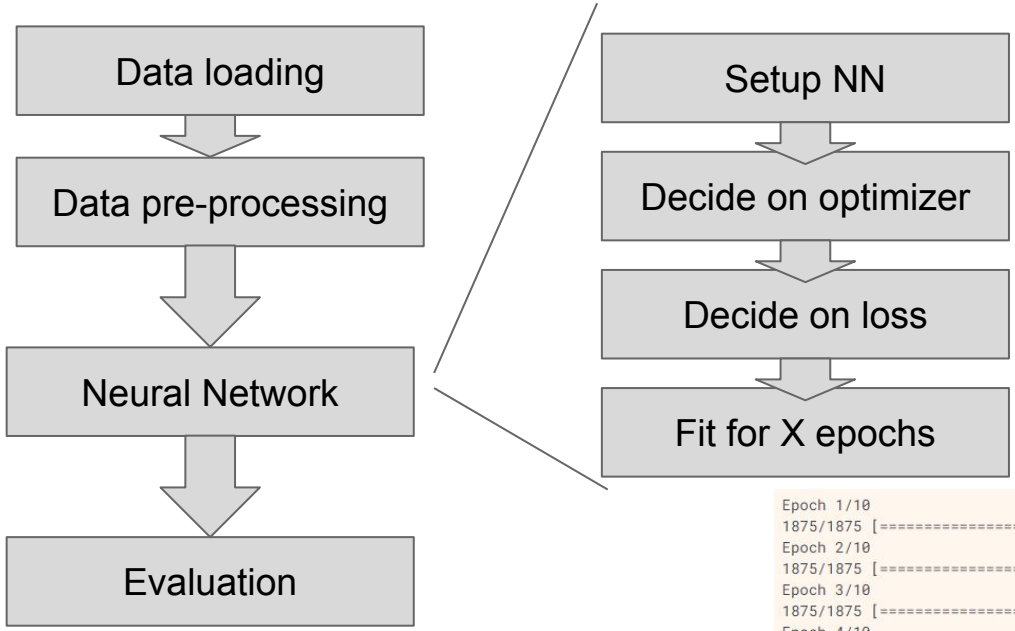
1	1	0
4	2	1
0	2	1

1
1
0
4
2
1
0
2
1

INPUT layer

FLATTEN layer

How to create a NN (in Keras)




```
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10)
])
```

```
model.compile(optimizer='adam',
              loss=SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

```
model.fit(train_images, train_labels, epochs=10)
```

```
Epoch 1/10
1875/1875 [==============================] - 3s 1ms/step - loss: 0.4917 - accuracy: 0.8277
Epoch 2/10
1875/1875 [==============================] - 3s 1ms/step - loss: 0.3702 - accuracy: 0.8674
Epoch 3/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.3328 - accuracy: 0.8793
Epoch 4/10
1875/1875 [==============================] - 3s 1ms/step - loss: 0.3106 - accuracy: 0.8859
```

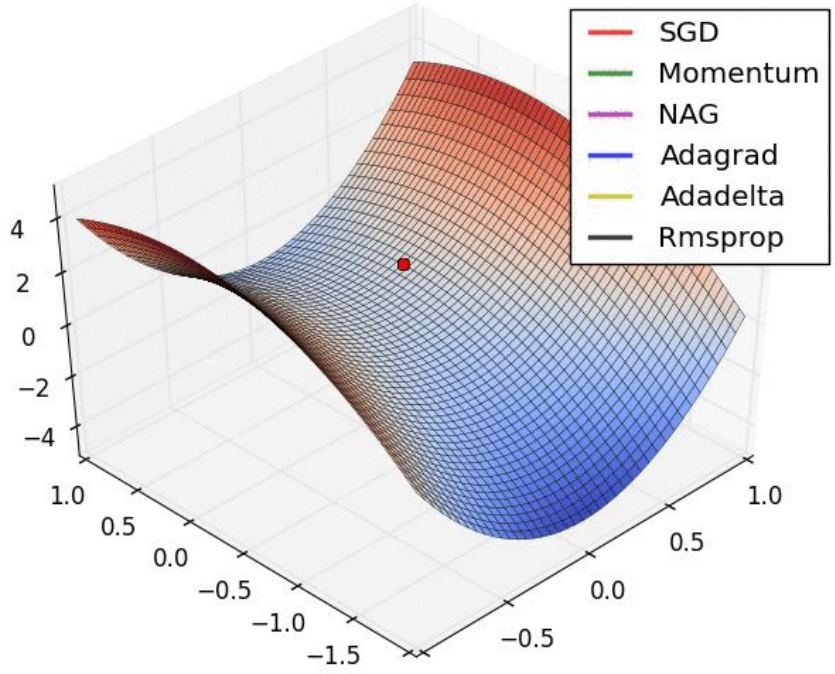
```
313/313 - 1s - loss: 0.3637 - accuracy: 0.8693

Test accuracy: 0.8693000078201294
```

Optimizers

Strategies to optimize the network and perform gradient descent:

- Stochastic gradient descent (SGD)
- SGD + momentum
- RMSprop
- Adadelta
- Adam
- ...

Try SGD(momentum=.9) and Adam



The loss function

Important:
your objective function should represent what you would like to optimize!!

Regression Losses

Mean Square Error/Quadratic Loss/L2 Loss

Mathematical formulation :-

$$MSE = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}$$

Mean Squared Error

Cross Entropy Loss/Negative Log Likelihood

This is the most common setting for classification problems. Cross-entropy loss increases as the predicted probability diverges from the actual label.

Mathematical formulation :-

$$CrossEntropyLoss = -(y_i log(\hat{y}_i) + (1 - y_i) log(1 - \hat{y}_i))$$

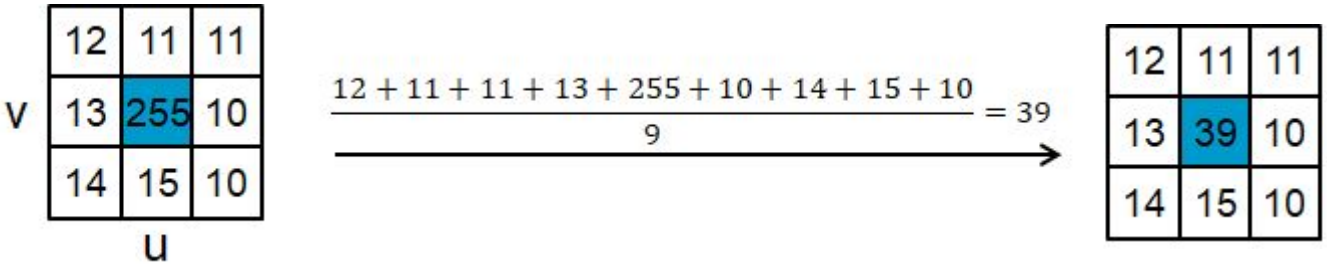
Cross entropy loss

Code for an MLP

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation='softmax')
]) # Multilayer perceptron
```

Changing gears...

Imagine a noisy pixel

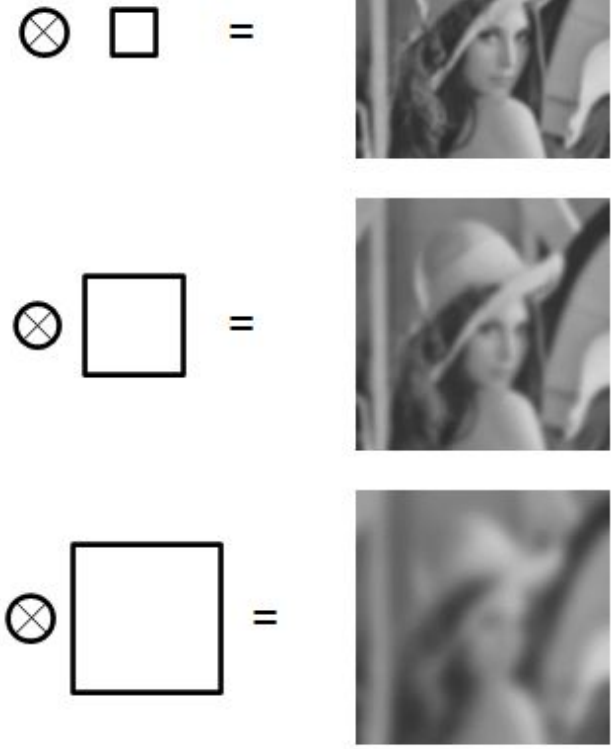


$$I'(u,v) = \frac{\sum_{i=-1}^{1}\sum_{j=-1}^{1} I(u+i,v+j)}{9}$$

1	1	1
1	1	1
1	1	1

Applying a linear filter across a whole image

Kernel size effect



Edge detection linear filters

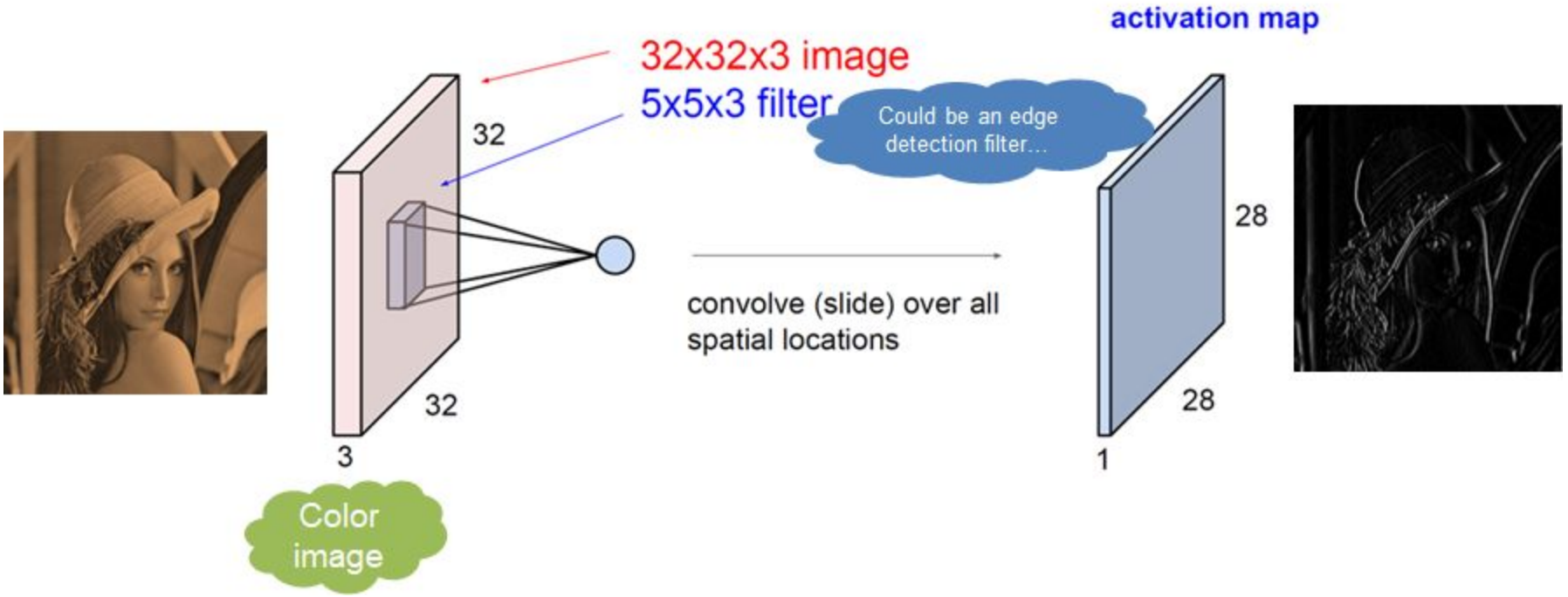
Prewitt-Filter

$$H_x^P = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \quad and \quad H_y^P = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

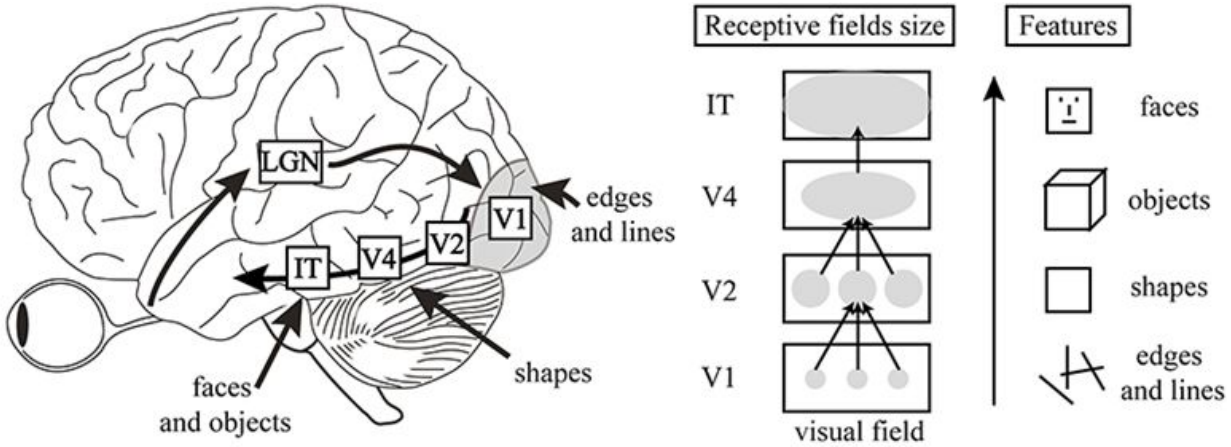
Sobel-Filter

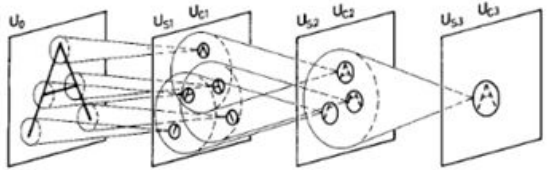
$$H_x^S = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad and \quad H_y^S = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Basis of convolutional neural networks



How does our visual cortex work?





https://neurdiness.wordpress.com/2018/05/17/deep-convolutional-neural-networks-as-models-of-the-visual-system-qa/

Convolutional NNs (1998)

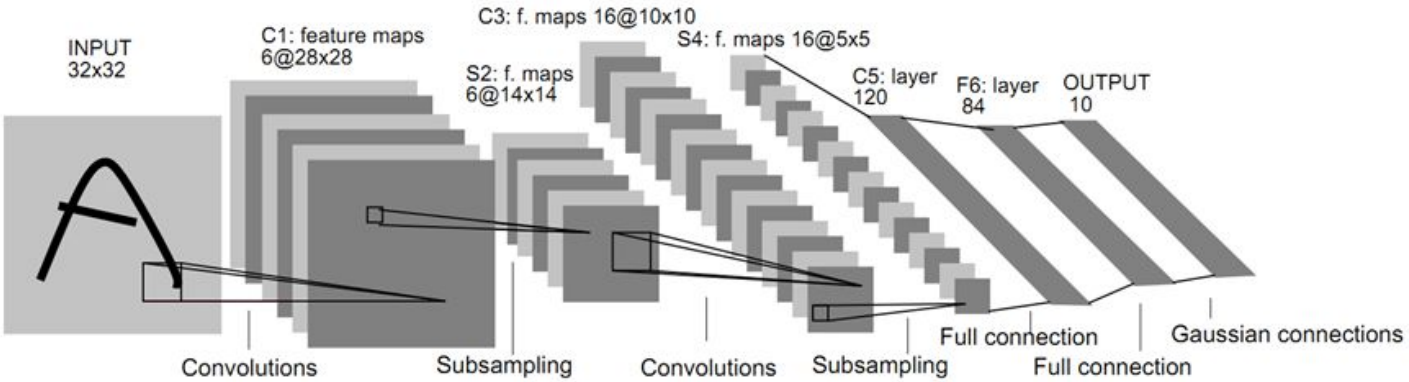


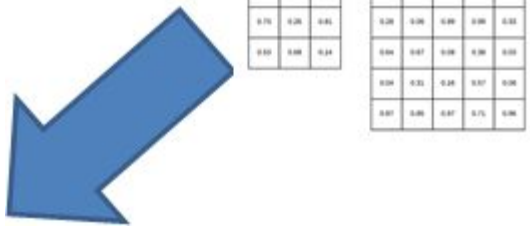
Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

3 elements:

- Convolutions
- Subsamplings
- Fully Connected/Dense/Perceptron layers

Filters/
Activation maps

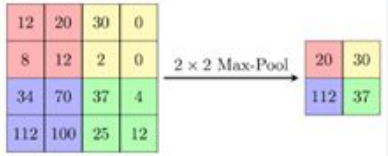
$H_x^p = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$ and $H_y^p = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$



Missing the last

In [46]:

```
cnn = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(8, (3,3), input_segment="(28,28,1)", padding='same', activation='relu'),
    tf.keras.layers.MaxPool2D(),
    tf.keras.layers.Conv2D(16, (3,3)
    tf.keras.layers.MaxPool2D(),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activ
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10, activation='softmax')
])
```



Same as in MLPs...

The last slide

- Gradient descent is a common principle for adjusting weights in a model
- Artificial neural networks are loosely related to real, biological networks
- Multilayer perceptrons (MLPs) are the classic ANNs consisting of simple neural units (perceptrons)
- Convolutional neural networks are loosely related to our visual cortex and are superior in image processing (in many tasks)