Machine learning and physical modelling-2

julien.brajard@nersc.no

21-25 January 2018

NERSC/Sorbonne University

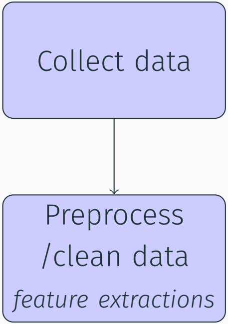
https://github.com/brajard/Geilo-Winter-school

Table of contents

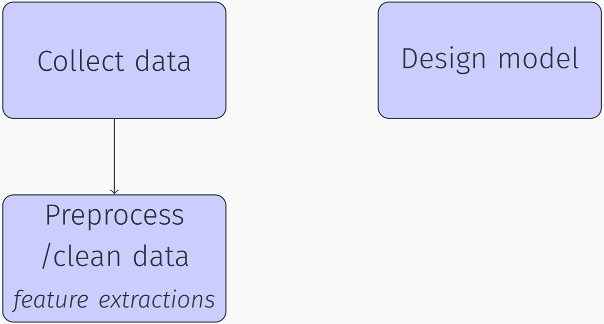
Steps of a machine learning process

Steps

Steps

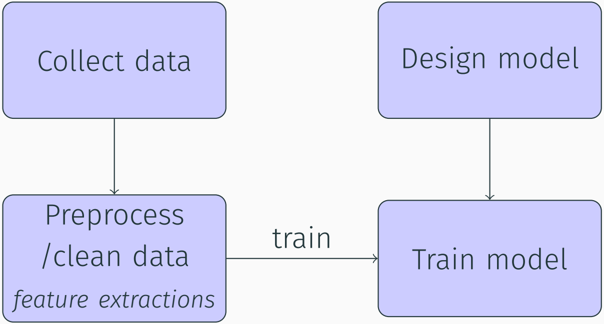


Steps

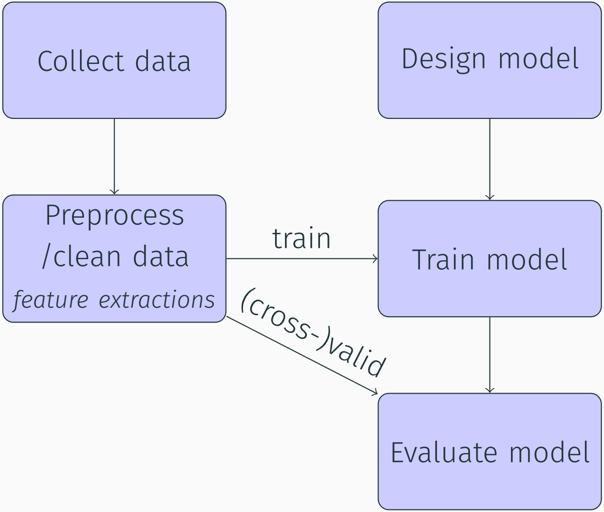


Collect data
Design model
Preprocess /clean data
feature extractions

Steps

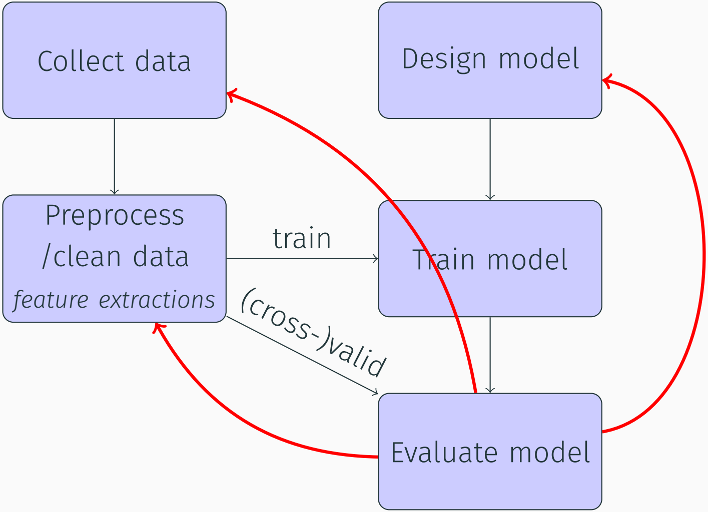

Collect data
Design model
Preprocess
/clean data
feature extractions
train
Train model

Steps

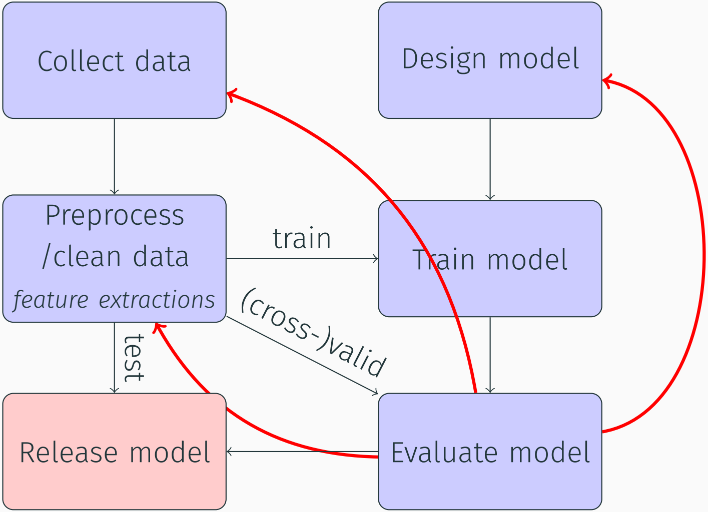


Collect data
Design model
Preprocess
/clean data
feature extractions
train
Train model
(cross-)valid
Evaluate model

Steps


Collect data
Design model
Preprocess
/clean data
feature extractions
train
Train model
(cross-)valid
Evaluate model

Steps


Collect data
Design model
Preprocess
/clean data
feature extractions
train
Train model
(cross-)valid
test
Release model
Evaluate model

In summary

From one dataset, 3 sub-datasets have to be extracted:

- A training dataset
- A validation dataset

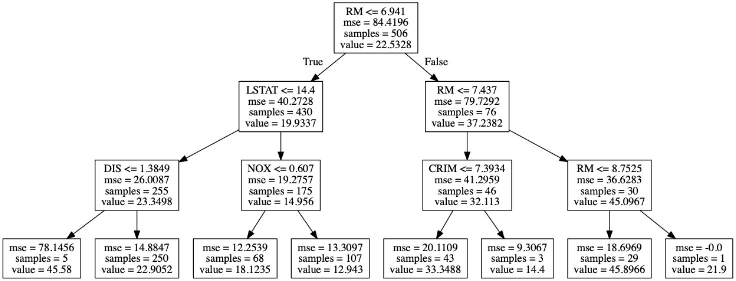
Can be done iteratively in a cross-validation procedure. Some parameters of the model (e.g. polynomial order in a polynomial regression) were determined from the validation dataset.

- A test dataset (independent from the two other) to estimate the final performance of the model.

A standard Machine learning model: Random Forests

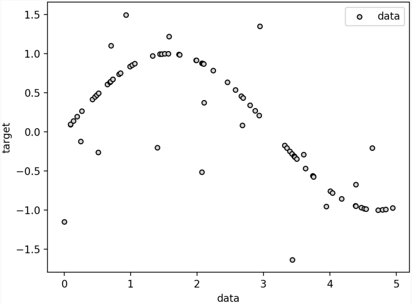
A decision tree

Predict house price (in $1000's) from 13 features:

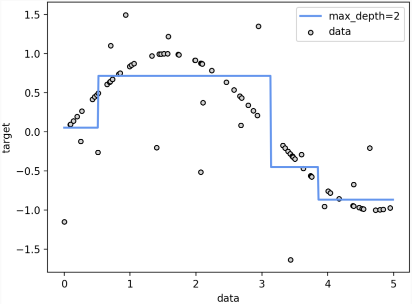


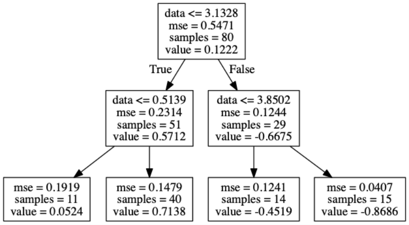
CRIM	per capita crime rate by town
NOX	nitric oxides concentration
RM	average number of rooms per dwelling
DIS	distance to employment centres
LSTAT	lower status of the population

Uni-variate example

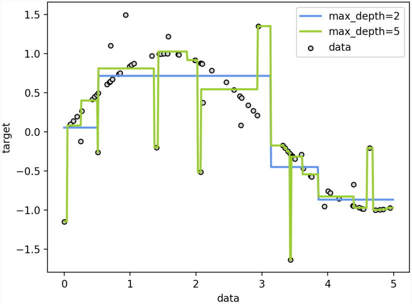


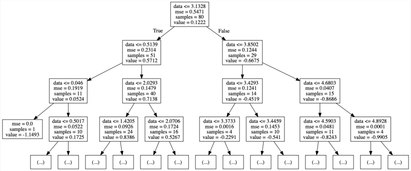
Uni-variate example





Uni-variate example





From tree to forest

Disadvantages of regression tree:

- Can overfit the data

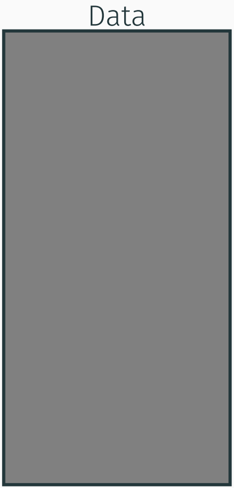
From tree to forest

Disadvantages of regression tree:

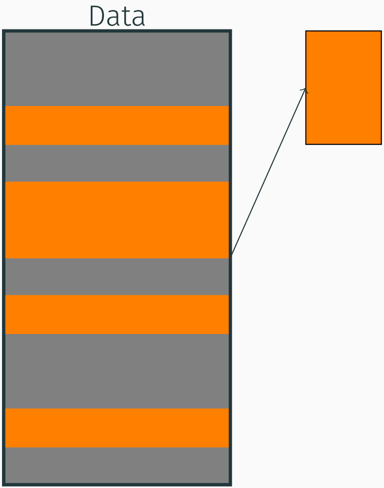
- Can overfit the data

One extension of Regression Tree: Random Forest

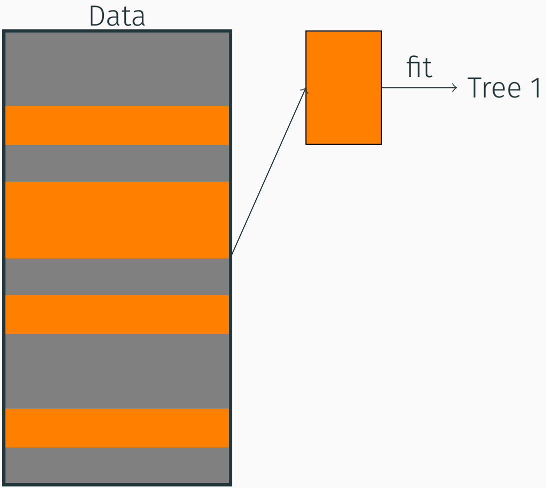
The (over simplified) principle of Random Forest



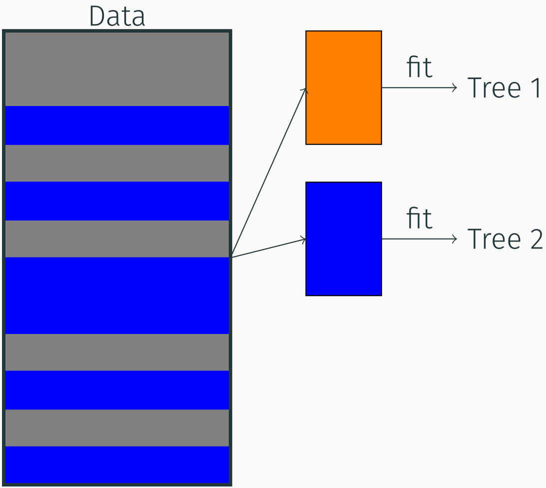
The (over simplified) principle of Random Forest



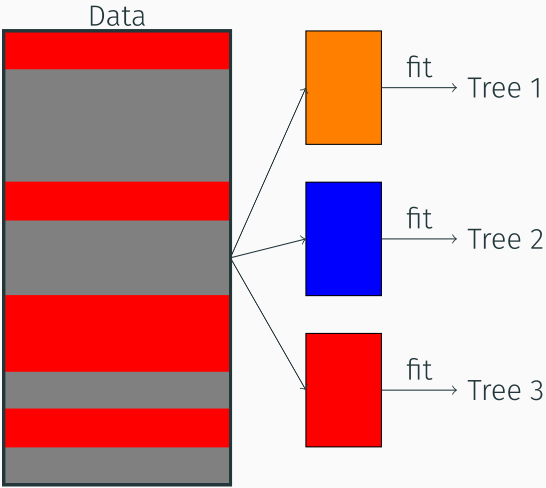
The (over simplified) principle of Random Forest



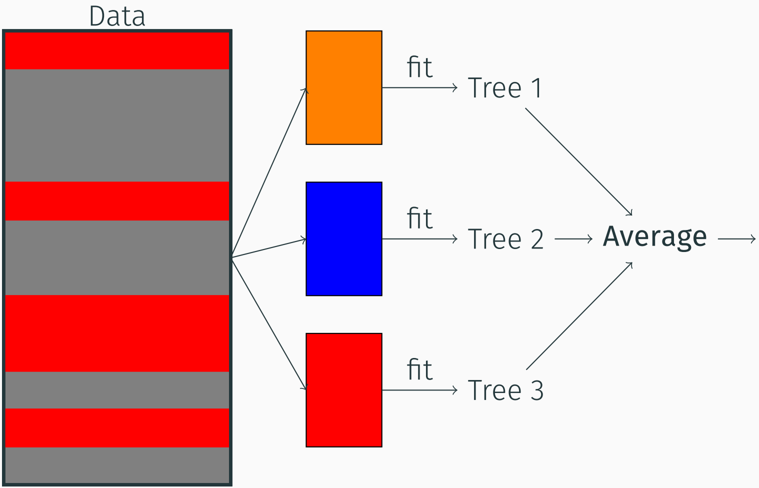
The (over simplified) principle of Random Forest



The (over simplified) principle of Random Forest

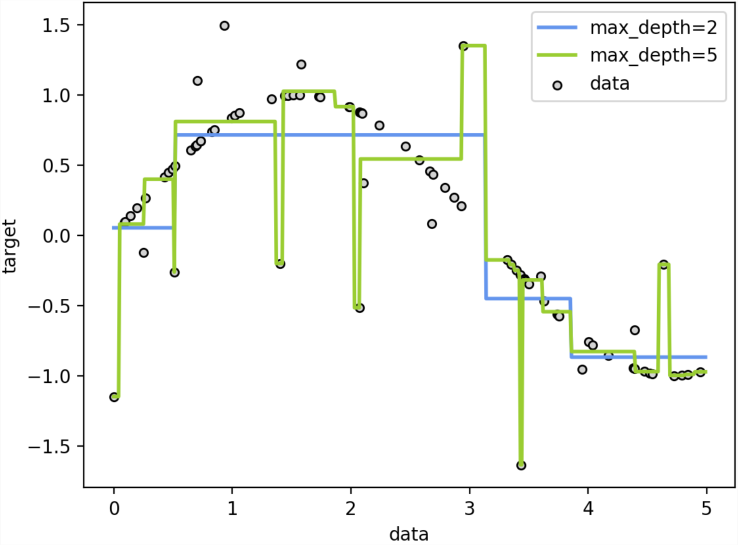


The (over simplified) principle of Random Forest

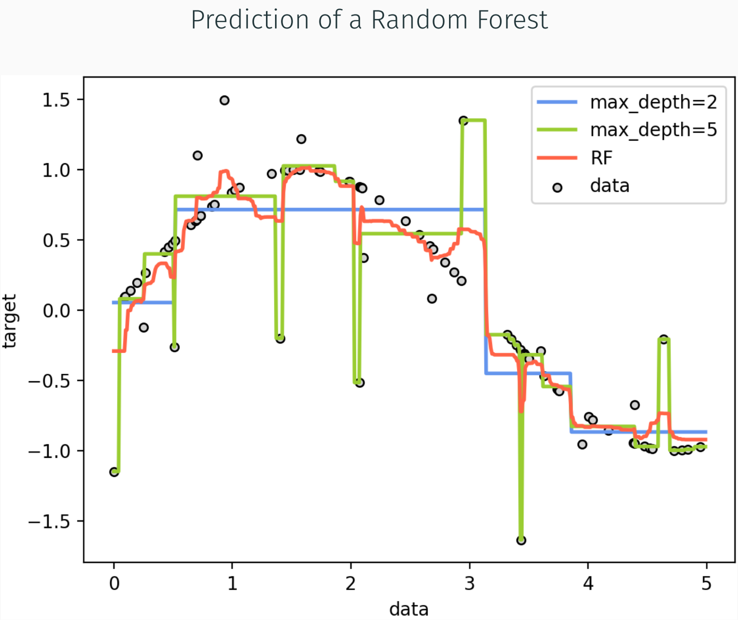


Results on the univariate experiment

Prediction of Randoms trees



Results on the univariate experiment



Some key parameters

```python
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestRegressor(n_estimators=n, max_features=
    maxf, min_samples_split=min_split, ...)
```

- `n_estimators`: number of trees (generally the larger is the better)

Some key parameters

```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestRegressor(n_estimators=n, max_features=
    maxf, min_samples_split=min_split, ...)
```

- `n_estimators`: number of trees (generally the larger is the better)
- `max_features`: number of features to consider at each split. The default number is the total number of features. A larger value makes provides a smaller bias (accuracy) but a bigger variance (risk of overfitting)

Some key parameters

```python
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestRegressor(n_estimators=n, max_features=
    maxf, min_samples_split=min_split, ...)
```

- `n_estimators`: number of trees (generally the larger is the better)
- `max_features`: number of features to consider at each split. The default number is the total number of features. A larger value makes provides a smaller bias (accuracy) but a bigger variance (risk of overfitting)
- `min_samples_fit`: number of features to consider at each split. The minimum value of 2 means that the tree is fully developed (small bias but great variance).

Determination of the parameters

- Parameters that are not optimized during the training are called hyper parameters.

Determination of the parameters

- Parameters that are not optimized during the training are called hyper parameters.
- They can be determined using a score on the validation dataset or using a cross-validation procedure.

Determination of the parameters

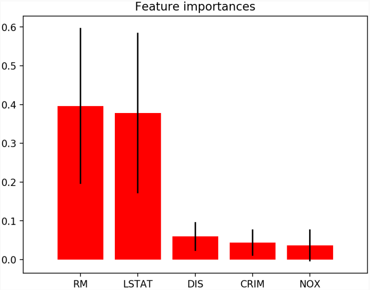
- Parameters that are not optimized during the training are called hyper parameters.
- They can be determined using a score on the validation dataset or using a cross-validation procedure.
- A convenient (but very costly) procedure in scikit-learn: gridsearch.

Example on notebook

Feature importance

```
rf = RandomForestRegressor(n_estimators=1000,
    max_features=10,random_state=10)
rf.fit(X,y)
importances = rf.feature_importances_
```

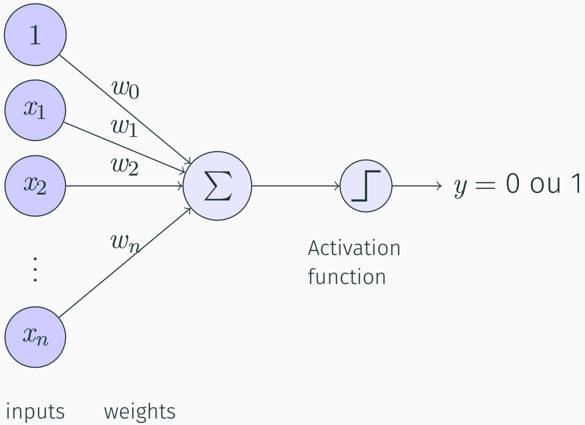
Indicates the impact of a feature in predicting the target.



CRIM	per capita crime rate by town
NOX	nitric oxides concentration
RM	average number of rooms per dwelling
DIS	distance to employment centres
LSTAT	lower status of the population

Neural Networks

The perceptron : an artificial neuron



Computation

$$y = f(w_0 + w_1.x_1 + w_2.x_2 + \cdots + w_n.x_n) = f(w_0 + \sum_{i=1}^{n} w_i.x_i)$$

Some remarks

- Inputs x_i are the different features of the data

Some remarks

- Inputs x_i are the different features of the data
- Weight w_i are the parameters of the model to optimize

Some remarks

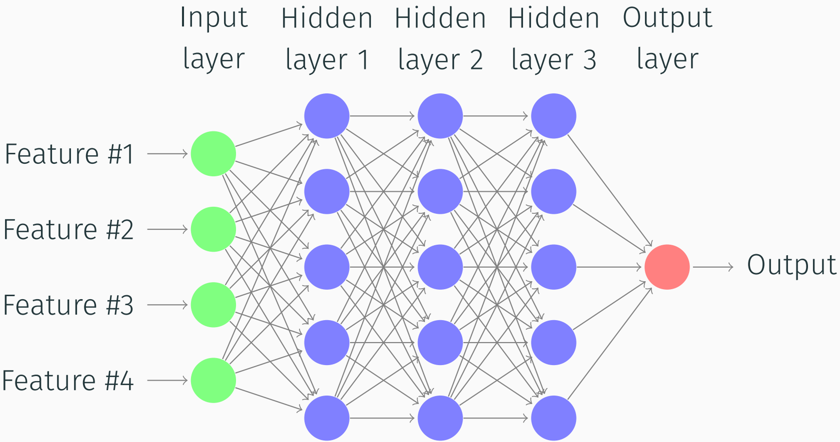
- Inputs x_i are the different features of the data
- Weight w_i are the parameters of the model to optimize
- If the activation function is identity, it is equivalent to a linear regression

Some remarks

- Inputs x_i are the different features of the data
- Weight w_i are the parameters of the model to optimize
- If the activation function is identity, it is equivalent to a linear regression

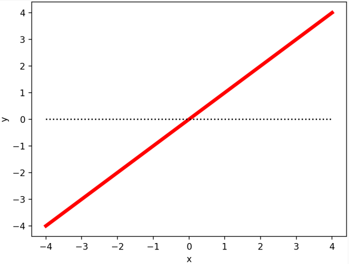
> More complexe models are build by combining several perceptrons

Multi-layer perceptron

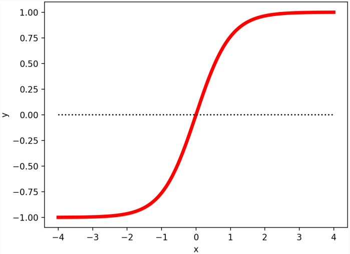


More usual activation functions

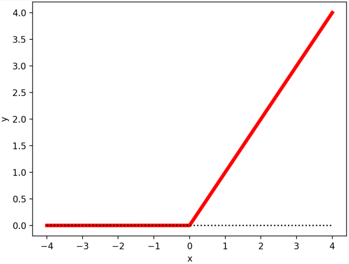
Linear



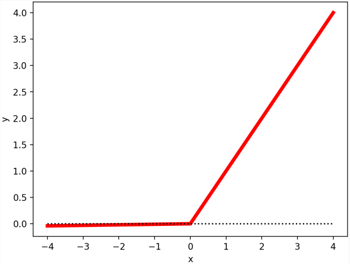
Hyperbolic tangent



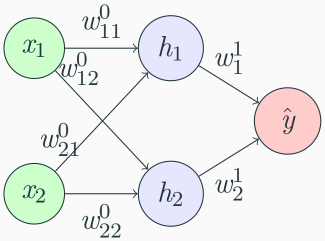
ReLU



Leaky-ReLU



Training a neural-net: gradient backpropagation

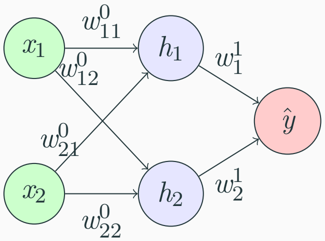


Objective

Determination of the best set of weights $\mathbf{w}$ to minimize the Loss function $L = ||\hat{y} - y||$.
Gradient descent algorithms based on $\partial L / \partial w$

1. Given a couple (x, y)

Training a neural-net: gradient backpropagation

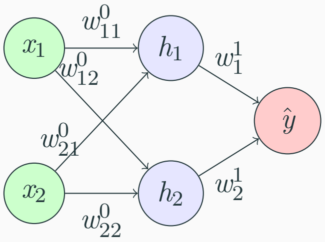


1. Given a couple (x, y)
2. Forward computation:
   $h_j = f_0(\sum_{i=1}^{2} w^0_{ij}.x_i)$
   $\hat{y} = f_1(\sum_{j=1}^{2} w^1_j.h_j)$

Objective

Determination of the best set of weights $\mathbf{w}$ to minimize the Loss function $L = ||\hat{y} - y||$.
Gradient descent algorithms based on $\partial L / \partial w$

Training a neural-net: gradient backpropagation

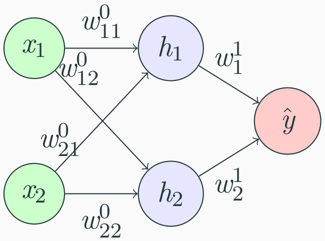


1. Given a couple (x, y)
2. Forward computation:
   $h_j = f_0(\sum_{i=1}^2 w^0_{ij}.x_i)$
   $\hat{y} = f_1(\sum_{j=1}^2 w^1_j.h_j)$
3. Compute the gradient of the loss:
   $\boxed{\partial L / \partial \hat{y}}$

Objective

Determination of the best set of weights $\mathbf{w}$ to minimize the Loss function $L = ||\hat{y} - y||$.
Gradient descent algorithms based on $\partial L / \partial w$

Training a neural-net: gradient backpropagation

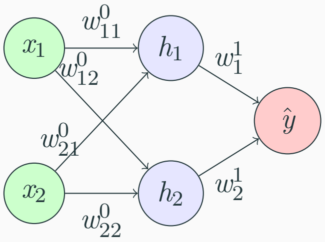


Objective

Determination of the best set of weights $\mathbf{w}$ to minimize the Loss function $L = ||\hat{y} - y||$.
Gradient descent algorithms based on $\partial L / \partial w$

1. Given a couple (x, y)
2. Forward computation:
   $h_j = f_0(\sum_{i=1}^{2} w^0_{ij}.x_i)$
   $\hat{y} = f_1(\sum_{j=1}^{2} w^1_j.h_j)$
3. Compute the gradient of the loss:
   $\boxed{\partial L / \partial \hat{y}}$
4. Gradient Backpropagation:

Training a neural-net: gradient backpropagation

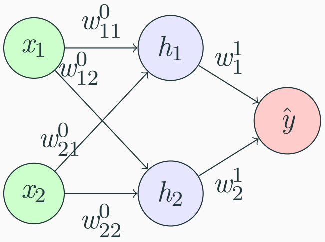


Objective

Determination of the best set of weights $\mathbf{w}$ to minimize the Loss function $L = ||\hat{y} - y||$.
Gradient descent algorithms based on $\partial L / \partial w$

1. Given a couple (x, y)
2. Forward computation:
   $h_j = f_0(\sum_{i=1}^2 w^0_{ij}.x_i)$
   $\hat{y} = f_1(\sum_{j=1}^2 w^1_j.h_j)$
3. Compute the gradient of the loss:
   $\boxed{\partial L / \partial \hat{y}}$
4. Gradient Backpropagation:
   - Layer 1
     $\partial L / \partial w^1_j = \boxed{\partial L / \partial \hat{y}}.\partial f_1 / \partial w^1_j$
     $\boxed{\partial L / \partial h_j} = \boxed{\partial L / \partial \hat{y}}.\partial f_1 / \partial h_j$

Training a neural-net: gradient backpropagation



Objective

Determination of the best set of weights $\mathbf{w}$ to minimize the Loss function $L = ||\hat{y} - y||$.
Gradient descent algorithms based on $\partial L/\partial w$

1. Given a couple (x, y)
2. Forward computation:
   $h_j = f_0(\sum_{i=1}^{2} w^0_{ij}.x_i)$
   $\hat{y} = f_1(\sum_{j=1}^{2} w^1_j.h_j)$
3. Compute the gradient of the loss:
   $\boxed{\partial L/\partial \hat{y}}$
4. Gradient Backpropagation:
   - Layer 1
     $\partial L/\partial w^1_j = \boxed{\partial L/\partial \hat{y}}.\partial f_1/\partial w^1_j$
     $\boxed{\partial L/\partial h_j} = \boxed{\partial L/\partial \hat{y}}.\partial f_1/\partial h_j$
   - Layer 0
     $\partial L/\partial w^0_{ij} = \boxed{\partial L/\partial h_j}.\partial f_1/\partial w^0_{ij}$

Classification and regression loss

Regression

- Last layer:
  linear or hyperbolic
  tangent
- Loss function:

$$L(\hat{y}, y) = \sum_i (\hat{y}_i - y_i)^2$$

Classification and regression loss

Regression

- Last layer:
  linear or hyperbolic tangent
- Loss function:

$$L(\hat{y}, y) = \sum_i (\hat{y}_i - y_i)^2$$

Classification

- Last layer:
  Soft-max

$$p_j = f_j(\mathbf{h}) = \frac{e^{h_j}}{\sum_k e^{h_k}}$$

- Loss function:
  Negative crossentropy

$$L(p, y) = -\sum_i \sum_j y_{i,j}. \log p_{i,j}$$

Batch/Stochastic training

Dataset: (X, y) with N samples, $\mathbf{w}$: initial weights

Batch Training:

- For i from 1 to N:
    1. $L = L + L(f(x_i), y_i)$
- Calculate $\partial L / \partial w$
- update weights:$\mathbf{w}$

1 Update is performed after N forward passes of the neural net.

Batch/Stochastic training

Dataset: (X, y) with N samples, $\mathbf{w}$: initial weights

Batch Training:

- For i from 1 to N:
  1. $L = L + L(f(x_i), y_i)$
- Calculate $\partial L / \partial w$
- update weights:$\mathbf{w}$

1 Update is performed after N forward passes of the neural net.

Stochastic Training:

- For i from 1 to N:
  1. Compute $L(f(x_i), y_i)$
  2. Calculate $\partial L / \partial w$
  3. update weights:$\mathbf{w}$

N Updates are performed after N forward passes of the neural net.

Batch training

Dataset: (X, y) with N samples, $\mathbf{w}$: initial weights

- for k from 1 to $N//B$:
  - for i from $B(k-1)+1$ to Bk:
    1. Compute $L(f(x_i), y_i)$
  - Calculate $\partial L/\partial w$
  - update weights:$\mathbf{w}$

N//B updates are performed after N forward passes of the neural net

B is the batchsize

- B=1: stochastic training
- B=N: batch training
- Generally B«N

Convolutional neural net

X: an image

x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	x_{16}
x_{21}	x_{22}	x_{23}	x_{24}	x_{25}	x_{26}
x_{31}	x_{32}	x_{33}	x_{34}	x_{35}	x_{36}
x_{41}	x_{42}	x_{43}	x_{44}	x_{45}	x_{46}
x_{51}	x_{52}	x_{53}	x_{54}	x_{55}	x_{56}
x_{61}	x_{62}	x_{63}	x_{64}	x_{65}	x_{66}

$$\begin{matrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{matrix}$$

$\mathbf{w}$

h: first feature

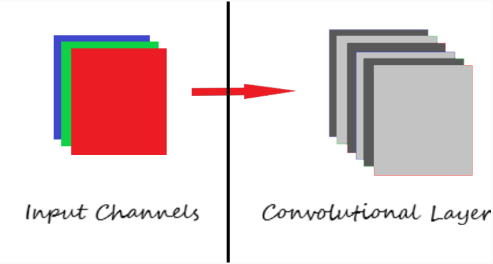
h_{11}	h_{12}	h_{13}	h_{14}
h_{21}	h_{22}	h_{23}	h_{24}
h_{31}	h_{32}	h_{33}	h_{34}
h_{41}	h_{42}	h_{43}	h_{44}

Perform a standard convolution

$$h_{i,j} = \sum_{k=1}^{3} \sum_{l=1}^{3} x_{i+k-1,j+l-1} \cdot w_{k,l}$$

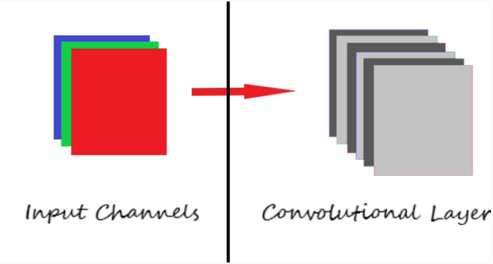
Convolutional layer

A convolutional layer is composed of p convolutions (size of layer) extracting p features from the data.



Convolutional layer

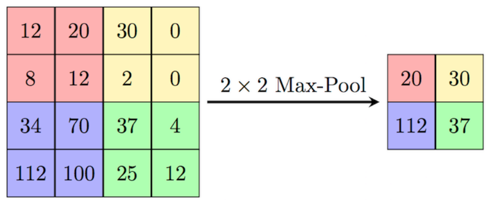
A convolutional layer is composed of p convolutions (size of layer) extracting p features from the data.



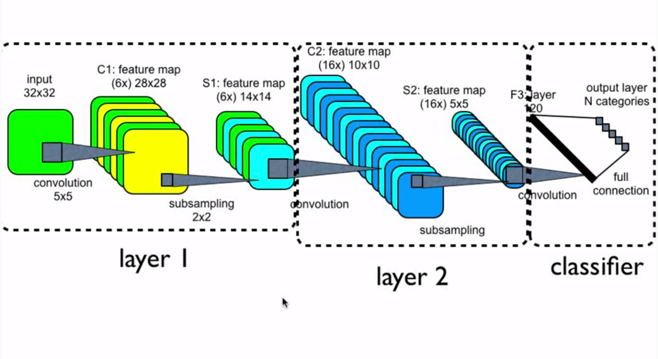
The size of the feature space is generally very big

Max-Pooling

In order to reduce the size of the feature space, a common operation is to perform a max-pooling.



A traditionnal CNN architecture



Example of AlexNet

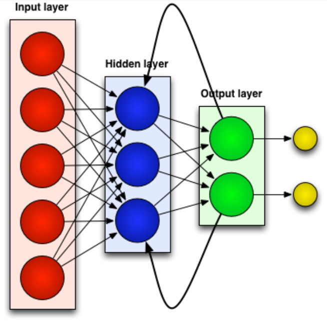
AlexNet is the first Deep architecture used on ImageNet challenge in 2012 and achieved an error of 15.3% (10% better than the previous best classifier). The paper was cited more than 34,000 times.

Alex Krizhevsky and Geoffrey E Hinton, *ImageNet Classification with Deep Convolutional Neural Networks*, Neural Information Processing Systems (2012), 1–9.

Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	227x227x3	-	-	-
1	Convolution	96	55 x 55 x 96	11x11	4	relu
	Max Pooling	96	27 x 27 x 96	3x3	2	relu
2	Convolution	256	27 x 27 x 256	5x5	1	relu
	Max Pooling	256	13 x 13 x 256	3x3	2	relu
3	Convolution	384	13 x 13 x 384	3x3	1	relu
4	Convolution	384	13 x 13 x 384	3x3	1	relu
5	Convolution	256	13 x 13 x 256	3x3	1	relu
	Max Pooling	256	6 x 6 x 256	3x3	2	relu
6	FC	-	9216	-	-	relu
7	FC	-	4096	-	-	relu
8	FC	-	4096	-	-	relu
Output	FC	-	1000	-	-	Softmax

A quick typology of few neural nets

Recurrent Neural Networks

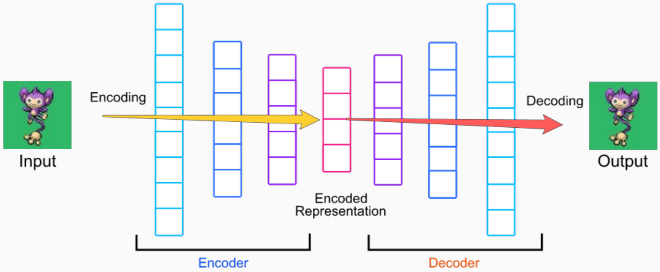


Some popular types of recurrent neural networks:

- Long short-term memory (LSTM)
- Gated Reccurent Unit (GRU)

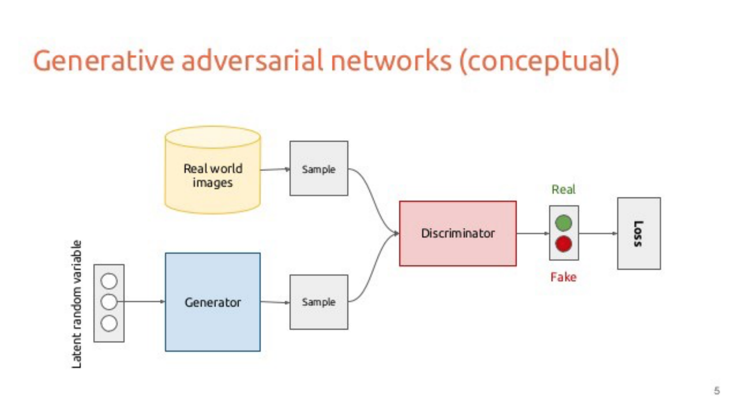
Used in machine translation and text processing

Autoencoders

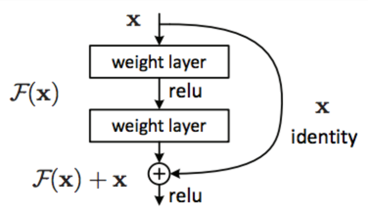


Used in image denoising, compressing, generation,...

Generative adversarial networks



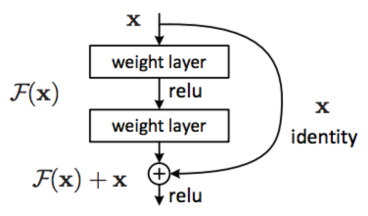
Residual Networks



x : input, y : output

$$y = x + \mathcal{F}(x)$$

Residual Networks



x : input, y : output

$$y = x + \mathcal{F}(x)$$