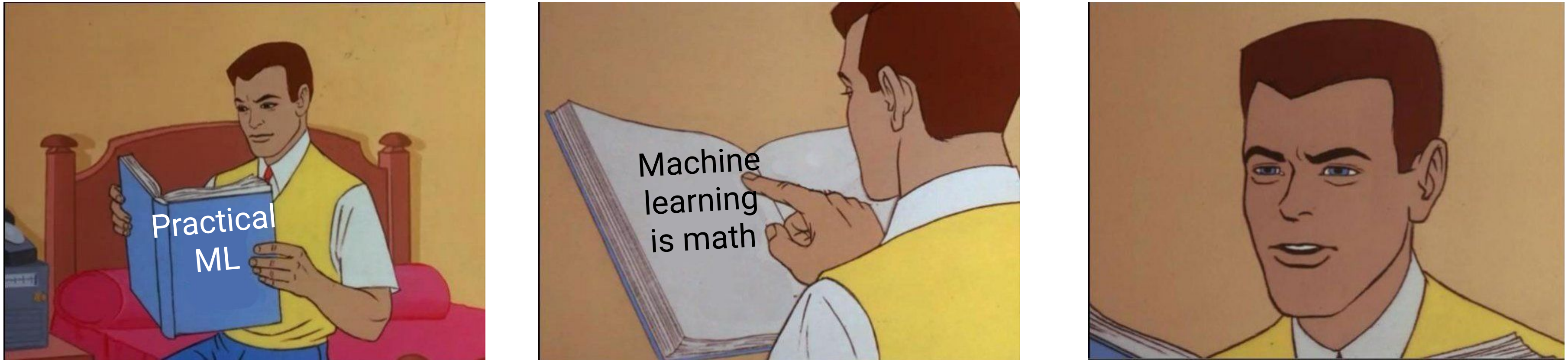
Practical ML

Antonio Pitasi
Samuele Sabella

2019 - Polo Fibonacci

Before we get started, we need some theory



Machine learning

- **Practice**: We define machine learning as a set of methods that can automatically detect patterns in data, and then use the uncovered patterns to predict future data, or to perform other kinds of decision making under uncertainty

  - *Machine Learning: A Probabilistic Perspective, Kevin P. Murphy*

- **Theory:** How does learning performance vary with the number of training examples presented? Which learning algorithms are most appropriate for various types of learning tasks?

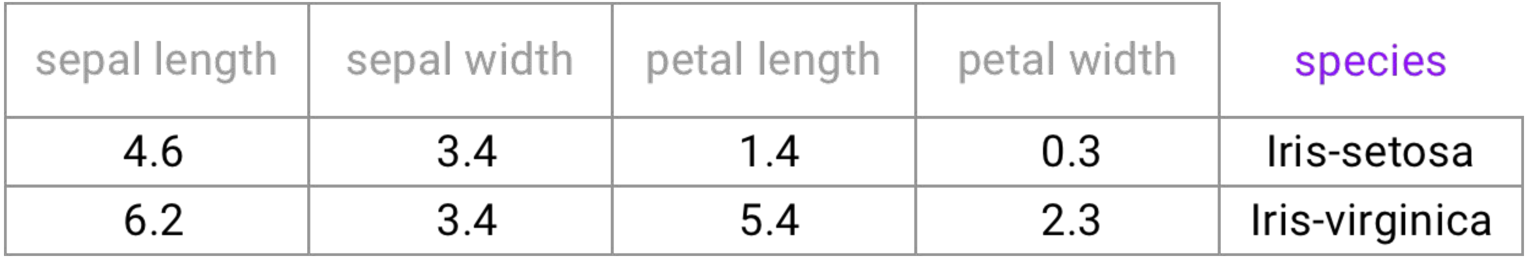
  - *Machine Learning, Tom Mitchell*

ML is not only artificial neural networks

- Lots of mathematical models
  - Hidden Markov models
  - Support Vector Machines
  - Decision trees
  - Boltzmann machines, Deep belief network, Deep Boltzmann
- Neural network models are many...
  - Shallow network, Deep neural network
  - CNN (Yolo, AlexNet, GoogLeNet)
  - Echo state network, Deep echo state network
  - Rnn, LSTM, GRU

Machine learning categories

- **Supervised:** the goal is to learn a mapping from input to output given a dataset of labeled pairs $\mathcal{D} = \{x_i, y_i\}_{i=1}^{N}$ called training set (e.g. Iris Data Set [2])



sepal length	sepal width	petal length	petal width	species
4.6	3.4	1.4	0.3	Iris-setosa
6.2	3.4	5.4	2.3	Iris-virginica

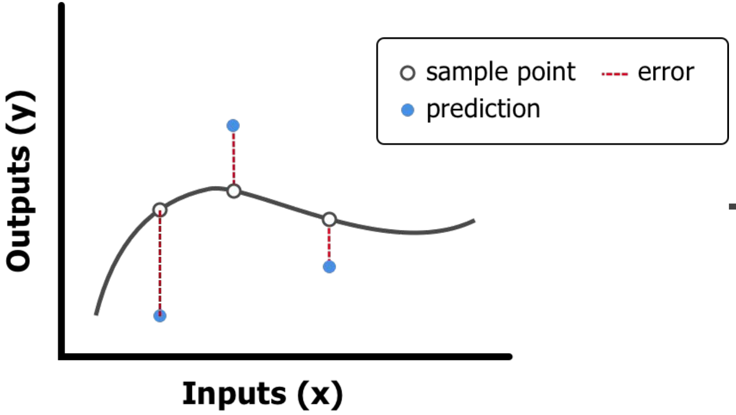
- **Unsupervised**: we have only a set of data points $\mathcal{D} = \{x_i\}_{i=1}^{N}$ and the goal is to find interesting patterns in the data
  Example: young American males who buy diapers also have a predisposition to buy beer (original story [3])

How does it work?

- Dataset of examples to learn from
- A model to learn from that data (e.g. neural net)
  - With some parameters to tune
  - With some hyperparameter to choose (neurons, layers, ...)
- Target function (loss) to minimize

cat: 0.9

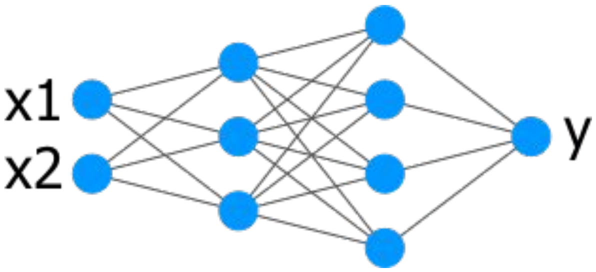
lobster: 0.1



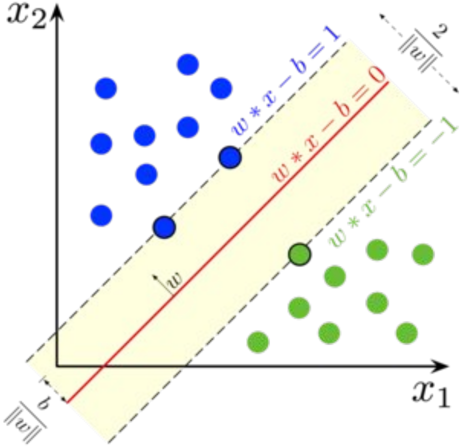
You are
[0.3, 0.2, 0.1]
wrong!

What is usually done

- Validation phase: compare different models and configurations
  - Which model to choose



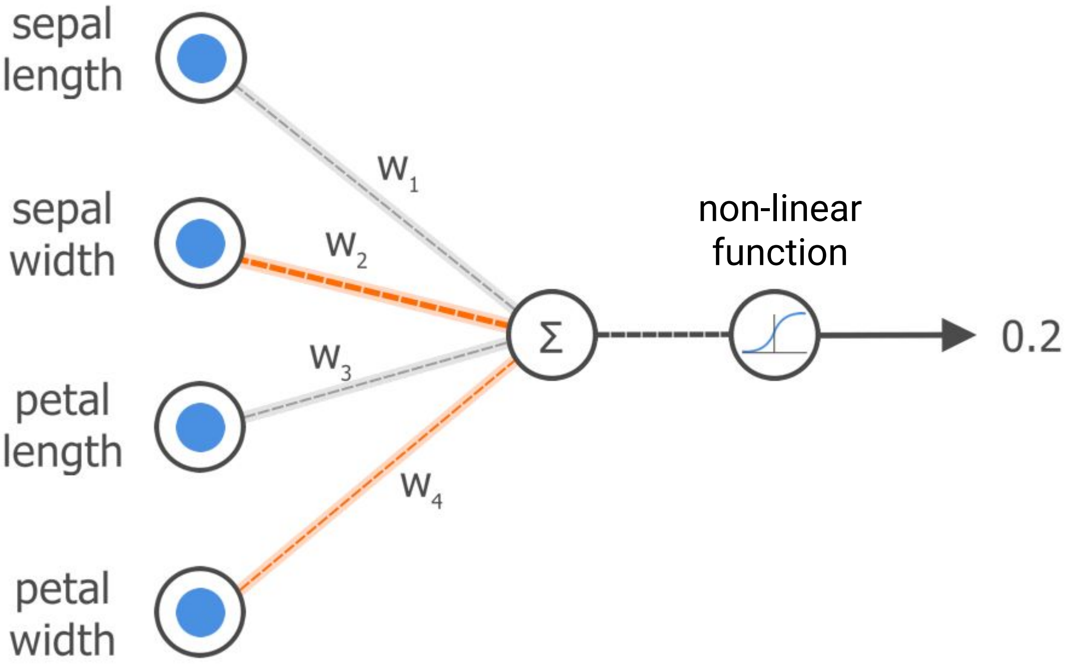
OR?

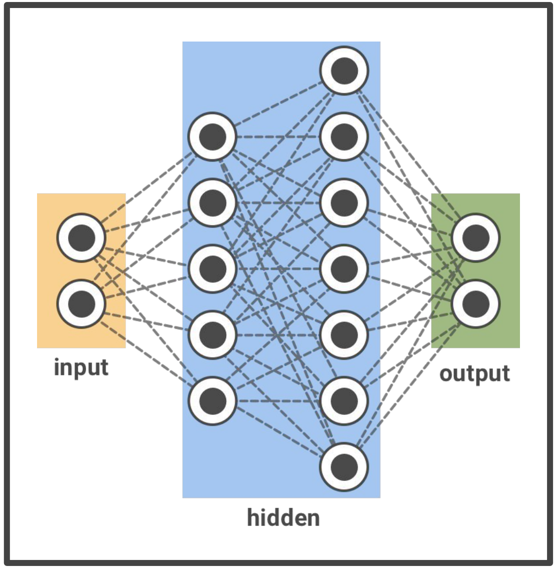


  - Model hyper-parameters
- Test phase: loss, accuracy, recall, precision…
- We skip all this for seek of simplicity

Note: train/validation/test on different data

Models: feed-forward neural networks



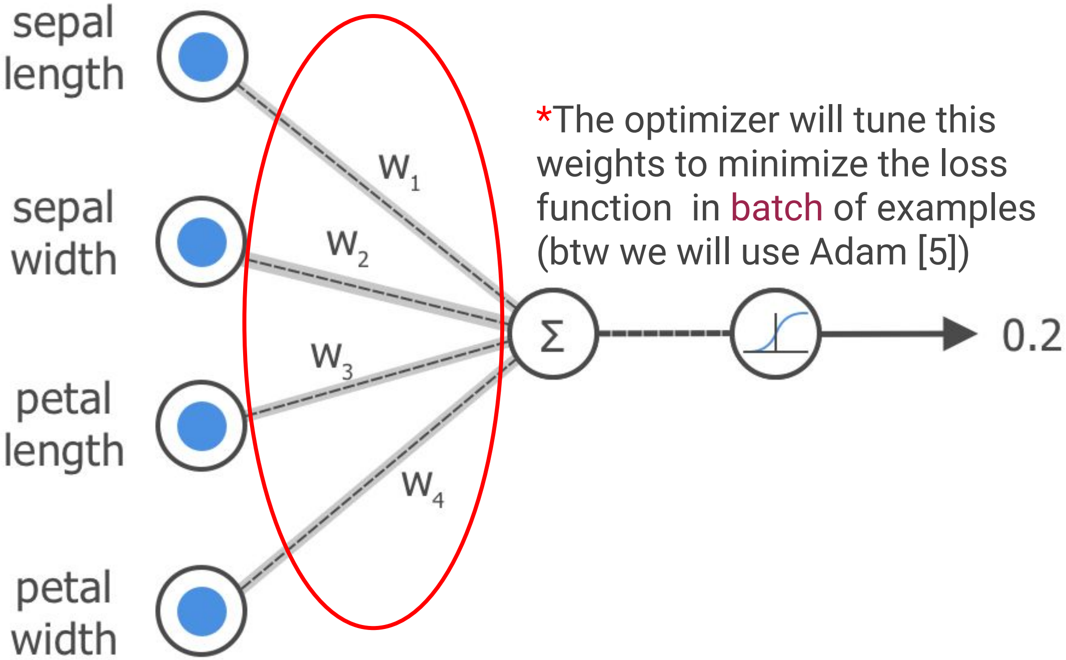


Stack neurons in layers

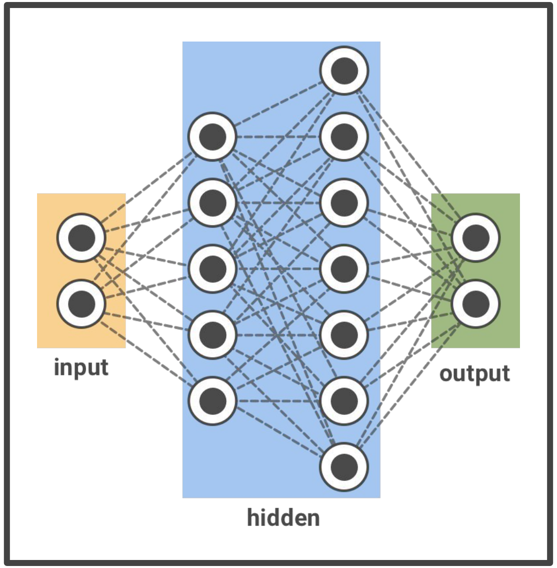
virginica if argmax(net(input))==0 else setosa

Note: A non-linear function ensure ANN to be a universal approximator [4]

Models: feed-forward neural networks



*The optimizer will tune this weights to minimize the loss function in batch of examples (btw we will use Adam [5])



Stack neurons in layers

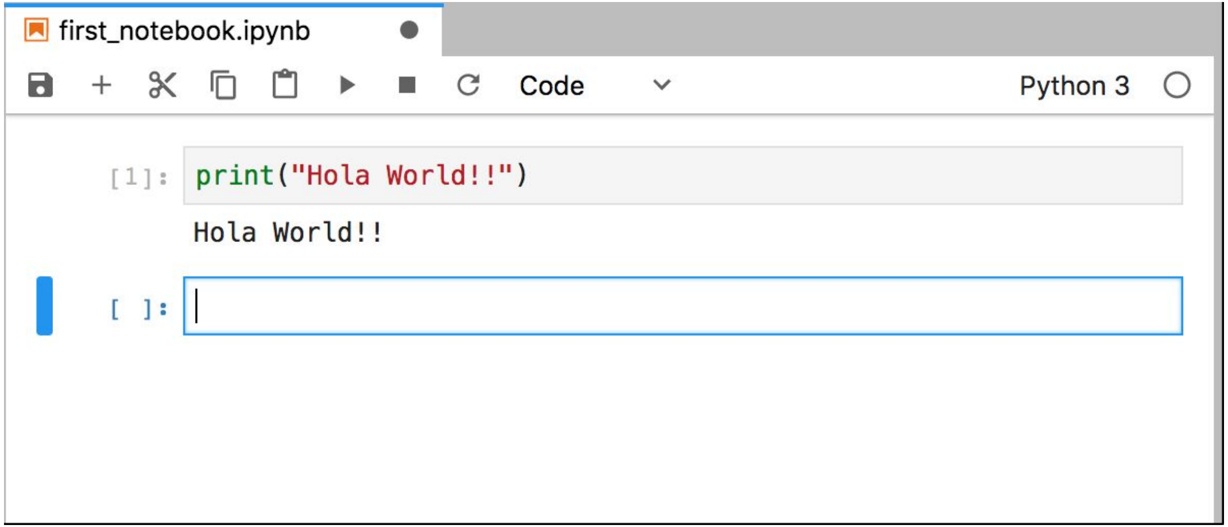
virginica if argmax(net(input))==0 else setosa

A lot more stuff to know but for us...

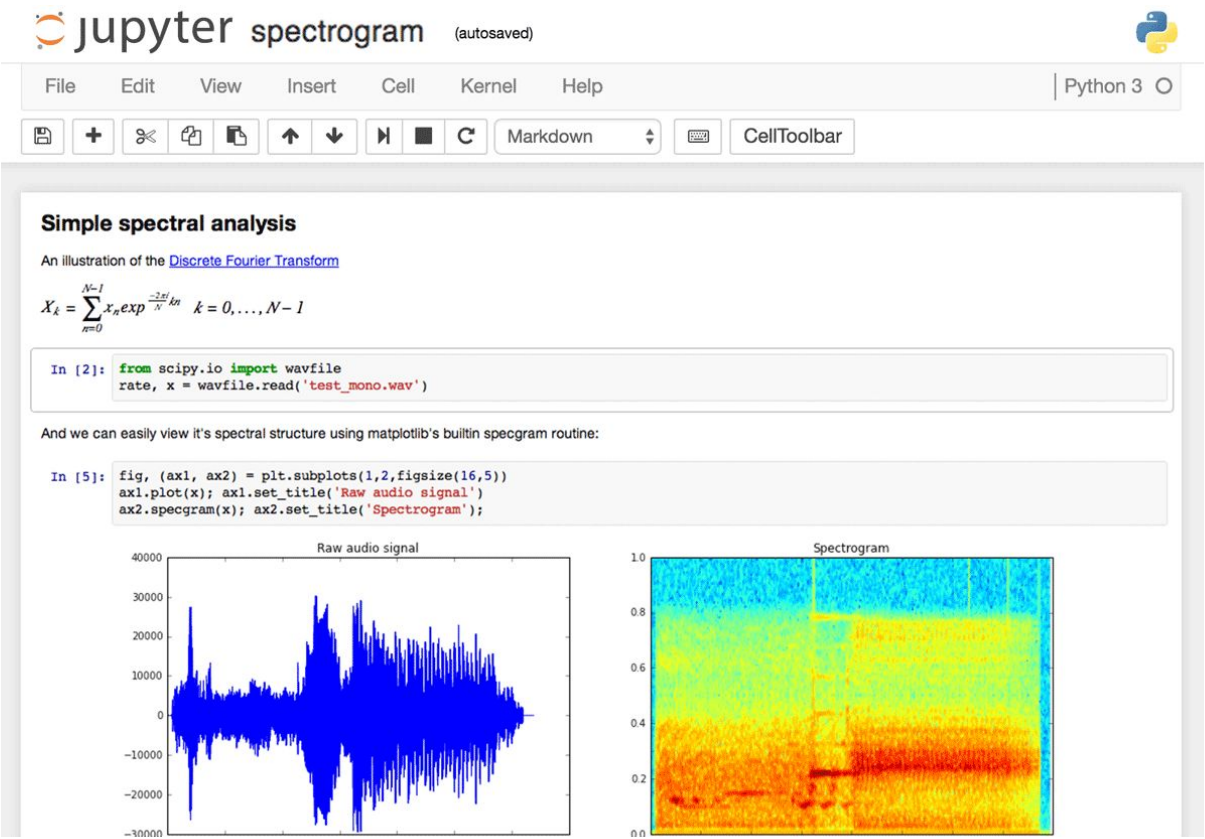
UNDERSTANDING MACHINE LEARNING

import keras

Practical ML



```
[1]: print("Hola World!!")
Hola World!!
[ ]:
```



Simple spectral analysis

An illustration of the Discrete Fourier Transform

$$X_k = \sum_{n=0}^{N-1} x_n exp^{\frac{-2\pi i}{N}kn} \quad k = 0,\dots,N-1$$

```
In [2]: from scipy.io import wavfile
        rate, x = wavfile.read('test_mono.wav')
```

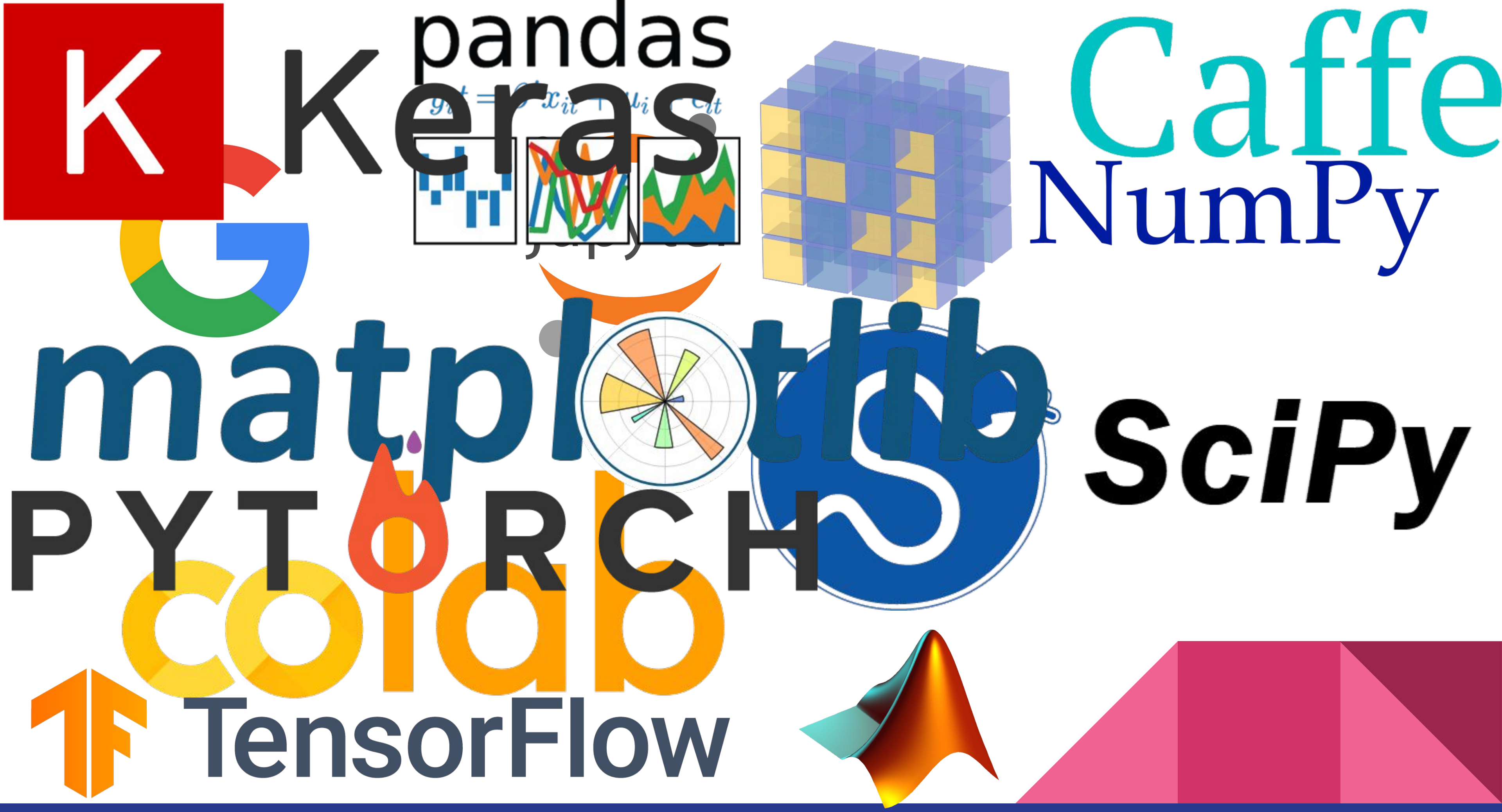
And we can easily view it's spectral structure using matplotlib's builtin specgram routine:

```
In [5]: fig, (ax1, ax2) = plt.subplots(1,2,figsize(16,5))
        ax1.plot(x); ax1.set_title('Raw audio signal')
        ax2.specgram(x); ax2.set_title('Spectrogram');
```

jupyter

- Interactive
- Collaborative
- Python, R, Julia, Scala, ...

+
jupyter
=
colab


K
Keras
pandas
Caffe
NumPy
jupyter
matplotlib
SciPy
PYTORCH
colab
TensorFlow

Keras

Features:

Easy to build a neural network

Easy to build a neural network wrong

Keras

Keep an eye for:

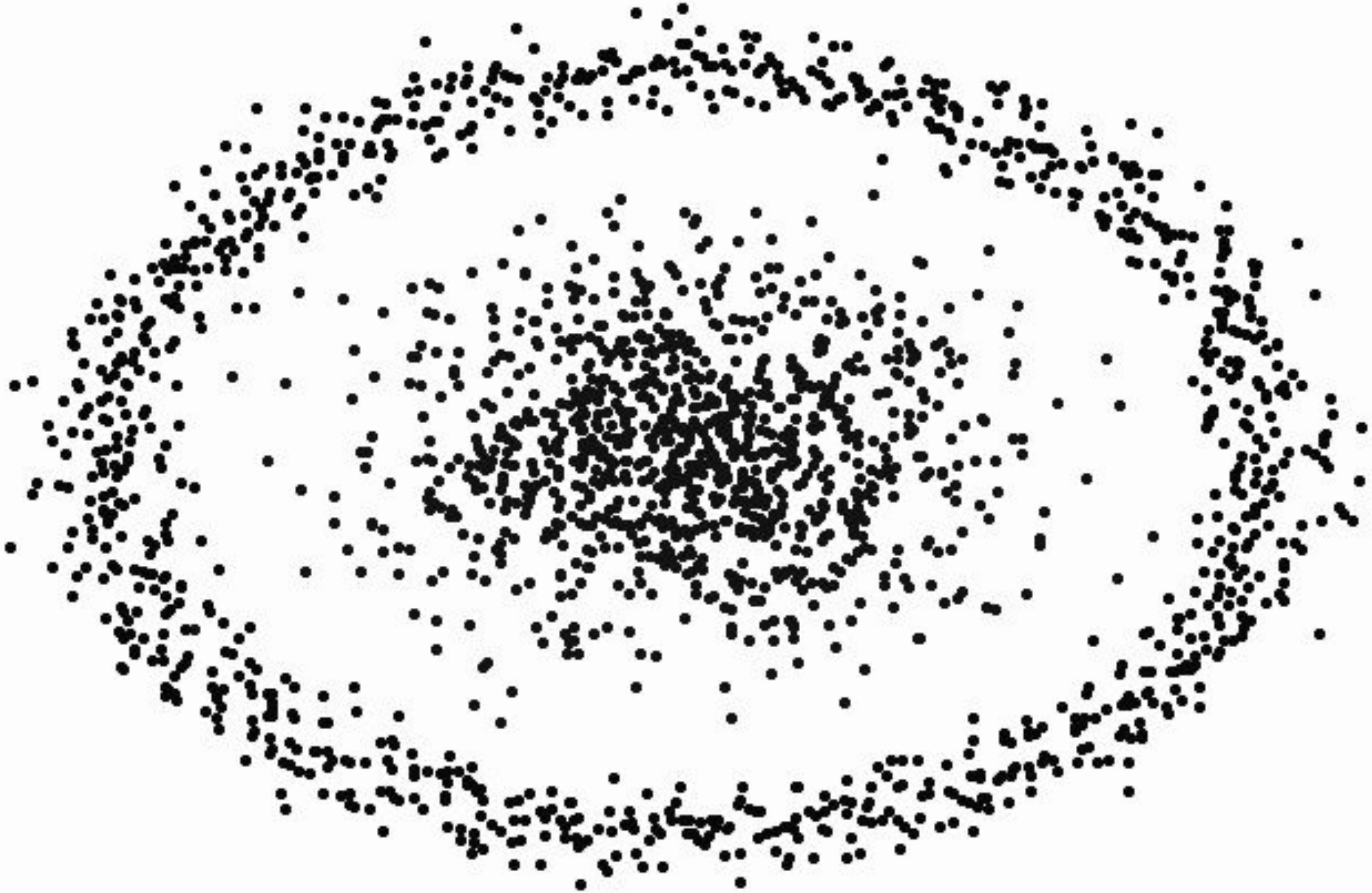
Accuracy

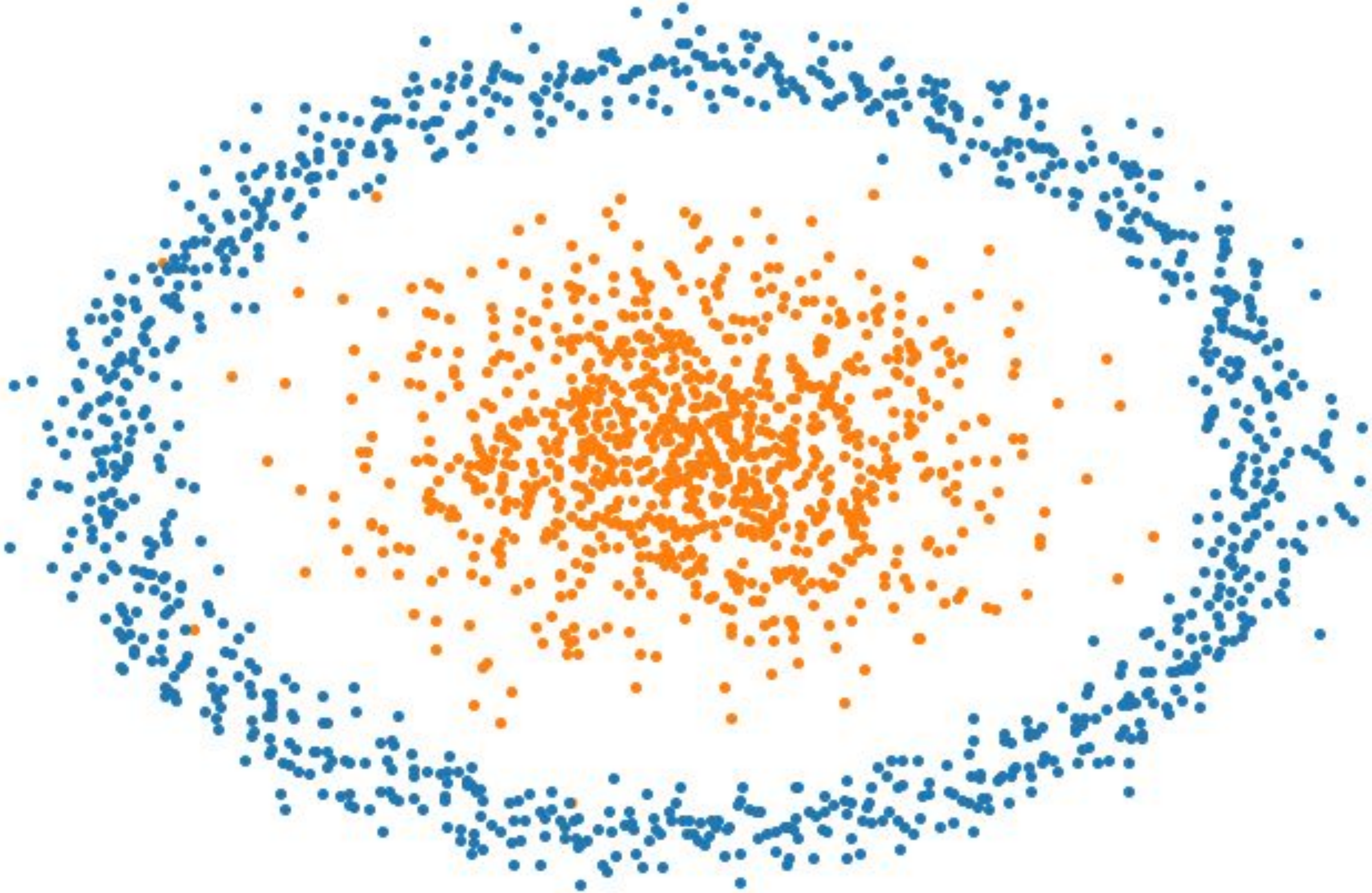
Fitting

Performance

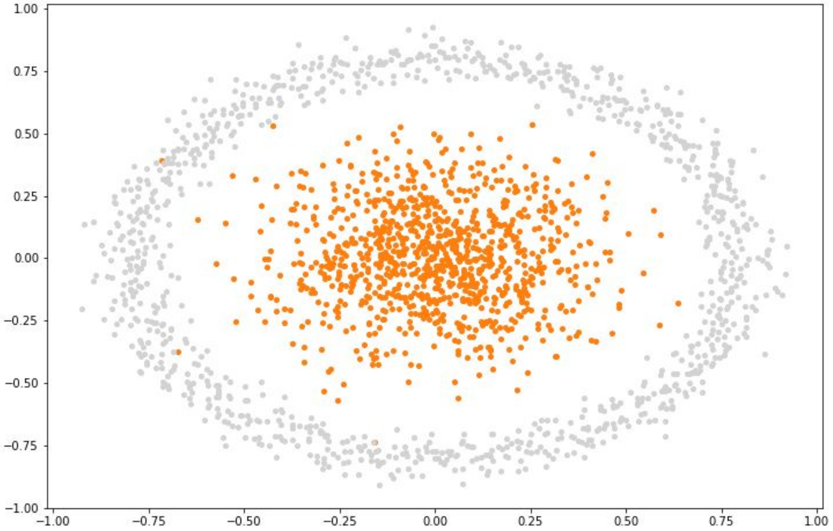
Problem 1

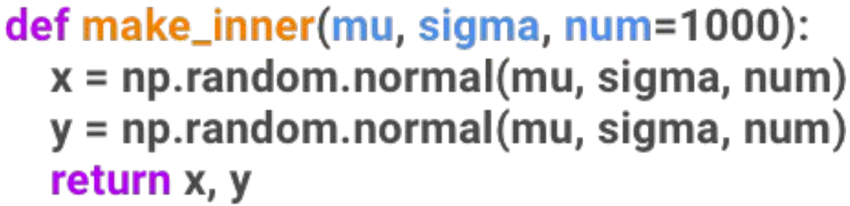
Points classification





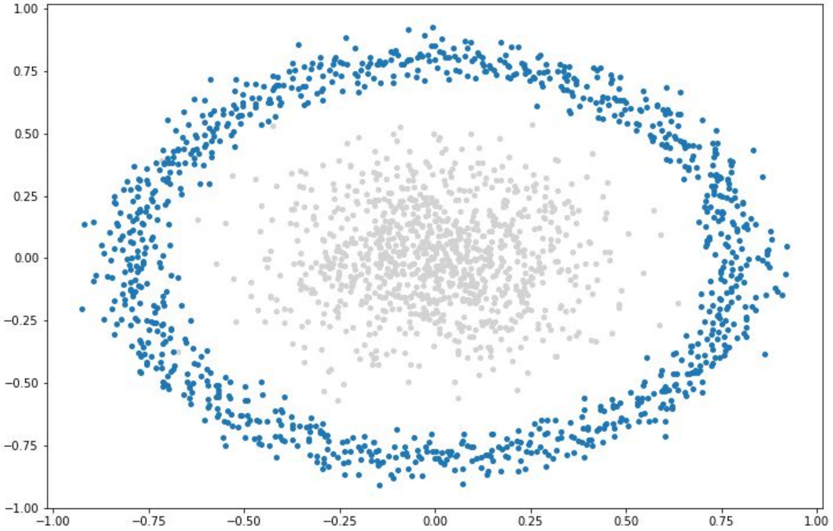
Generating the dataset

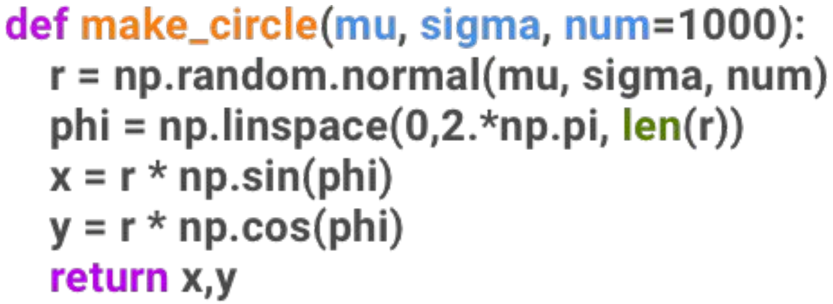




```
def make_inner(mu, sigma, num=1000):
   x = np.random.normal(mu, sigma, num)
   y = np.random.normal(mu, sigma, num)
   return x, y
```

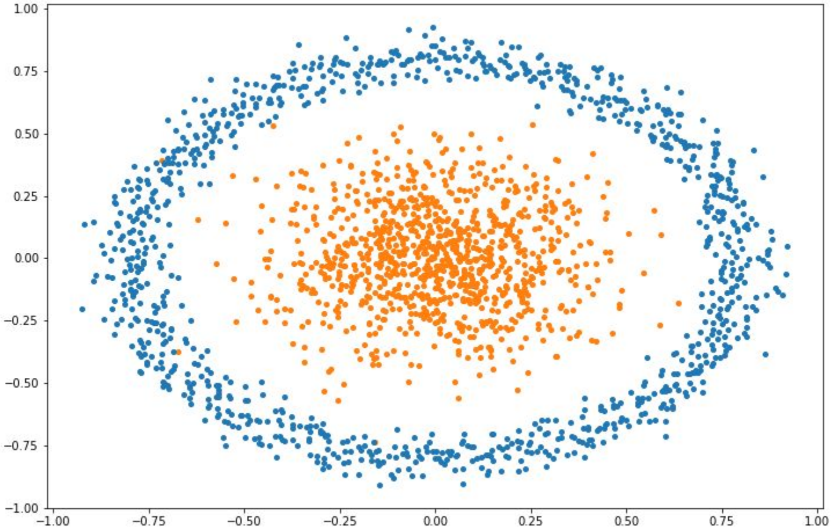
Generating the dataset

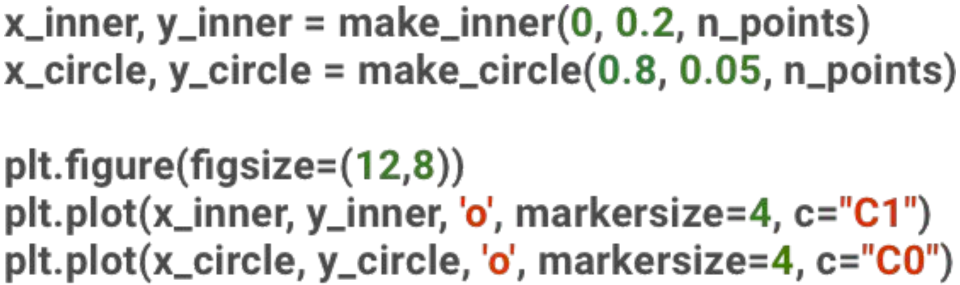




```
def make_circle(mu, sigma, num=1000):
    r = np.random.normal(mu, sigma, num)
    phi = np.linspace(0,2.*np.pi, len(r))
    x = r * np.sin(phi)
    y = r * np.cos(phi)
    return x,y
```

Plotting





```
x_inner, y_inner = make_inner(0, 0.2, n_points)
x_circle, y_circle = make_circle(0.8, 0.05, n_points)

plt.figure(figsize=(12,8))
plt.plot(x_inner, y_inner, 'o', markersize=4, c="C1")
plt.plot(x_circle, y_circle, 'o', markersize=4, c="C0")
```

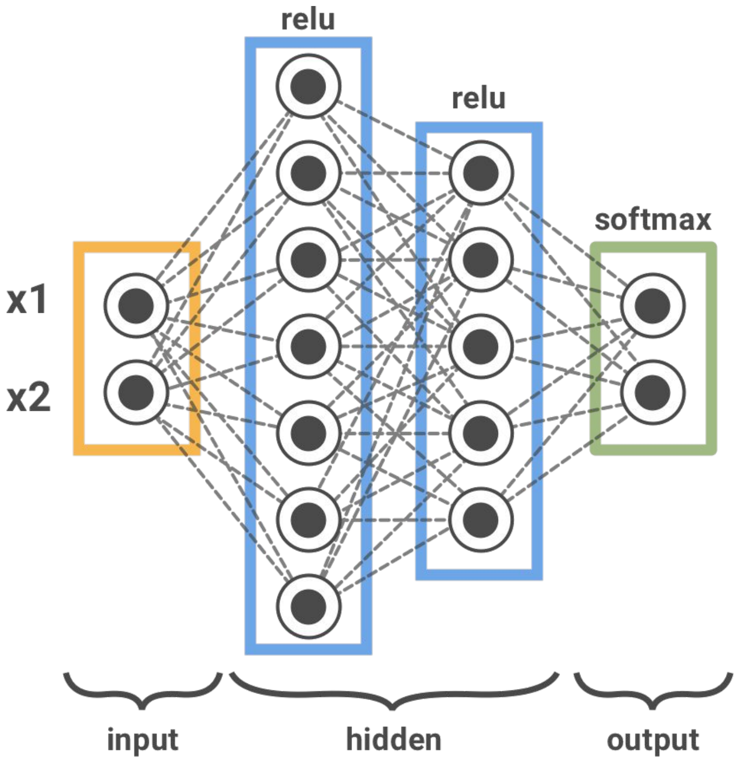
Our model

- For non-linearity: rectifier linear unit

$$relu(x) = max(0, x)$$

- We use a softmax function in the output layer to represent a probability distribution

$$softmax(x)_i = \frac{exp(x_i)}{\sum_j exp(x_j))}$$

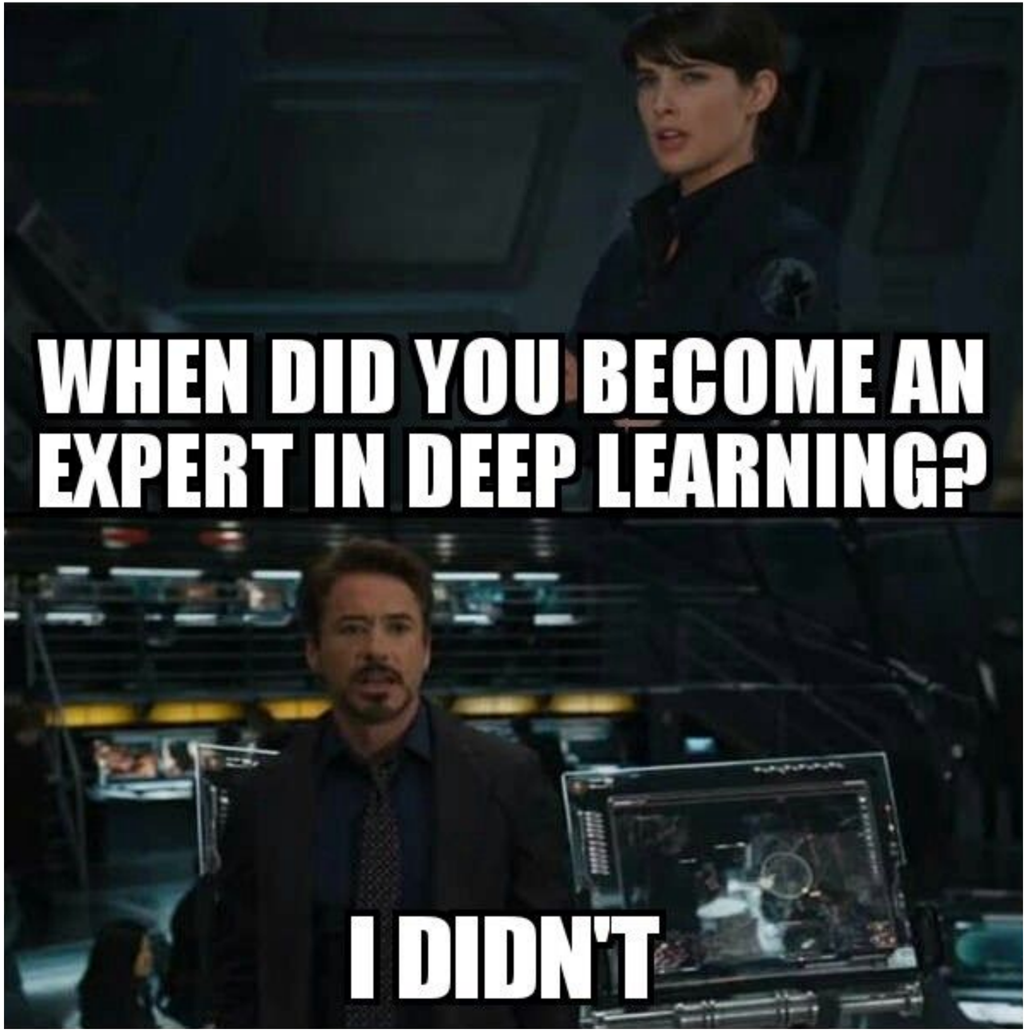


Let’s code!

https://colab.research.google.com

https://ml.anto.pt

Me after training a neural network

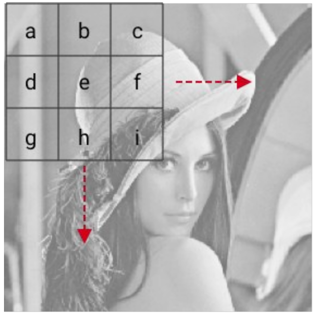


Back to theory - Convolving Lenna

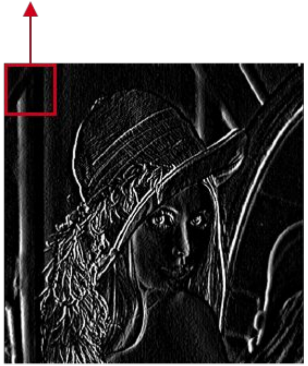
- Given a function f, a convolution g with a kernel w is given by a very complex formula with a very simple meaning: *“adding each element of the image to its local neighbors, weighted by the kernel” (wikipedia)*

$$G_x = \begin{pmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{pmatrix}$$

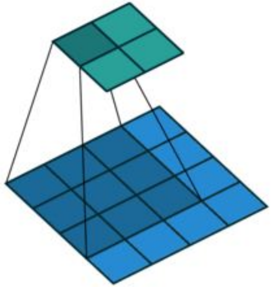
$$\begin{aligned}&(-1*a+0*b+1*c)+\\&+(-2*d+0*e+2*f)+\\&+(-1*g+0*h+1*i)\end{aligned}$$



INPUT (I)



$G_x * I$

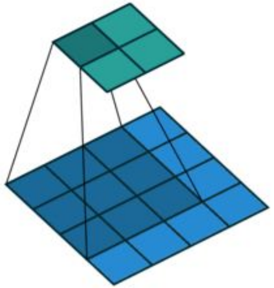


$$\boxed{w * f(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x-s, y-t)}$$

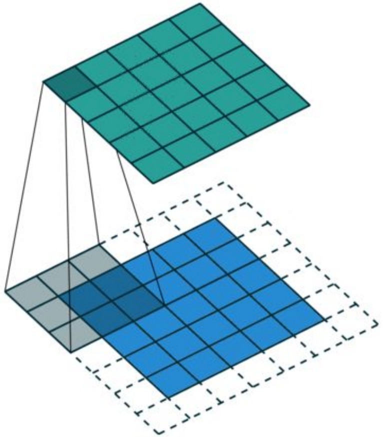
Note: check the manual implementation at link http://bit.ly/ml-dummies-lenna References: [11]

Convolution arithmetic

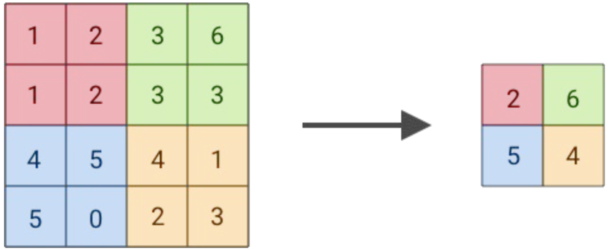
- Zero-padding: deal with borders pixels by adding zeros (preserves the size)
- Pooling: helps the network to become transformation invariant (translations, rotations...) [7]



No padding, no strides



padding=same && no strides



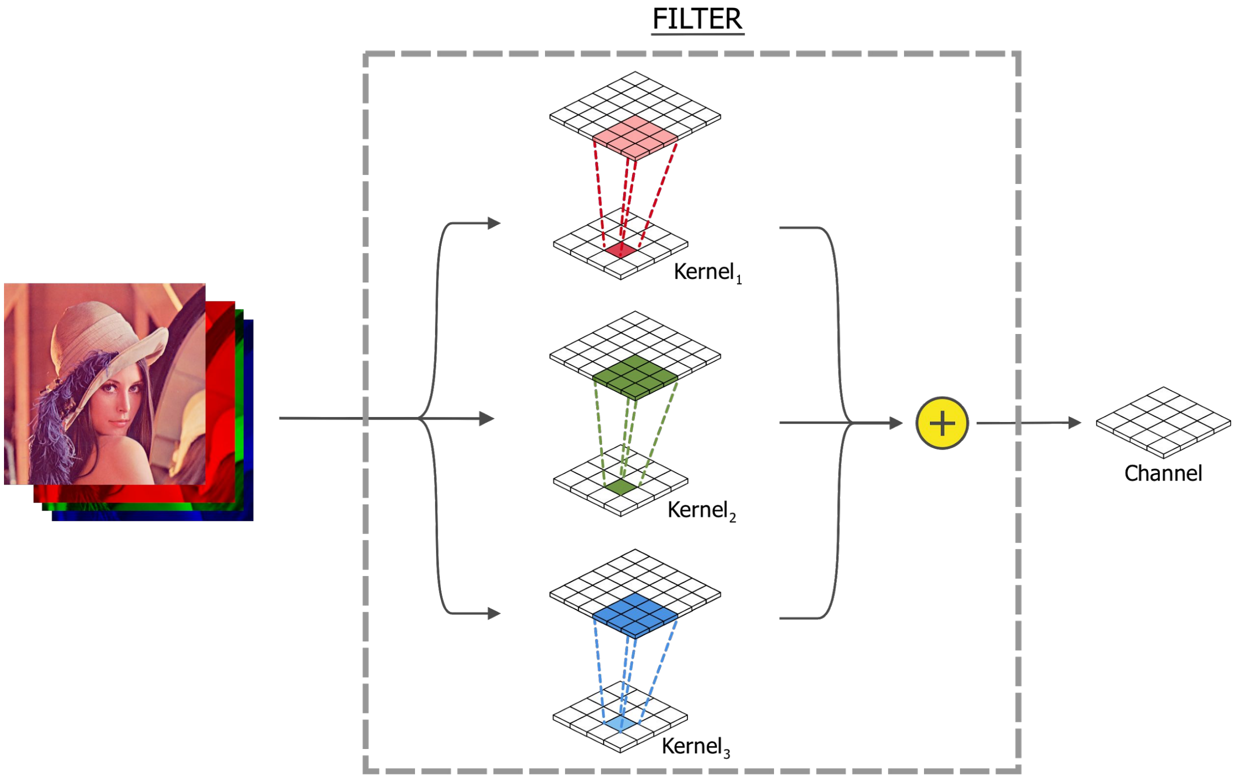
1	2	3	6
1	2	3	3
4	5	4	1
5	0	2	3

2	6
5	4

max pooling && 2x2 strides

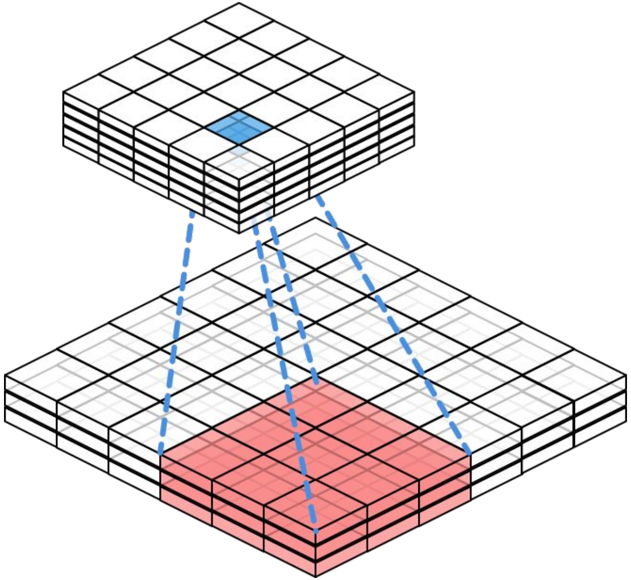
GIF credits - https://github.com/vdumoulin/conv_arithmetic

Dealing with multiple input channels

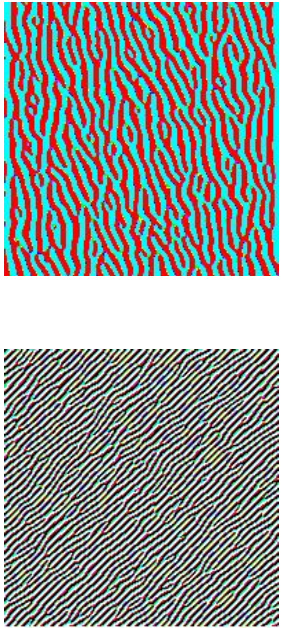


References: [12]

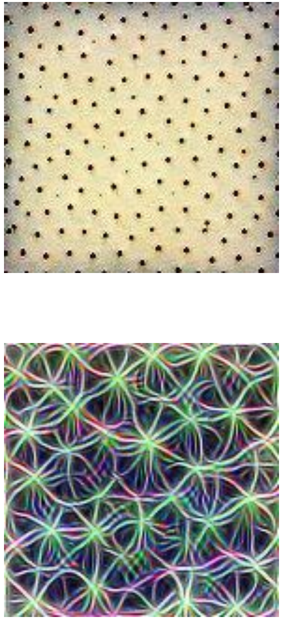
GoogLeNet on ImageNet - Feature visualization



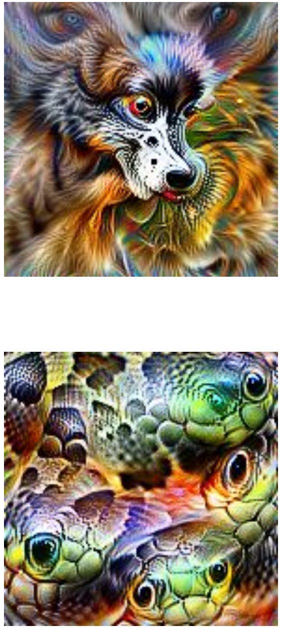
Stack multiple filters and learn kernels dynamically (hierarchy of features)



feature visualization of the 1s conv. layer



layer 3a



layer 4d

References: [8,9,10]

Problem 2

Digits recognition

References

[1] Pattern Recognition in a Bucket
https://link.springer.com/chapter/10.1007/978-3-540-39432-7_63

[2] Iris dataset: https://archive.ics.uci.edu/ml/datasets/iris

[3] Beer and diapers: http://www.dssresources.com/newsletters/66.php

[4] Multilayer feedforward networks are universal approximators:
http://cognitivemedium.com/magic_paper/assets/Hornik.pdf

[5] Adam: A Method for Stochastic Optimization: https://arxiv.org/abs/1412.6980

[6] MNIST dataset: http://yann.lecun.com/exdb/mnist/

References

[7] Bengio, Yoshua, Ian Goodfellow, and Aaron Courville. *Deep learning*. Vol. 1. MIT press, 2017: http://www.deeplearningbook.org/

[8] Feature-visualization: https://distill.pub/2017/feature-visualization/

[9] Going deeper with convolutions: https://arxiv.org/pdf/1409.4842.pdf

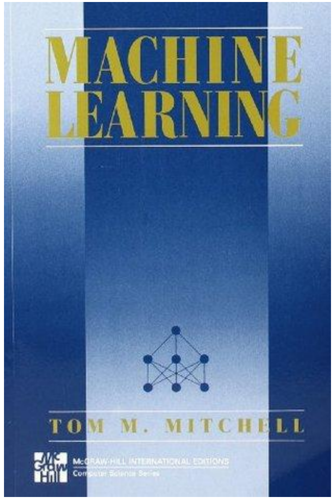
[10] Imagenet: A large-scale hierarchical image database: http://www.image-net.org/papers/imagenet_cvpr09.pdf

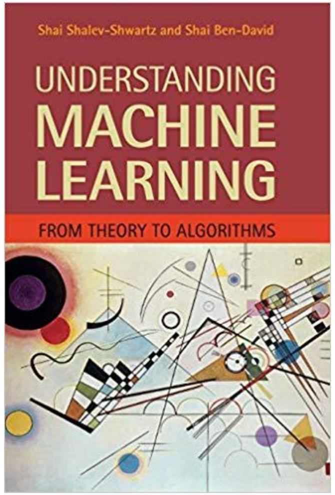
[11] Culture, Communication, and an Information Age Madonna: http://www.lenna.org/pcs_mirror/may_june01.pdf

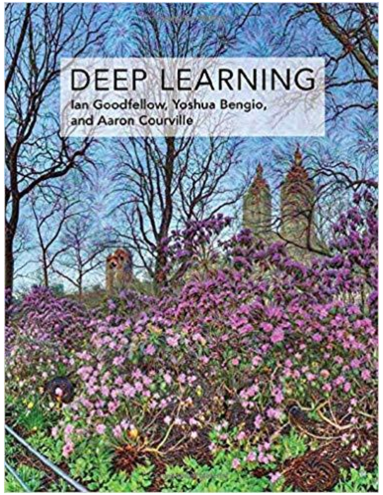
References

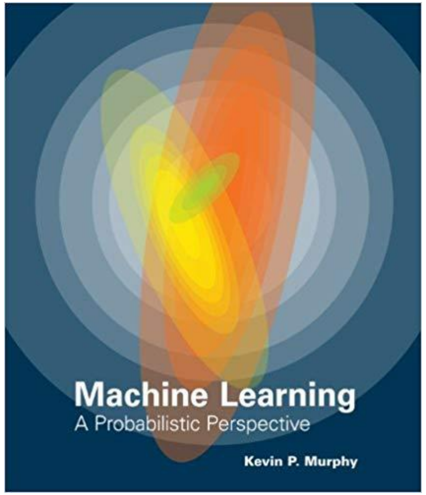
[12] Intuitively Understanding Convolutions for Deep Learning: https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1

Books




Shai Shalev-Shwartz and Shai Ben-David
UNDERSTANDING MACHINE LEARNING
FROM THEORY TO ALGORITHMS

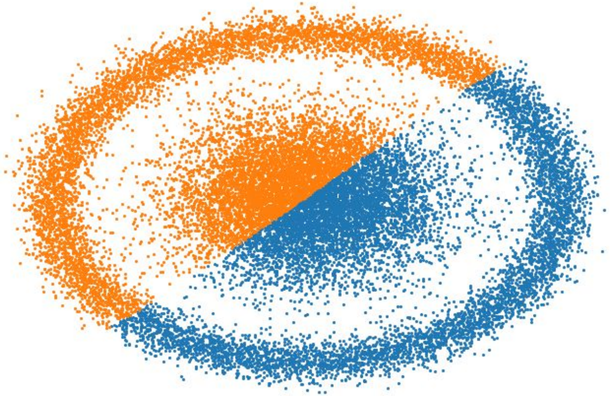




Difficulty →

Antonio Pitasi
Software Engineer, Nextworks
https://anto.pt

Samuele Sabella
https://github.com/samuelesabella



Leave your feedback

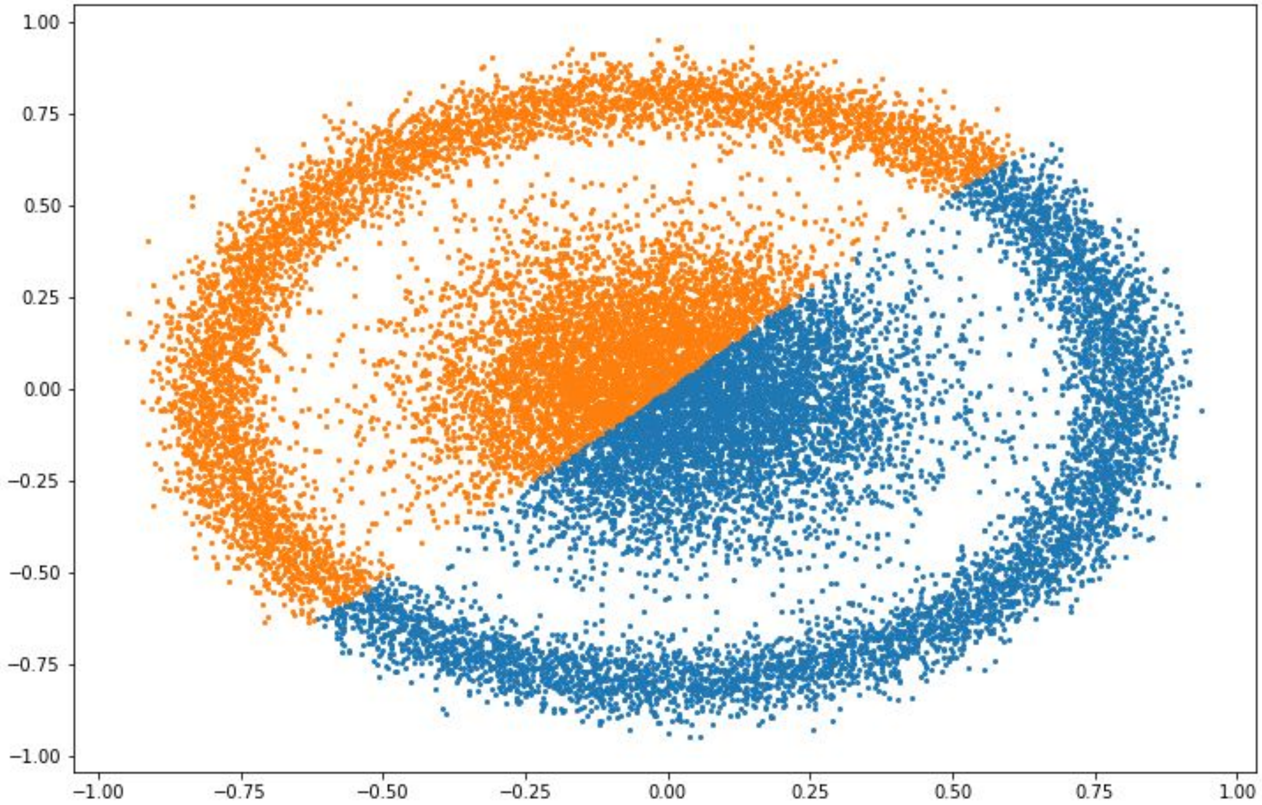
Antonio Pitasi

Software Engineer, Nextworks

https://anto.pt

Samuele Sabella

https://github.com/samuelesabella


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00