Computing and Physics

A few reflections about the role of computers in Physics
and life in occasion of prof. 임채호's retirement workshop

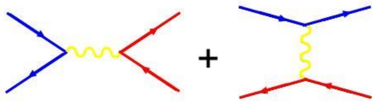
Stefano Scopel

서강대학교
SOGANG UNIVERSITY

Sogang University, February 26, 2020

OpenSource software in Physics and Cosmology http://pyflation.ianhuston.net/ (just some examples!)

MadGraph + MadEvent



Automated Tree-Level Feynman Diagram, Helicity Amplitude, and Event Generation

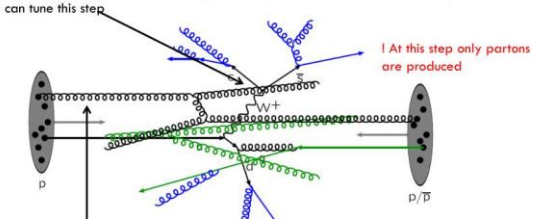
Tim Stelzer
Fabio Maltoni

PYTHIA Physics : ref 6.4 Manual

http://arxiv.org/abs/hep-ph/0603175

8

1) The first hard interaction is the first step of event machinery :
Computed in pQDC framework with factorization, possibility to select hard process : charm, bottom, jets, photon -> can tune this step



2) MPI (Multiple Parton Interaction) : other processes (soft or hard) can happen in parallel:
PYTHIA model : the first hard interaction is particular, other are reconstructed afterward, ordered in hardness, in PYTHIA 6, only g,u,d,s available in other interaction,
In PYTHIA 8 : second hard can include charm and bottom

Accelerator Physics

Dark SUSY

MicrOMEGAs: a code for the calculation of Dark Matter Properties including the **relic density**, direct and **indirect rates** in a general supersymmetric model and other models of New Physics

Astroparticle

Journal of Cosmology and Astroparticle Physics
An IOP and SISSA journal

Pyflation

CMBEASY: an object oriented code for the cosmic microwave background

Pyflation: a Python package for calculating cosmological perturbations during an inflationary expansion of the universe.

Cosmology

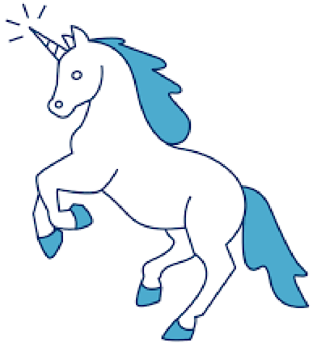
An alternative way to share knowledge?

Two examples of concepts that have changed the paradigm in computational physics:

- Object-oriented programming
- Machine learning

Object-oriented programming

- Any element in the code is an *object*. Trivial objects: floats, strings, arrays. For each of them we can do operations (sum two floats, compare two strings) or can use them them as inputs of functions.
- In object programming can *create* new, “exotic” objects, with their own properties and operations, that can be manipulated according to their nature



unicorn

dragon

An explicit example

Suppose you have to calculate a physical process that depends on a target nucleus

1. Create an "element" class
2. 'instantiate" as many time as needed to create many object belonging to the class "element":

```
>>> fluorine=PT.element('F')
>>> print(fluorine)
fluorine, symbol f, atomic number 9, average mass 17.689, 1 isotopes.
>>> carbon=PT.element('C')
>>> print(carbon)
carbon, symbol c, atomic number 6, average mass 11.182, 2 isotopes.
```

Now can write a routine that calculates a scattering cross section of a particle off the target:

```
>>> cross_section_carbon=cross_section(carbon,energy)
```

```
>>> cross_section_fluorine=cross_section(fluorine,energy)
```

(N.B. the "element" object carries all the needed properties, pass just one parameter)

3. Can define operation among elements to create more complicated targets:

```
>>> carbon_tetrafluoride=carbon+4*fluorine
```

```
>>> print(carbon_tetrafluoride)
```

N.B. The user is free to define what it means to sum two element, or to multiply an element times an integer
Also the output of print() is defined by the user

```
CF4 contains:
carbon, symbol c, atomic number 6, average mass 11.182, 2 isotopes.
Isotope-averaged mass: 81.938241fluorine, symbol f, atomic number 9, average mass 17.689, 1 isotopes.
Isotope-averaged mass: 81.938241
```

4.If the carbon_tetrafluoride object belongs to the same class as carbon as fluorine the routine cross section will handle it!

```
>>> cross_section_carbon_tetrafluoride=cross_section(carbon_tetrafluoride,energy)
```

Can extend this idea to many other concepts.

If the cross section depends on a Hamiltonian of the form:

$$H = \sum_i c_i(w_1, w_2, \ldots)\mathcal{O}_i$$

with $c_i(w_1, w_2, w_3,\ldots)$ some Wilson coefficients arbitrary functions of the parameters w_1, w_2, w_3, …and O_i some effective operators, can define a Hamiltonian object

```
>>> model1=WD.eft_hamiltonian('model1',wilson_coeff_list1)
>>> print(model1)
Hamiltonian name:model1
Hamiltonian:c_1(r)* Op_1+c_4(r)* Op_4
Squared couplings:c_1*c_1, c_4*c_4
```

```
>>> model2=WD.eft_hamiltonian('model2',wilson_coeff_list2)
>>> print(model2)
Hamiltonian name:model2
Hamiltonian:c_1(r)* Op_1+c_3(r)* Op_3
Squared couplings:c_1*c_1, c_3*c_1, c_3*c_3
```

N.B.: some operators may <u>interfere,</u> some may not, the class knows how to handle that automatically

Now the cross section routine may depend on both target and Hamiltonian:

```
>>> cross_section_fluorine_model2=cross_section(fluorine,model2)
>>> cross_section_carbon_model1=cross_section(carbon,model1)
```

Also in the case of the Hamiltonian objects can define operations such as multiplications times a constant or sum, etc…

$$H = 2H_1 + 3H_2 = 2\left(c_1\mathcal{O}_1 + c_4\mathcal{O}_4\right) + 3\left(c_1\mathcal{O}_1 + c_3\mathcal{O}_4\right) = 5c_1\mathcal{O}_1 + 3c_3\mathcal{O}_3 + 2c_4\mathcal{O}_4$$

```
>>> model=2*model1+3*model2
```

```
>>> print(model)
>>> Hamiltonian name:model
>>> Hamiltonian:5*c_1(r)*Op_1+2*c_3(r)*Op_3+3*c_4()*Op_4
>>> Squared couplings:25*c_1*c_1, 15*c_3*c_1, 4*c_3*c_3, 9*c_5*c_5
```

If 2*model1+3*model2 belongs to the same class as model1 and model2 the same cross section routine can handle it:

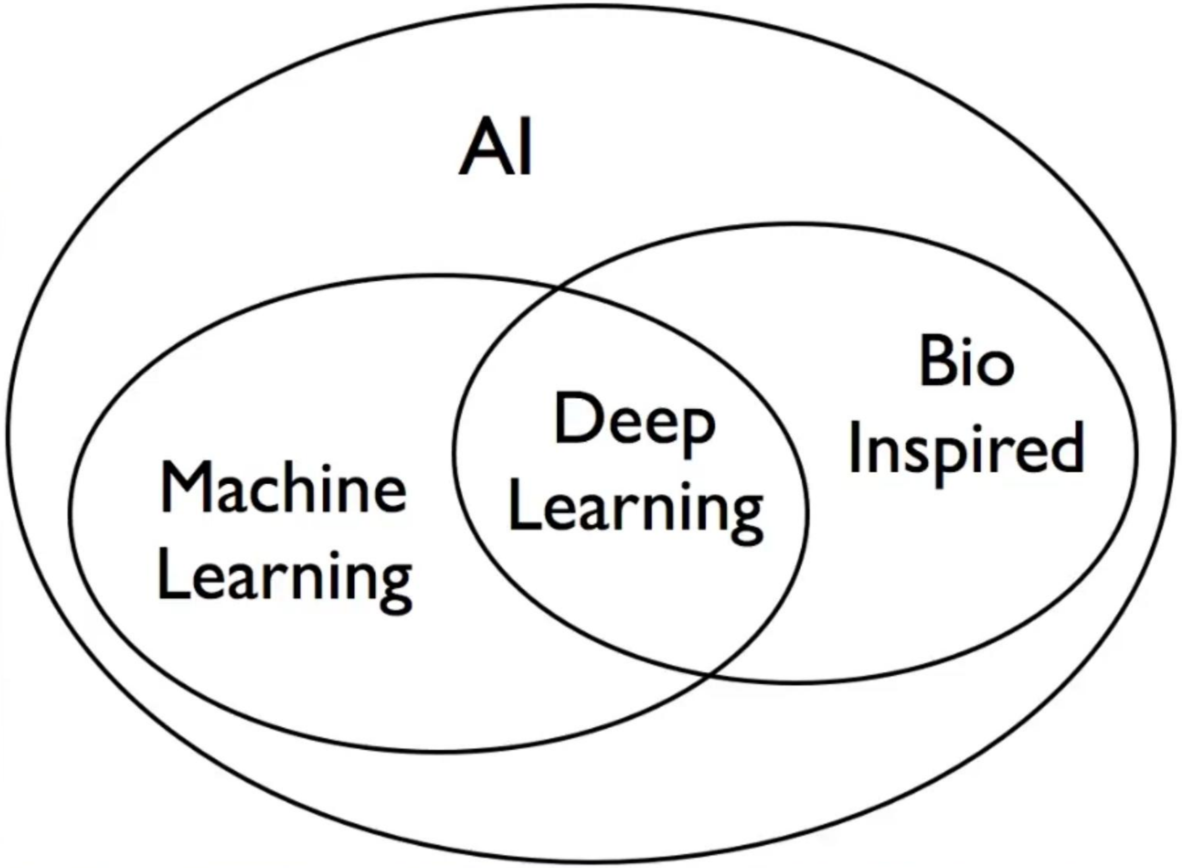
```
>>> carbon_tetrafluoride_model=cross_section(carbon_tetrafluoride,model)
```

Or, directly(!):

```
>>> carbon_tetrafluoride_model=cross_section(carbon+4*fluorine,2*model1+3*model2)
```

The key of object-oriented programming: once a certain operation is implemented and tested for a class of objects, the complexity of the objects can be arbitrary.

Onother important feature: inheritance, which allows to extend the properties of one class to another


AI
Machine Learning
Deep Learning
Bio Inspired

find title machine and learning | Brief format | Search | Easy Search | Advanced Search

find j "Phys.Rev.Lett.,105*" :: more

Search on INSPIRE beta

Sort by: earliest date | desc. | - or rank by -

Display results: 25 results | single list

HEP **391** records found 1 - 25 jump to record: 1 Search took 0.15 seconds.

1. **Reliable Photometric Membership (RPM) of Galaxies in Clusters. I. A Machine Learning Method and its Performance in the Local Universe**
   P.A.A. Lopes, A.L.B. Ribeiro. Feb 17, 2020. 14 pp.
   e-Print: **arXiv:2002.07263 [astro-ph.CO]** | **PDF**
   References | BibTeX | LaTeX(US) | LaTeX(EU) | Harvmac | EndNote
   ADS Abstract Service
   Detailed record

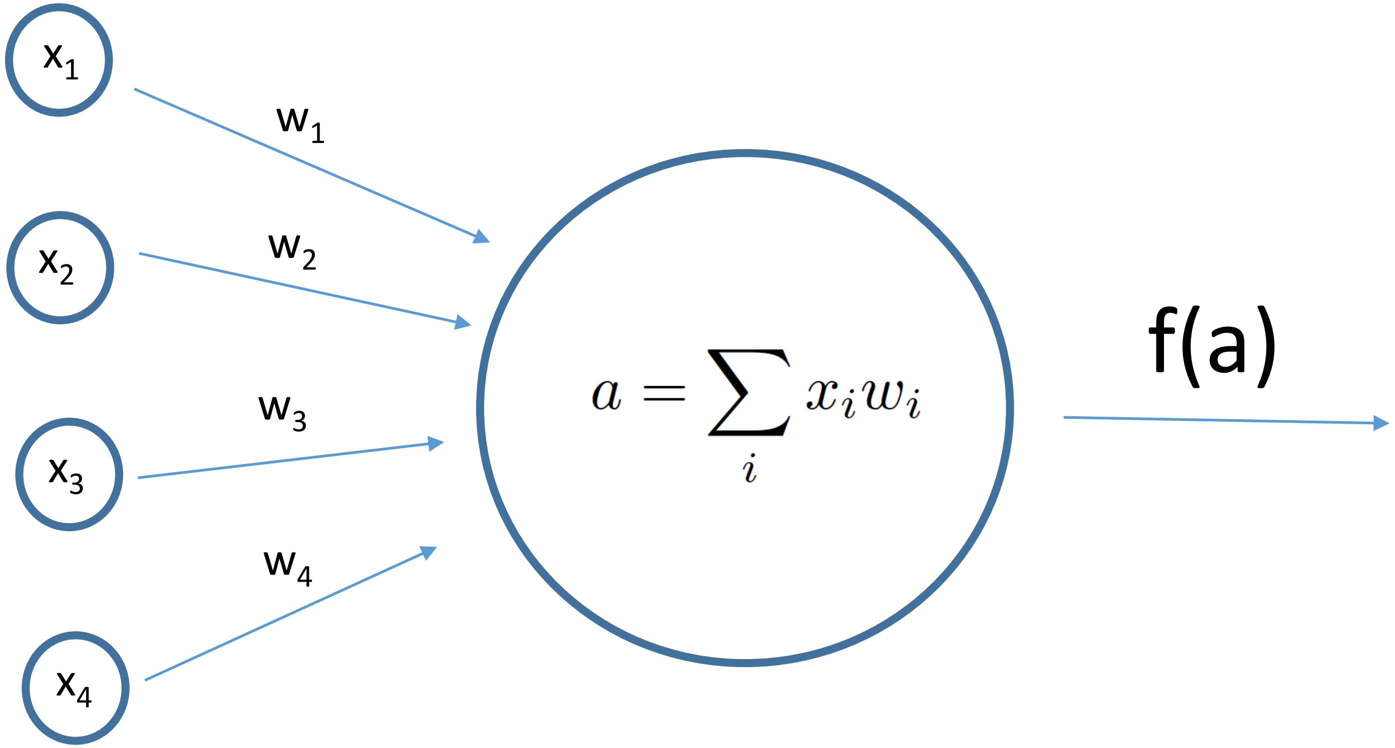
2. **Machine-learning-assisted insight into spin ice $Dy_2Ti_2O_7$**
   Anjana M. Samarakoon *et al.*. 2020.
   Published in **Nature Commun. 11 (2020) no.1, 892**
   DOI: 10.1038/s41467-020-14660-y
   References | BibTeX | LaTeX(US) | LaTeX(EU) | Harvmac | EndNote
   Detailed record

3. **Connecting Dualities and Machine Learning**
   Philip Betzler, Sven Krippendorf. Feb 12, 2020. 35 pp.
   LMU-ASC 05/20, MPP-2020-14

The building block of (artificial) intelligence: the neuron

The task of a neuron is to collect information from other neurons through its “synapses” and “decide” whether or not to “fire” an output to other neurons:

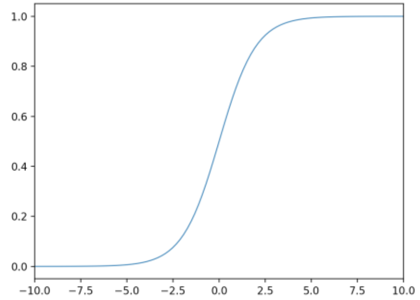
inputs: from other neurons



output: number between 0 and 1

popular choice for f: sigmoid

$$f(a) = \frac{1}{1+e^{-a}}$$



x_1, x_2, x_3, x_4 = inputs

w_1, w_2, w_3, w_4 = weights

$a = x_1 \cdot w_1 + x_2 \cdot w_2 + x_3 \cdot w_3 + x_4 \cdot w_4 = \Sigma_i x_i w_i$ =potential

f(a)= activation function (depends on potential a, “fires” a exceeds some threshold)

The neuron decision will depend on the weights (for instance if w=[1,0,0,0] only the first input will be taken into account). “Learning” means to choose the weights so that the output is the correct one.

Example: supervised learning

Provide many input examples with correct answer. Fit the weight w_i to minimize the error.

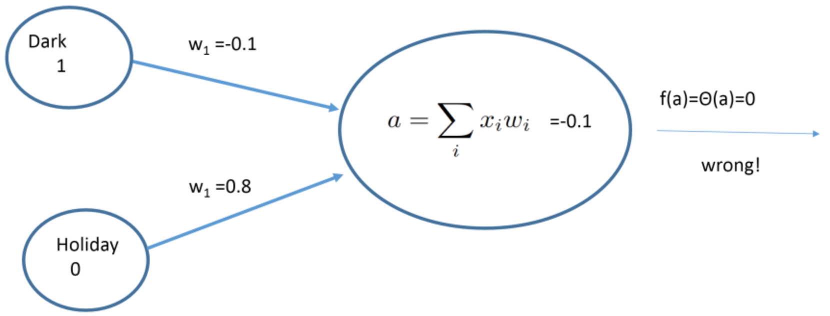
Suppose you connect a neuron to the light of a public room. You want to save energy, so the neuron should learn when it is the best moment to automatically shut off.

Here is the training data:

inputs		answers (labels)
Dark	Holiday	Light on?
Yes	Yes	No
Yes	No	Yes
No	Yes	No
No	No	No

Yes/no -> classificator (discrete number of possible answers)

At the beginning the neuron is “ignorant” → random weights w_i



Of course this is a trivial example. The output is: dark AND NOT holiday. If the neuron fires for a positive activation function we obtain the correct answer when $w_1>0$, $w_2<0$ and $|w_2|>|w_1|$:

$1*w_1+ 1*w_2<0$

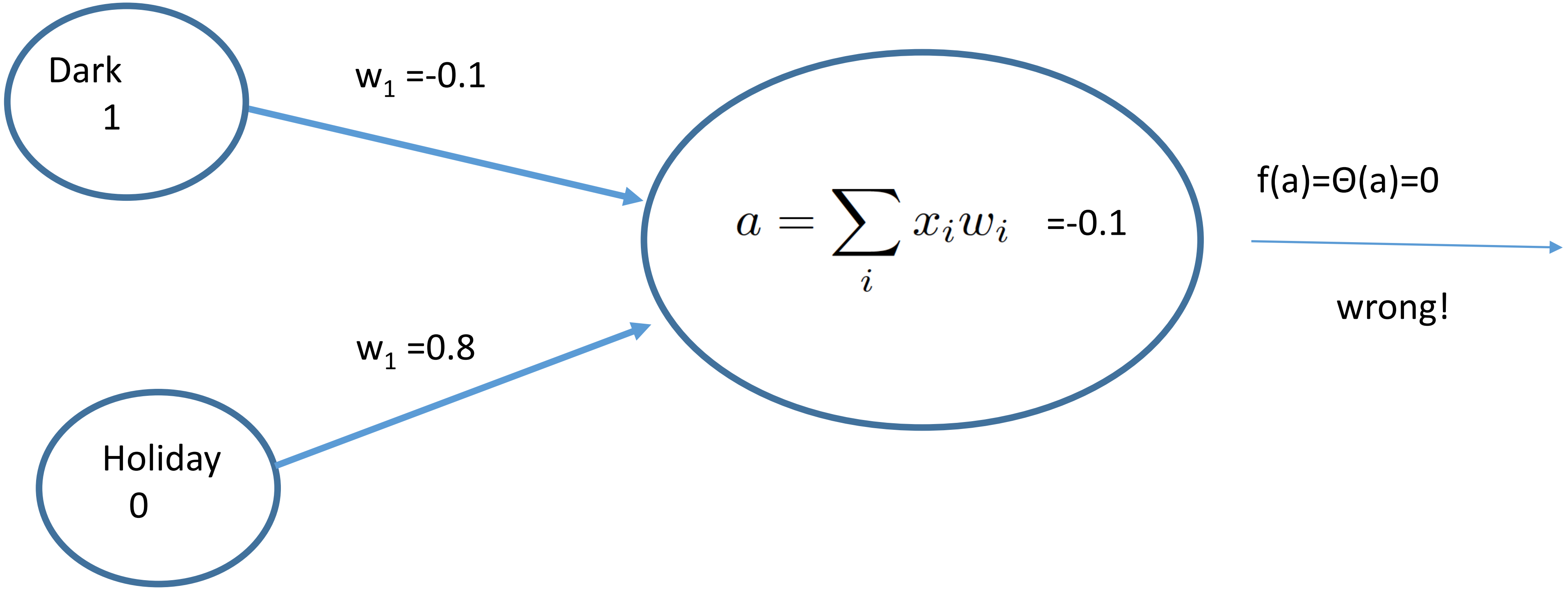
$1*w_1+ 0*w_2>0$ neuron fires when its dark and not holiday.

$0*w_1+ 1*w_2<0$

$0*w_1+ 0*w_2=0$

However, we want the code *to learn by itself*!

Each time we compare the output to the correct answer and change the weights. Each test is called “epoch”



- Compare to correct answer (1)
- Calculate error (1-0)
- Use error to “adjust“ the weights so that next time we do better

The process of adjusting the weights comparing the answers with the correct ones is called “back-propagation”

Delta rule:

$$\Delta w_i = \eta \cdot (t - o) \cdot x_i$$

$$w_i \to w_i + \Delta w_i$$

t=target
o=output
η=learning rate

Now we can teach the neuron.

```
>>> neuron=Neuron()
>>> X=[[1,1],[1,0],[0,1],[0,0]]
>>> y=[0,1,0,0]
>>> n_epochs=20
>>> for epoch in range(n_epochs):
...     print('epoch:'+str(epoch))
...     for i,x in enumerate(X):
...             target=y[i]
...             neuron.learn(x,target)
```

```
('epoch', 0)
('input', [1, 1], 'target', 0, 'error', 0.0)
('input', [1, 0], 'target', 1, 'error', 1.0)
('input', [0, 1], 'target', 0, 'error', -1.0)
('input', [0, 0], 'target', 0, 'error', 0.0)
('epoch', 1)
('input', [1, 1], 'target', 0, 'error', 0.0)
('input', [1, 0], 'target', 1, 'error', 1.0)
('input', [0, 1], 'target', 0, 'error', -1.0)
('input', [0, 0], 'target', 0, 'error', 0.0)
('epoch', 2)
('input', [1, 1], 'target', 0, 'error', 0.0)
('input', [1, 0], 'target', 1, 'error', 1.0)
('input', [0, 1], 'target', 0, 'error', -1.0)
('input', [0, 0], 'target', 0, 'error', 0.0)
('epoch', 3)
('input', [1, 1], 'target', 0, 'error', 0.0)
('input', [1, 0], 'target', 1, 'error', 1.0)
('input', [0, 1], 'target', 0, 'error', -1.0)
('input', [0, 0], 'target', 0, 'error', 0.0)
```

At the beginning the code gives wrong answers (error different from zero)
Weights keep changing

```
('epoch', 77)
('input', [1, 1], 'target', 0, 'error', 0.0)
('input', [1, 0], 'target', 1, 'error', 1.0)
('input', [0, 1], 'target', 0, 'error', 0.0)
('input', [0, 0], 'target', 0, 'error', 0.0)
('epoch', 78)
('input', [1, 1], 'target', 0, 'error', 0.0)
('input', [1, 0], 'target', 1, 'error', 1.0)
('input', [0, 1], 'target', 0, 'error', 0.0)
('input', [0, 0], 'target', 0, 'error', 0.0)
('epoch', 79)
('input', [1, 1], 'target', 0, 'error', 0.0)
('input', [1, 0], 'target', 1, 'error', 1.0)
('input', [0, 1], 'target', 0, 'error', 0.0)
('input', [0, 0], 'target', 0, 'error', 0.0)
('epoch', 80)
('input', [1, 1], 'target', 0, 'error', 0.0)
('input', [1, 0], 'target', 1, 'error', 0.0)
('input', [0, 1], 'target', 0, 'error', 0.0)
('input', [0, 0], 'target', 0, 'error', 0.0)
('epoch', 81)
('input', [1, 1], 'target', 0, 'error', 0.0)
('input', [1, 0], 'target', 1, 'error', 0.0)
('input', [0, 1], 'target', 0, 'error', 0.0)
('input', [0, 0], 'target', 0, 'error', 0.0)
```

Eventually the correct answers are reached. At this point the weights stop changing

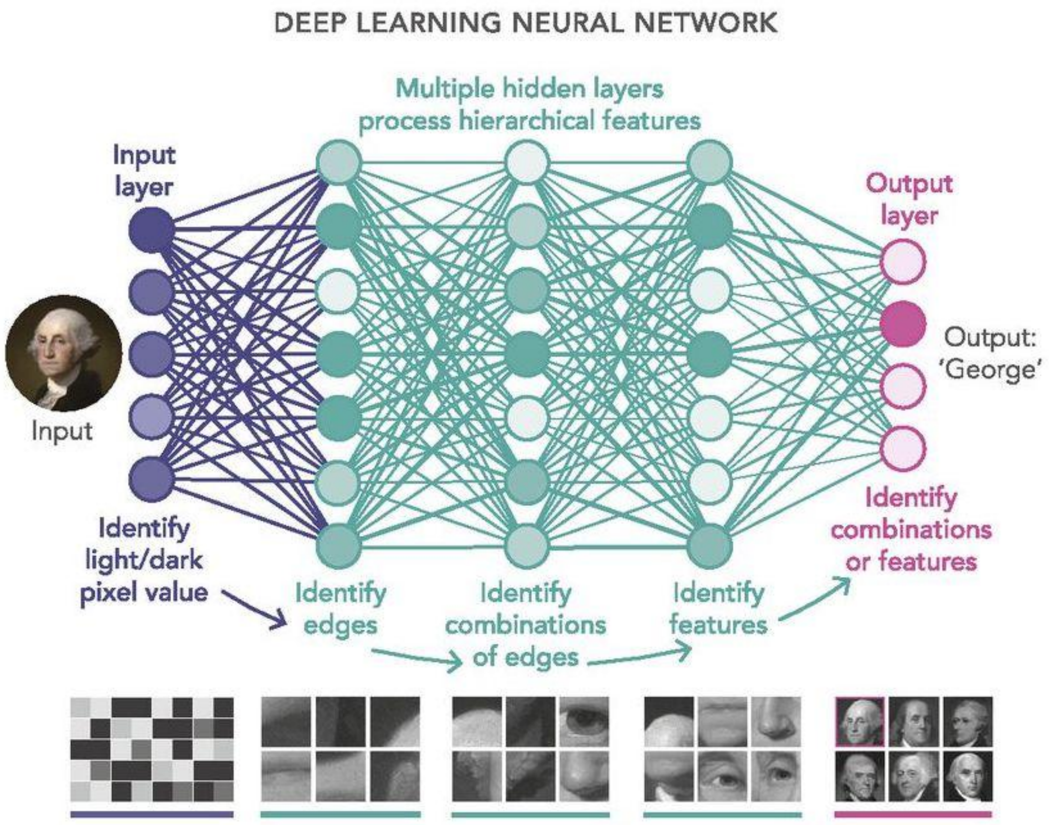
```
>>> print neuron.weights
[5.30825383648903e-16, -0.009999999999999999]
```

The neuron has learnt!

From now on we can use the neuron.

```
>>> neuron.output([1,0])
1.0
>>> neuron.output([0,0])
0.0
>>> neuron.output([0,1])
0.0
>>> neuron.output([1,1])
0.0
```

“Deep learning” refers to a large number of “layers” where neurons work in parallel



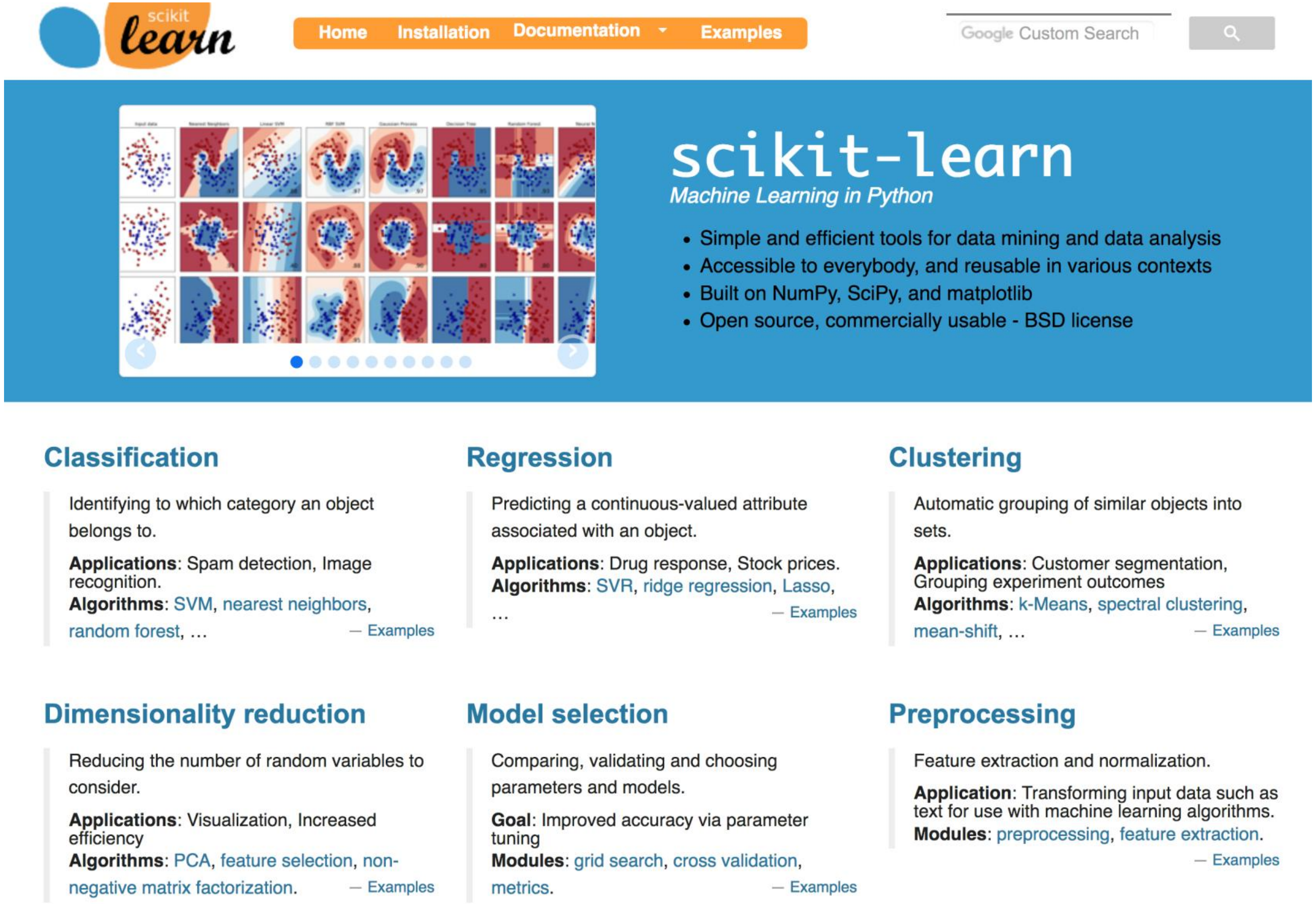
Each layer takes care of a different feature and contains many neurons
The number of weights can be HUGE
Need two ingredients:

- CPU power
- lots of DATA to train the code

N.B. today the data is the most valuable asset. Every time we use a free app with our mobile phone we are paying with our data (our preferences, our behavior, our displacements….)

In Machine Learning the critical issue is usually the data. Is it good quality? Is there noise? Is there some bias? Sometime unreadable/missing entries: how to handle them? So usually the Machine Learning itself is taken care by some library, used as a “black box”. The hard work is the preparation of the input and the analysis of the output to estimate if the code is working in a satisfactory way or not.

Scikit-learn: a set of libraries widely used and tested to do machine learning



scikit learn

Home Installation Documentation Examples

Google Custom Search

scikit-learn

Machine Learning in Python

- Simple and efficient tools for data mining and data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

Classification

Identifying to which category an object belongs to.

Applications: Spam detection, Image recognition.
Algorithms: SVM, nearest neighbors, random forest, ... — Examples

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.
Algorithms: SVR, ridge regression, Lasso, ... — Examples

Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, Grouping experiment outcomes
Algorithms: k-Means, spectral clustering, mean-shift, ... — Examples

Dimensionality reduction

Reducing the number of random variables to consider.

Applications: Visualization, Increased efficiency
Algorithms: PCA, feature selection, non-negative matrix factorization. — Examples

Model selection

Comparing, validating and choosing parameters and models.

Goal: Improved accuracy via parameter tuning
Modules: grid search, cross validation, metrics. — Examples

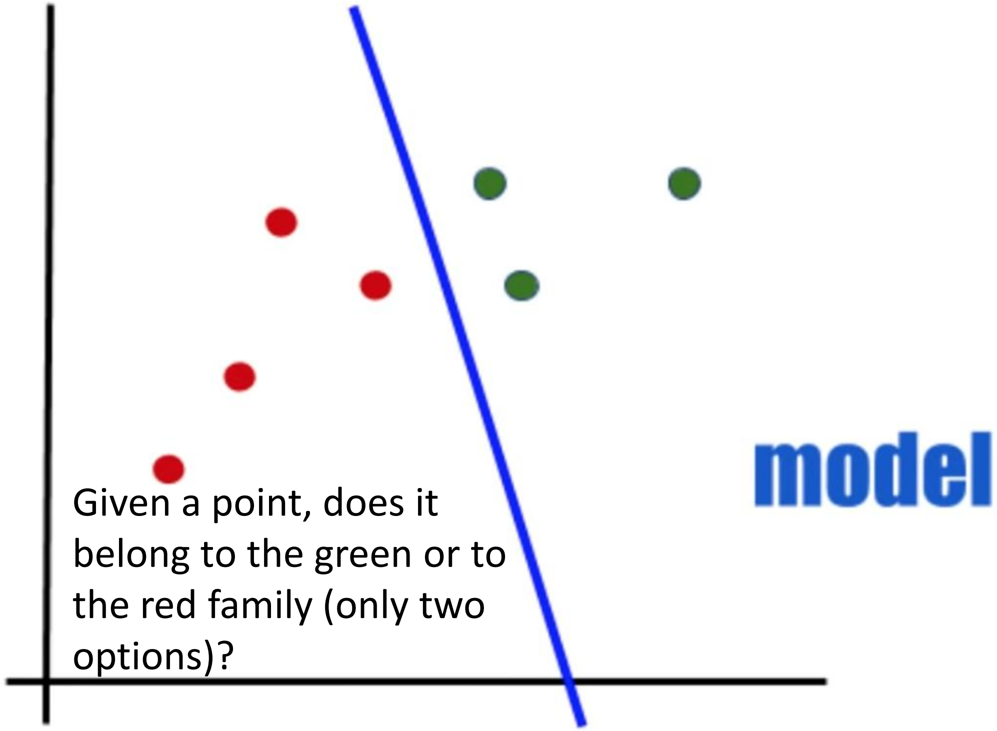
Preprocessing

Feature extraction and normalization.

Application: Transforming input data such as text for use with machine learning algorithms.
Modules: preprocessing, feature extraction. — Examples

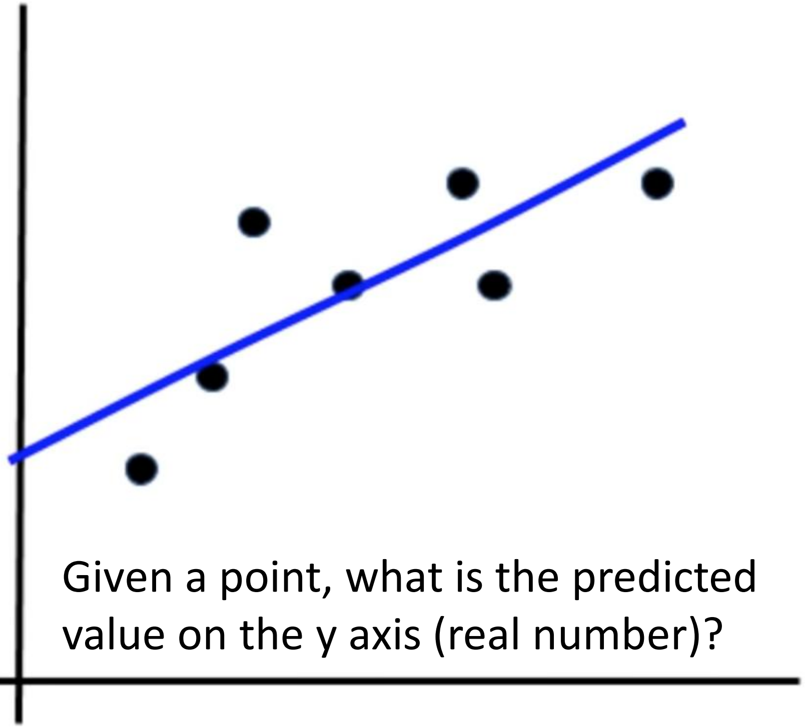
Machine Learning algorithms can be divided in two big families: classificators and regressors

Classification



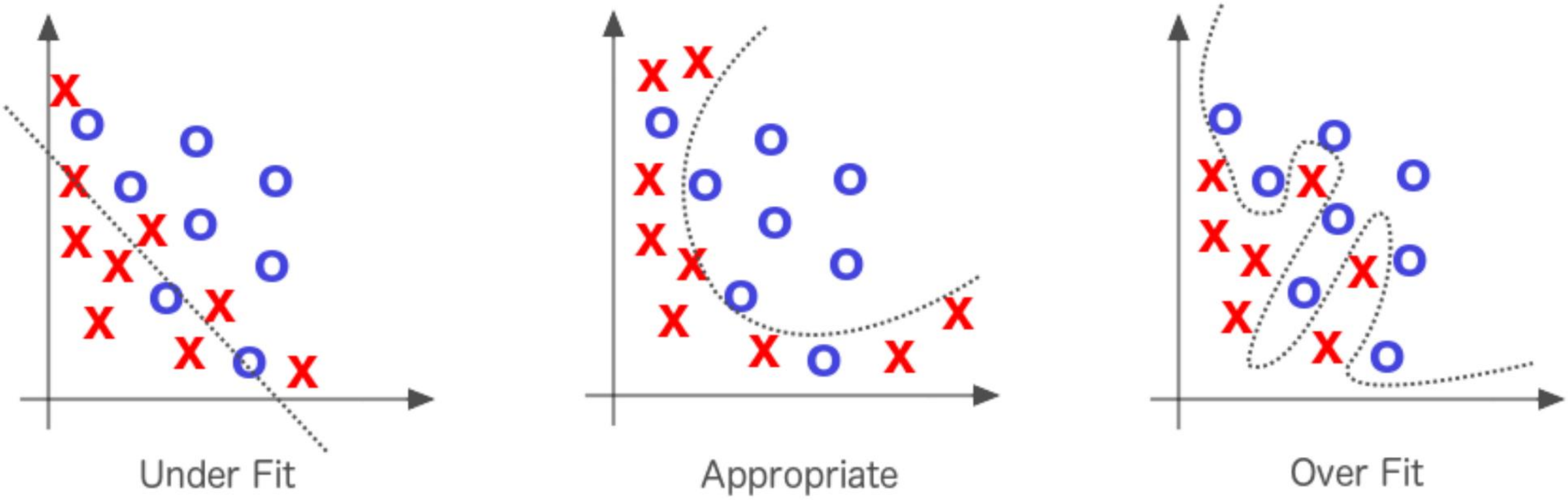
Classificator: the output is a category, discrete no. of possibilities
Ex: good or bad, a letter of the alphabet (only 26 possibilities), a digit (from 0 to 9).

Regression



Regressor: the output is a real number
Ex: a price, a person's life expectancy, mm of rain

Underfitting and overfitting



- Underfit: the model is too simple (wrong algorithm?)
- Overfit: the model captures noise. Excellent accuracy with training data, very bad with new data

Understanding overfitting: the analysis of random data (nothing to learn from them!)

Create some **totally random** data. For instance:

```
ntot=1000
features=np.random.rand(ntot,5)
labels=np.random.choice([0,1],ntot)
```

ntot 5-tuples containing random entries

ntot random answers (0 or 1)

The code should learn whether to return 1 or 0 based on 5 features per record. However the label are not correlated to the features, so there is no pattern behind them.

train on it a decision tree classifier and see what happens…

Accuracy score=fraction of correct answers

If you test the classificator on the SAME DATA that was used to train it:

```
accuracy_score on train sample:1.0
```

Good! (all answers are right)

If you test the classificator on DIFFERENT DATA:

```
accuracy_score on test sample:0.524
```

Bad! (practically half of answers are wrong)

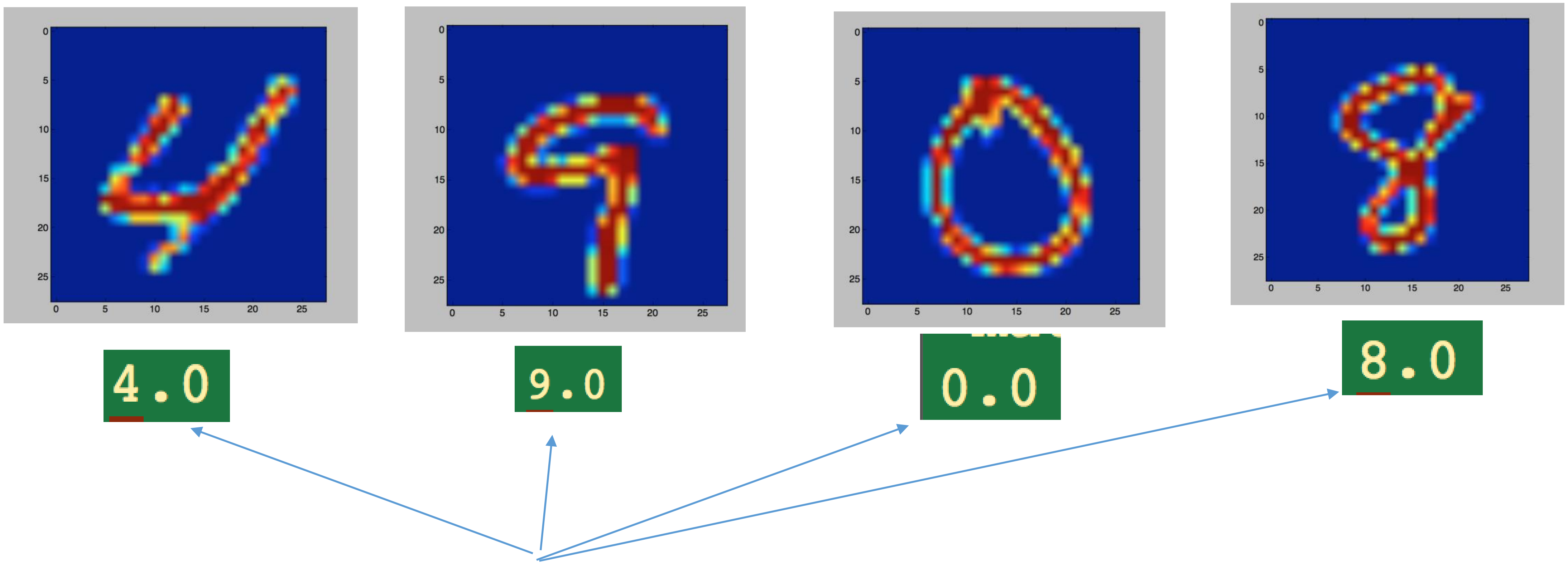
What happened? Simply the classificator has "memorized" the TRAIN data in the weights (one-to-one correspondence, this means that there were enough weights to do that!). However after training the model is just equivalent to the TRAIN data, it is only a complicated way to store them. In fact, if you try to use the model on the TEST data (that the model has never seen) the answers are random, as they should (half correct, half wrong).

IT IS FUNDAMENTAL NOT TO TEST MACHINE LEARNING ON THE SAME DATA USED FOR TRAINING

Data selection is crucial

Image recognition

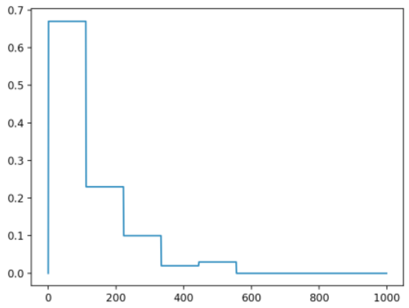
Use for training the mnist database. 60000 pictures for training and 10000 pictures for testing.
Each image is stored in a 784=20*28 array of float numbers between 0 and 1 that represent a color intensity.
Now instead of some features for each input (ex: size, position, etc of home to predict price) we have 28*28=784 different numbers (features) that characterize each pic. More complex but basically, more of the same!

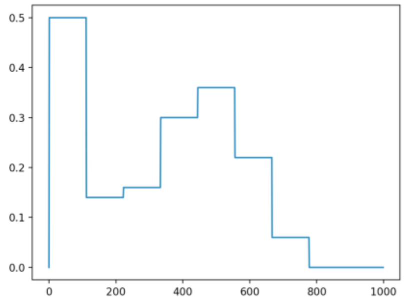


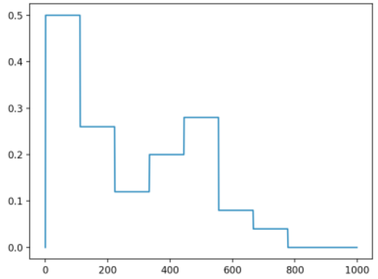
Answers from Machine learning after training

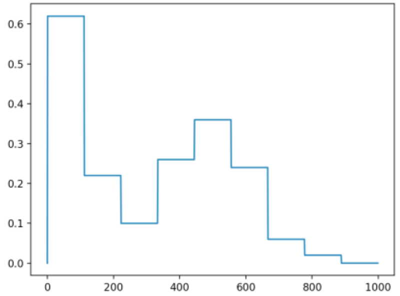
Finding peaks with machine learning

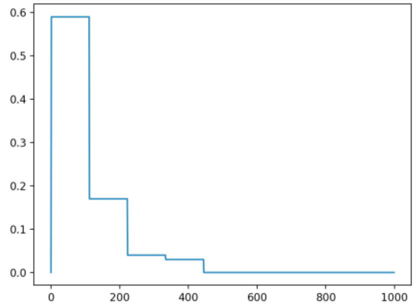
- Generate a large number of spectra, some with background+peak and some with only background
- Train a machine learning code to distinguish them
- Test the accuracy of the answer for different sizes of the peak or energy resolution











Is there a peak? (Machine Learning answer)

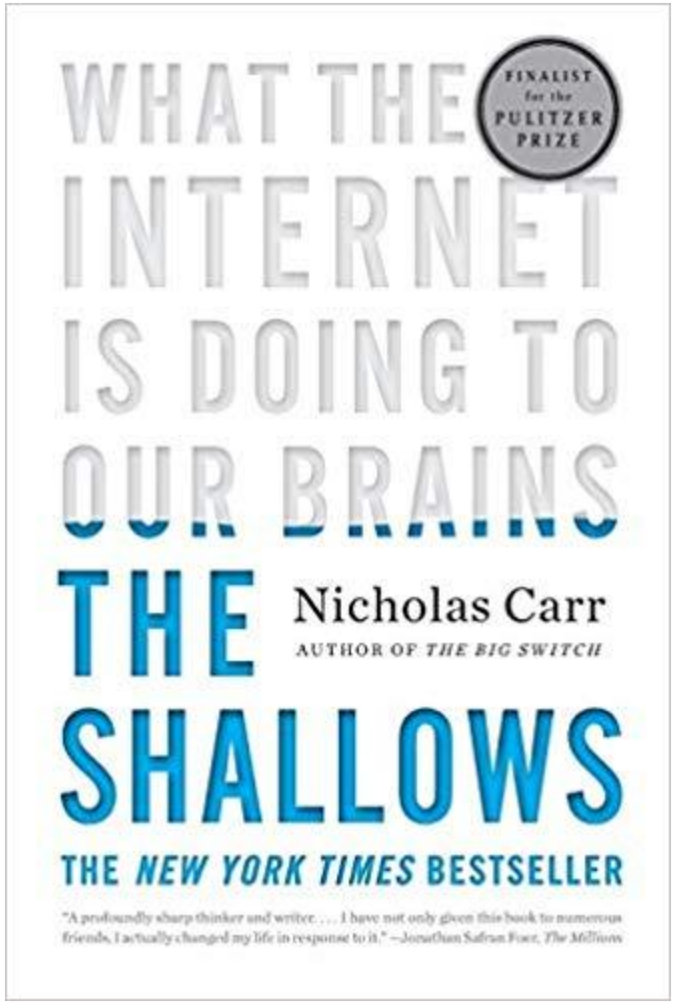
no yes yes yes no

Deceptively easy, the devil is in the detail. For *real* data

- Rescaling problems
- Bias
- Incompleteness
- Noise
- Systematics
- Etc…

can heavily affect the accuracy!

- Is the internet making us stupid?
- More boadly: are *computers* making us stupid?



- The answer from the book: not stupid, just different!

Probably we will loose our capacity to *memorize* things

But we will improve our *problem solving* skills

…and leave more repetitive an boring tasks to machines, to concentrate on more *interesting* problems

Dear Prof. 임채호,

best wishes for a healthy retirement full of interesting things to see and to do!