Machine learning and physical modelling-2

julien.brajard@nersc.no

October 2021

NERSC

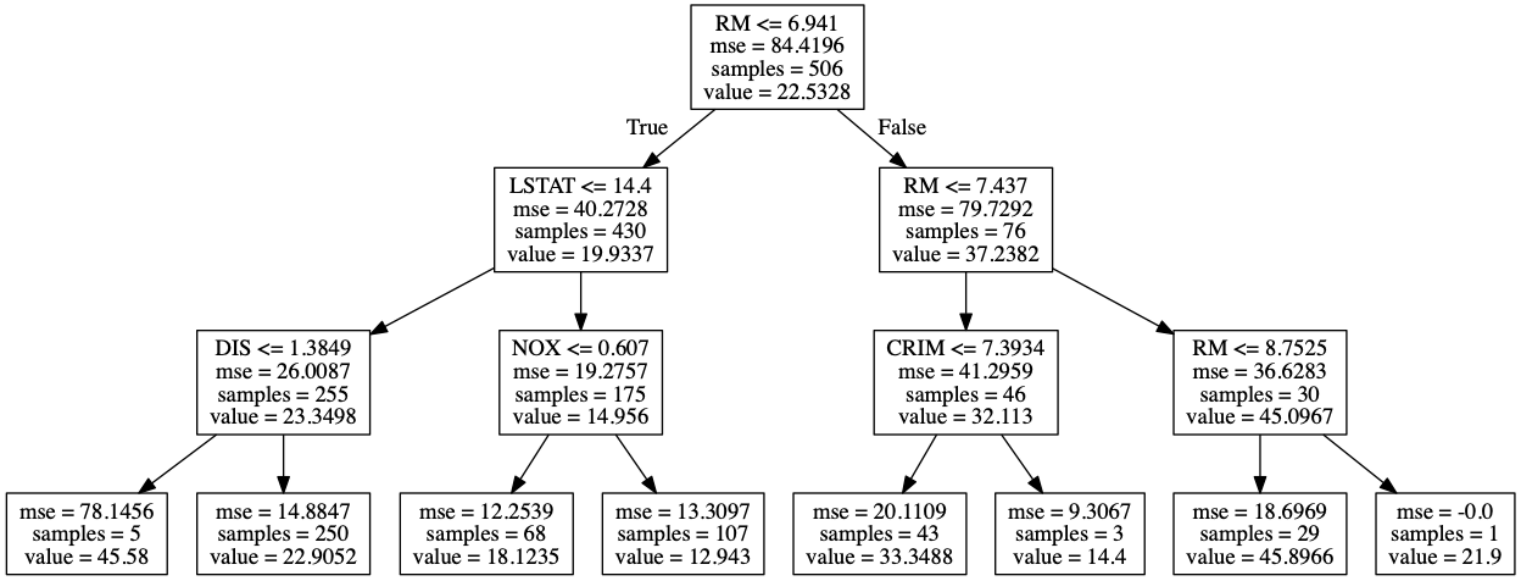
`https://github.com/brajard/MAT330`

Table of contents

A standard Machine learning model: Random Forests

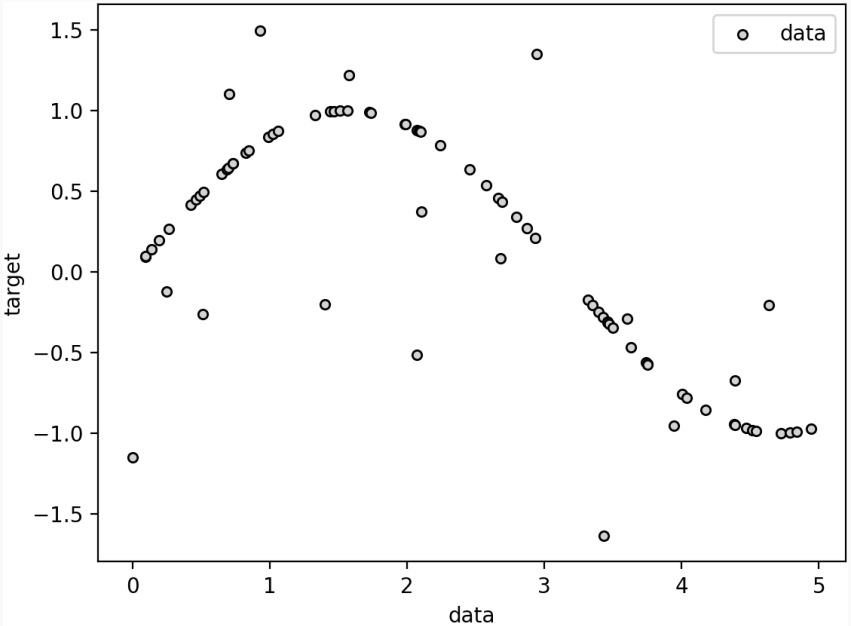
A decision tree

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

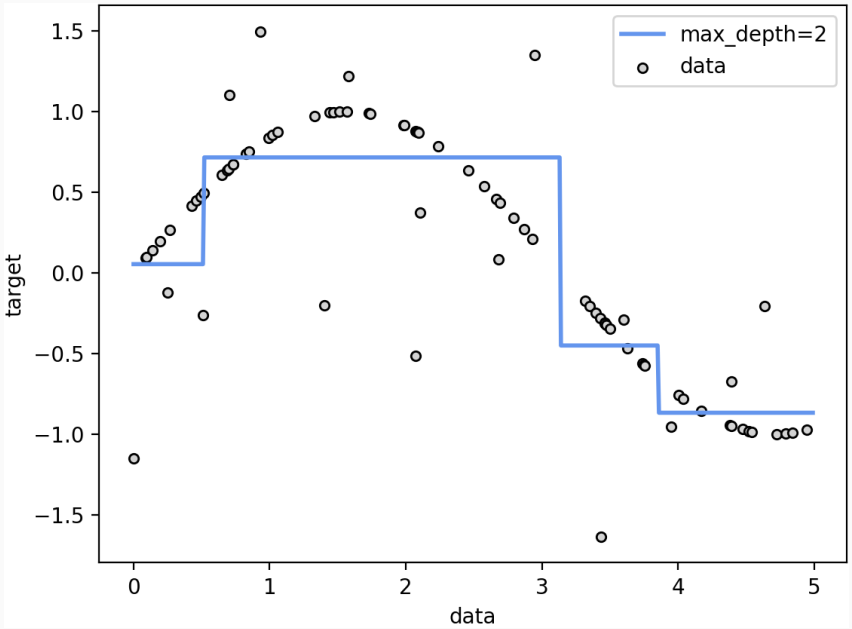


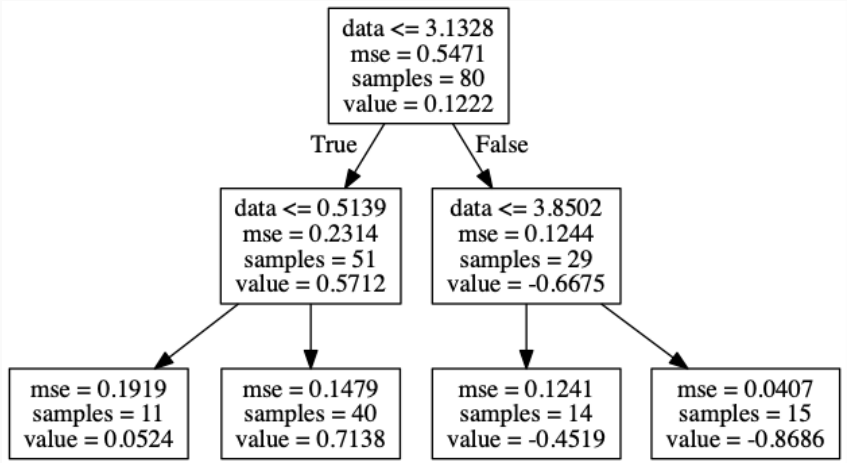
CRIM	per capita crime rate by town
NOX	nitric oxides concentration
RM	average number of rooms per dwelling
DIS	distance to employment centres
LSTAT	lower status of the population

Uni-variate example

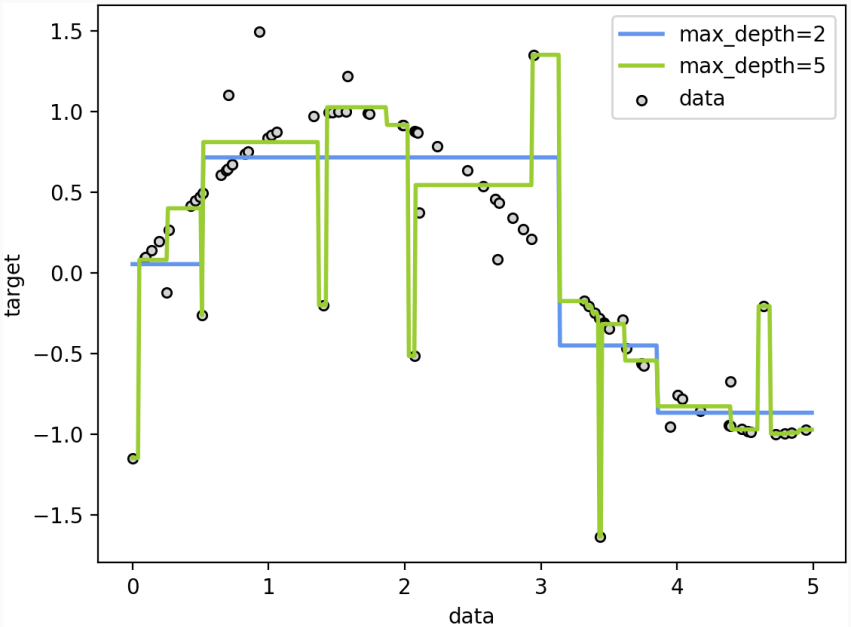


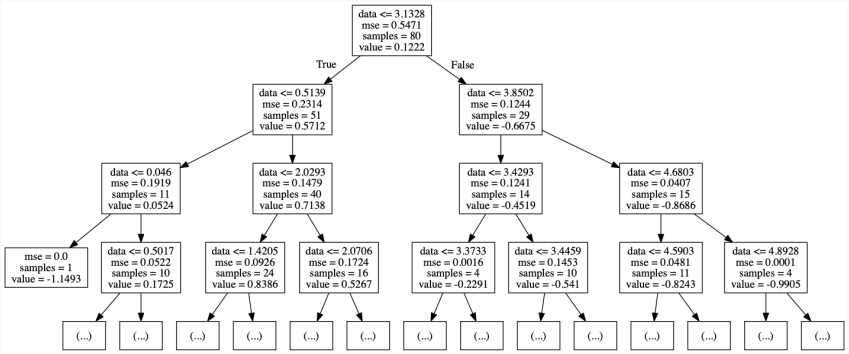
Uni-variate example





Uni-variate example





From tree to forest

Disadvantages of regression tree:

- Can overfit the data

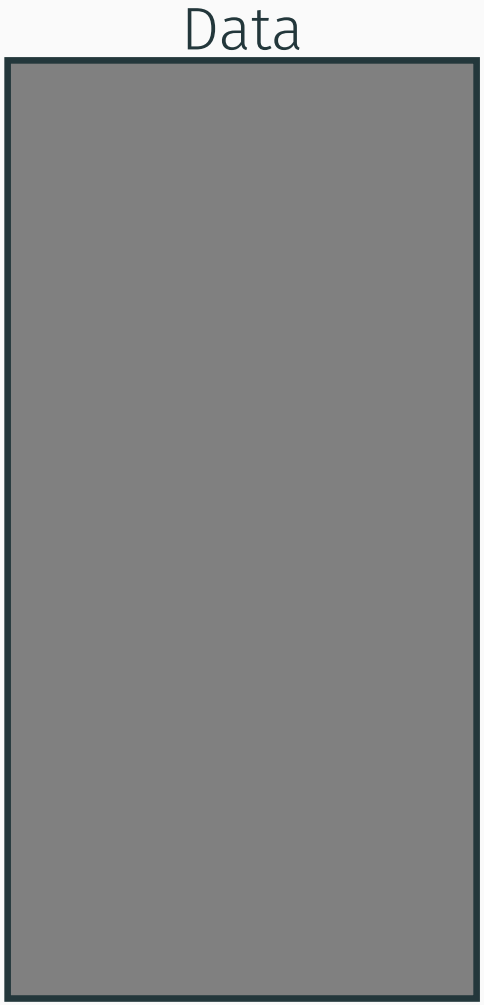
From tree to forest

Disadvantages of regression tree:

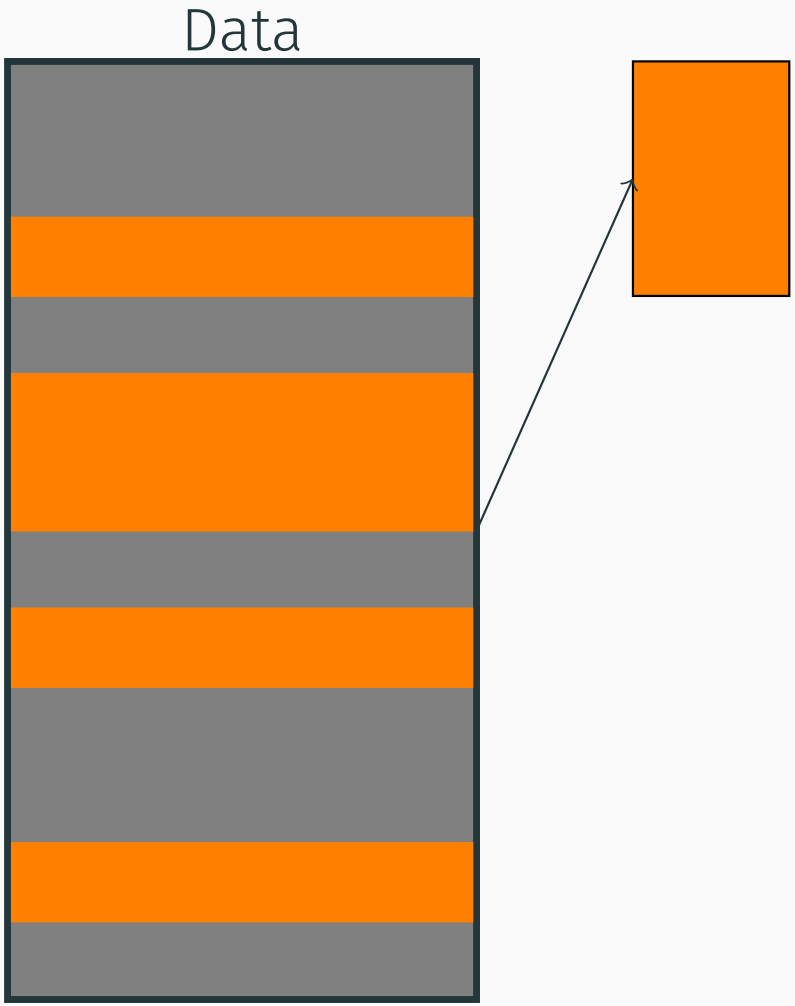
- Can overfit the data

One extension of Regression Tree: Random Forest

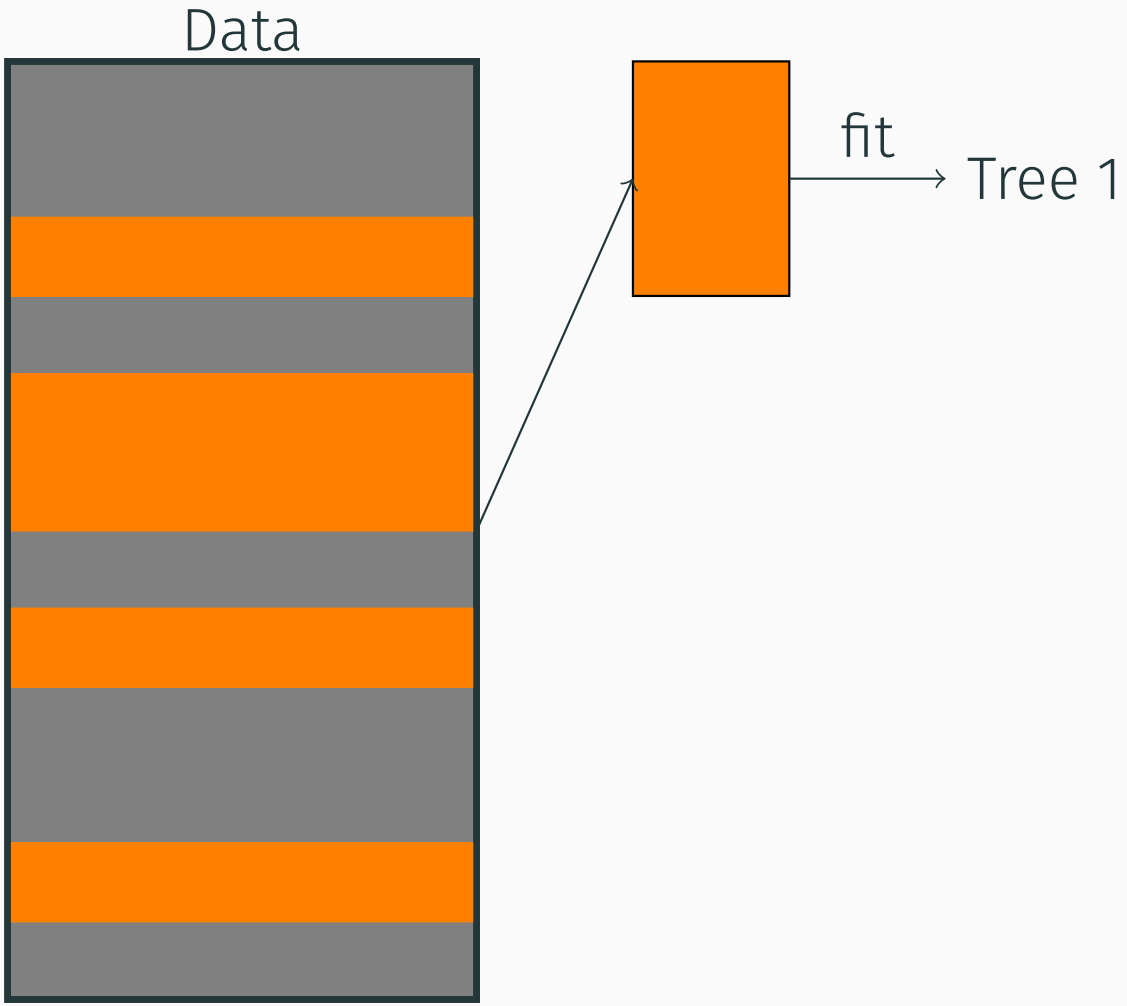
The (over simplified) principle of Random Forest



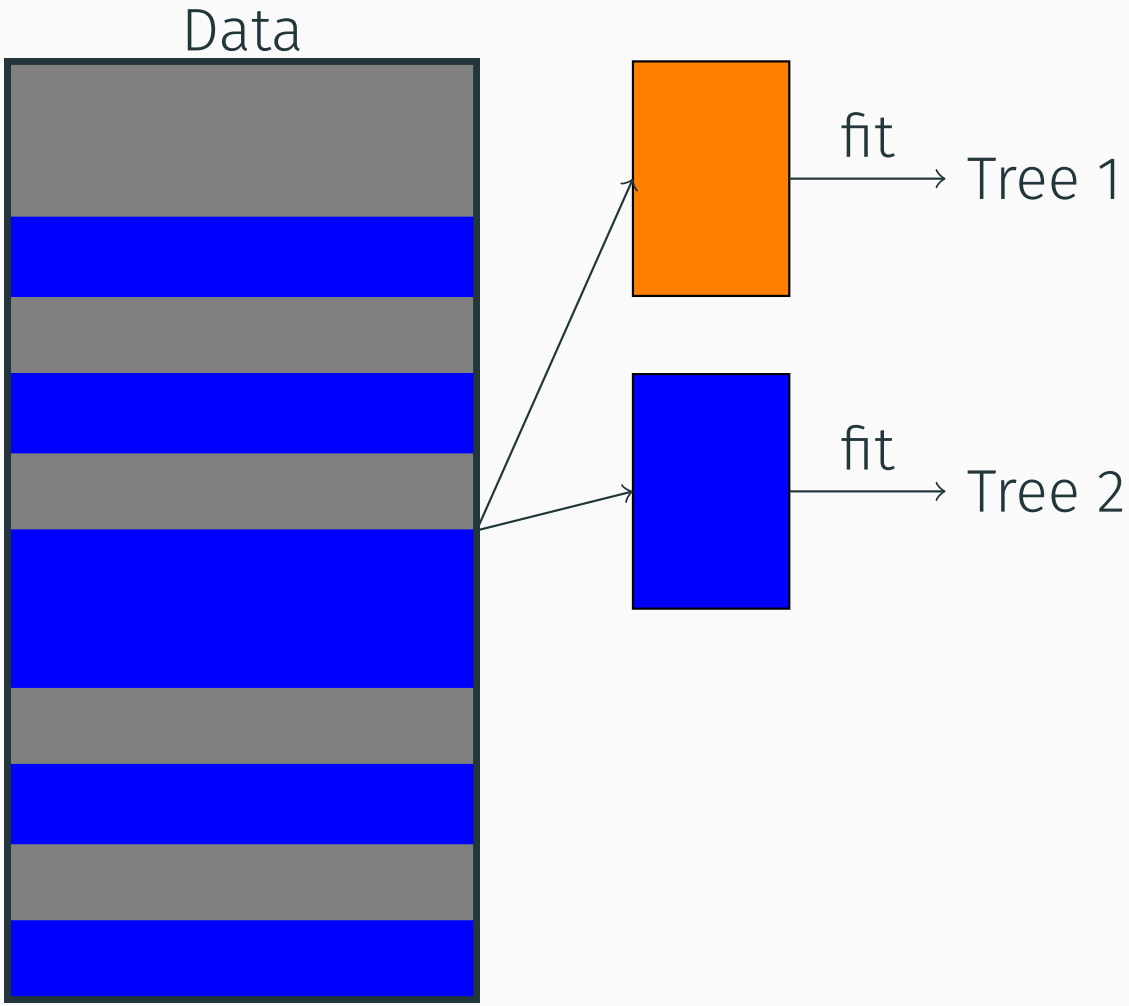
The (over simplified) principle of Random Forest



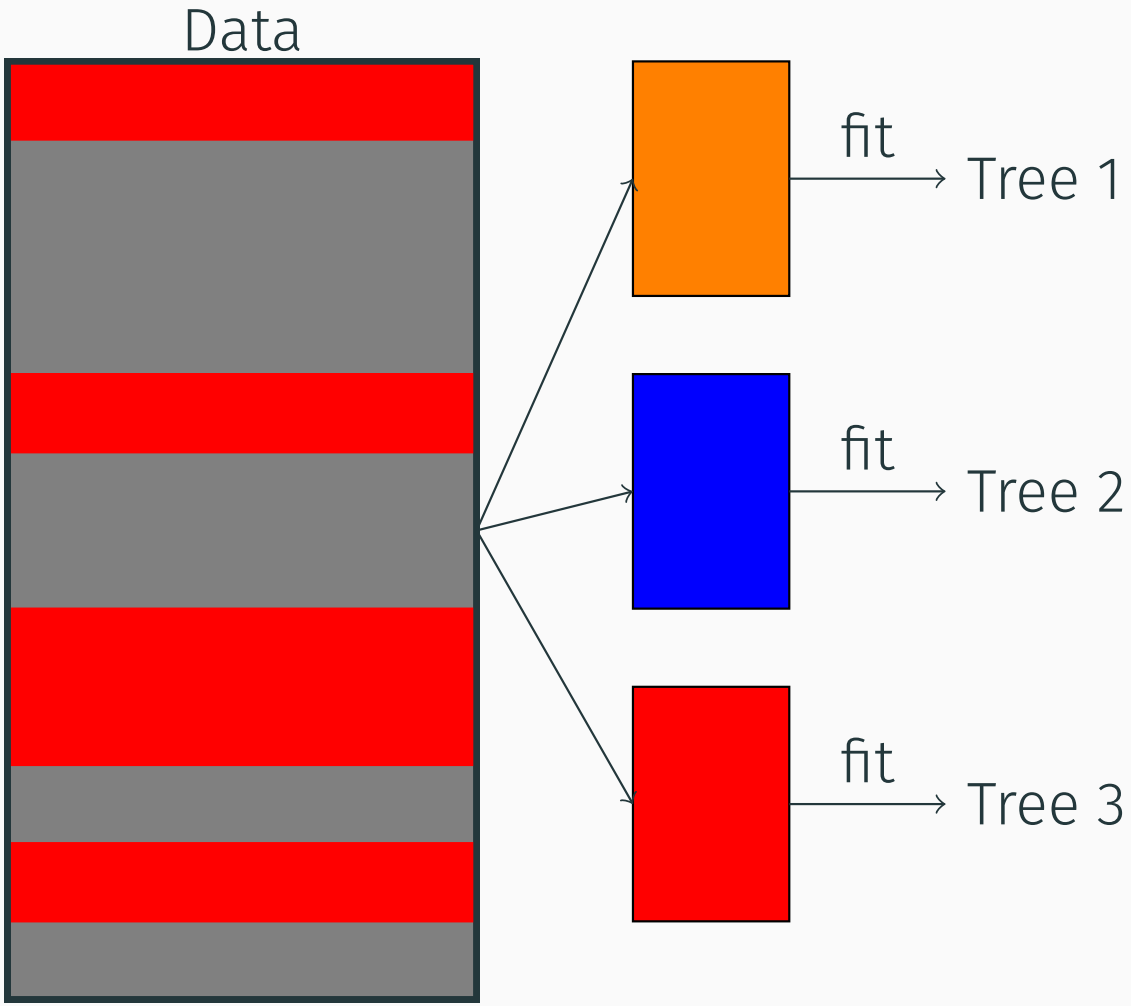
The (over simplified) principle of Random Forest



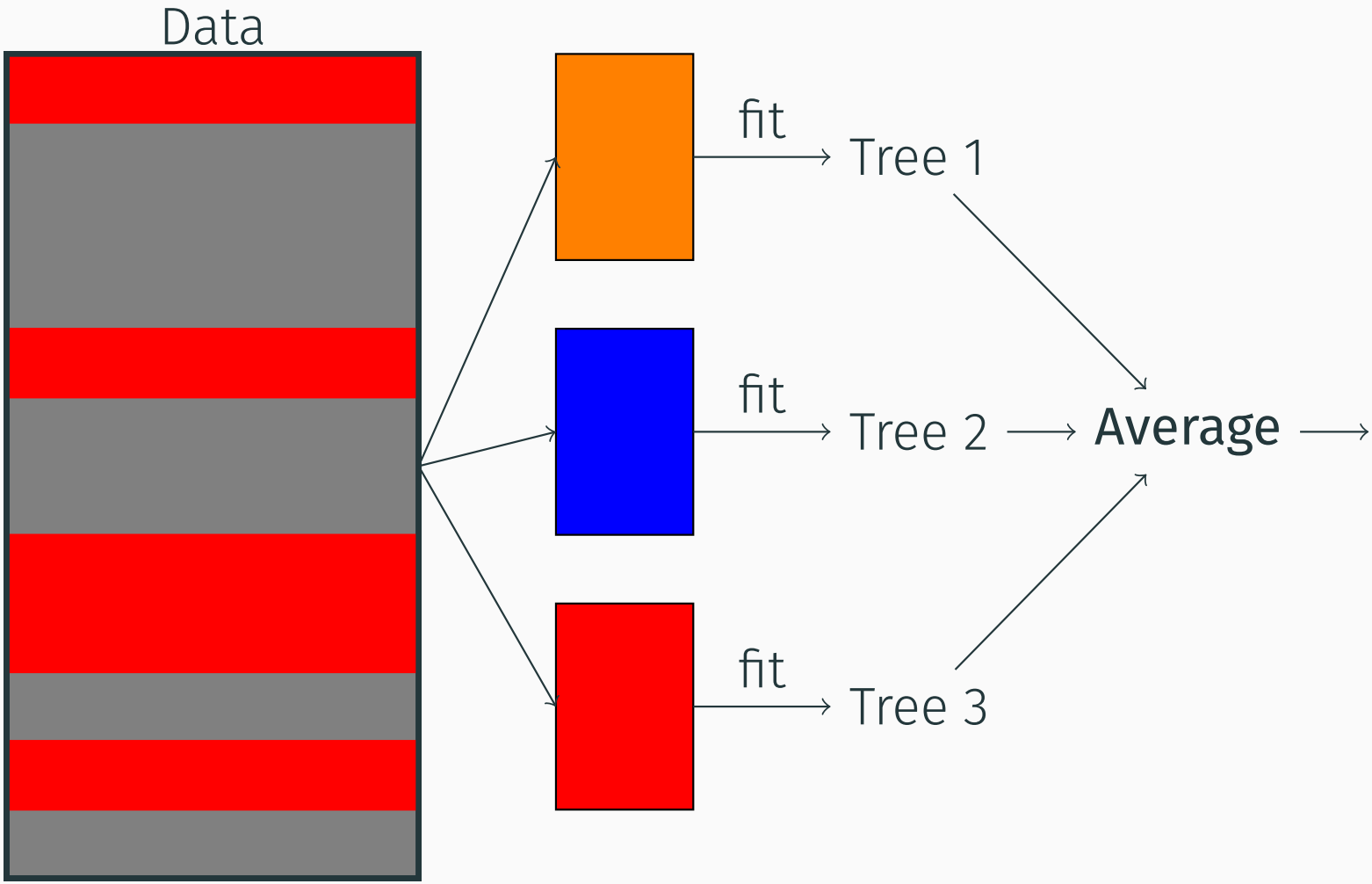
The (over simplified) principle of Random Forest



The (over simplified) principle of Random Forest

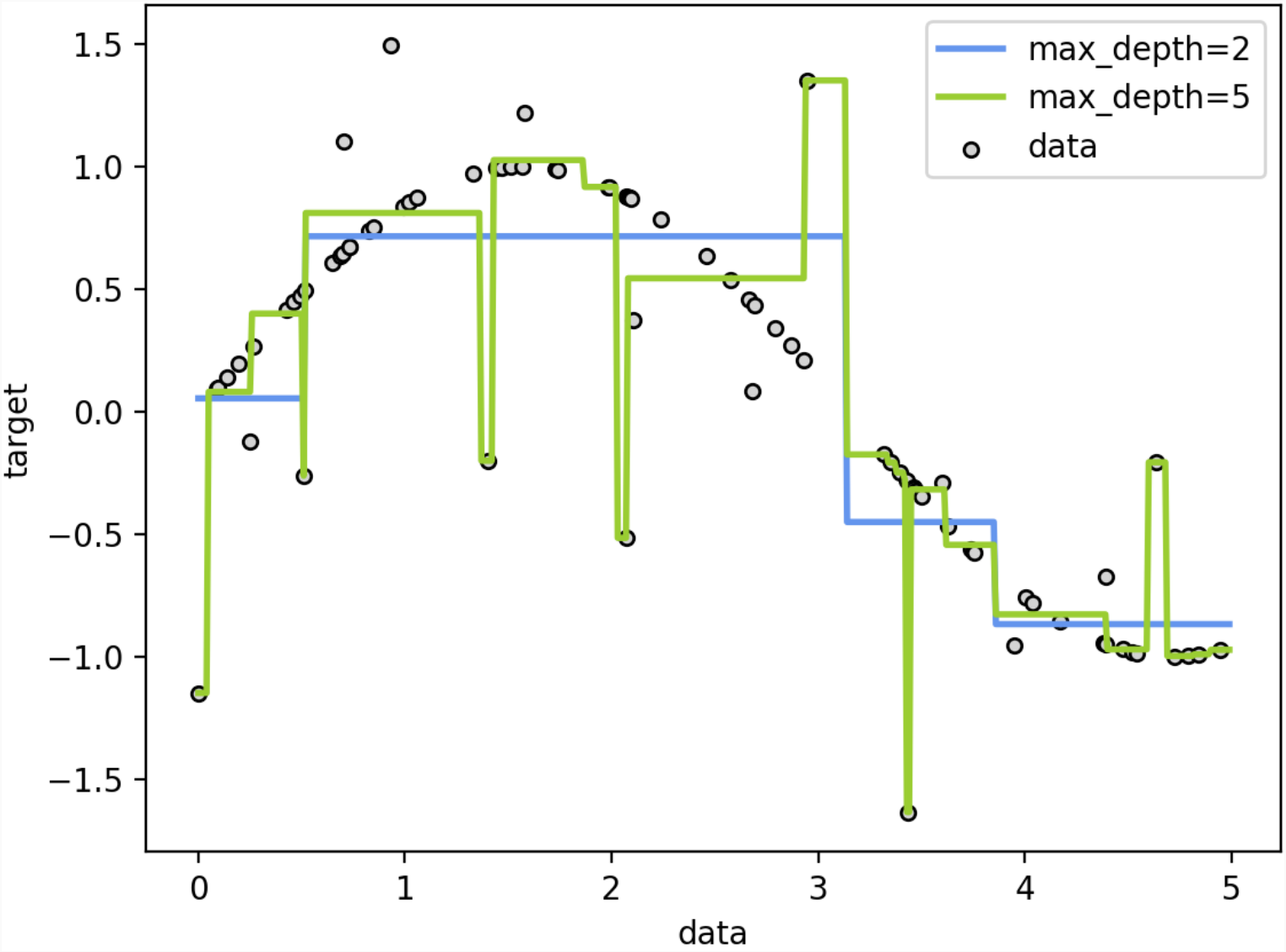


The (over simplified) principle of Random Forest



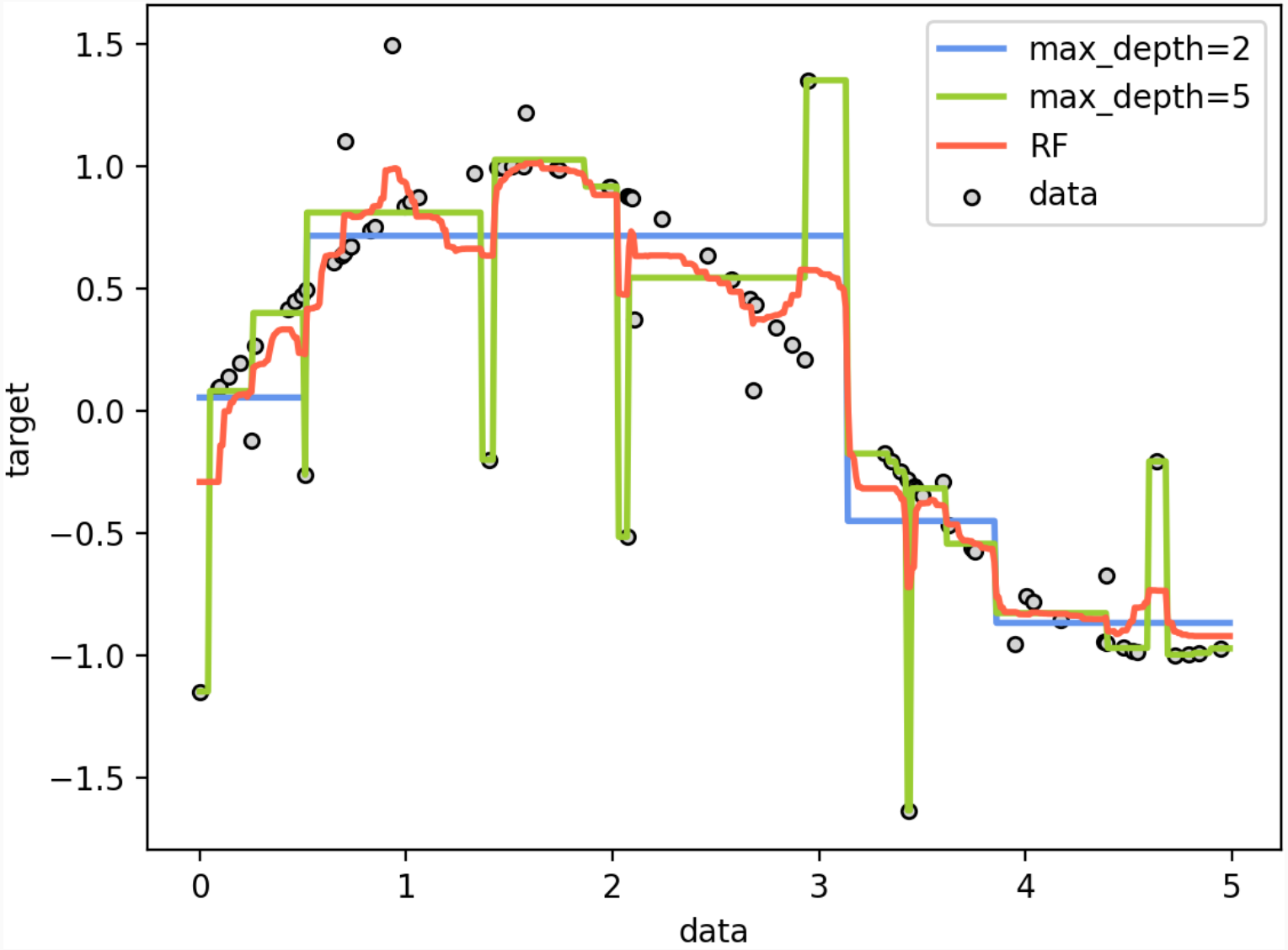
Results on the univariate experiment

Prediction of Randoms trees



Results on the univariate experiment

Prediction of a Random Forest



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)

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

```
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 hyperparameters

- Parameters that are not optimized during the training are called hyperparameters.

Determination of the hyperparameters

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

Determination of the hyperparameters

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

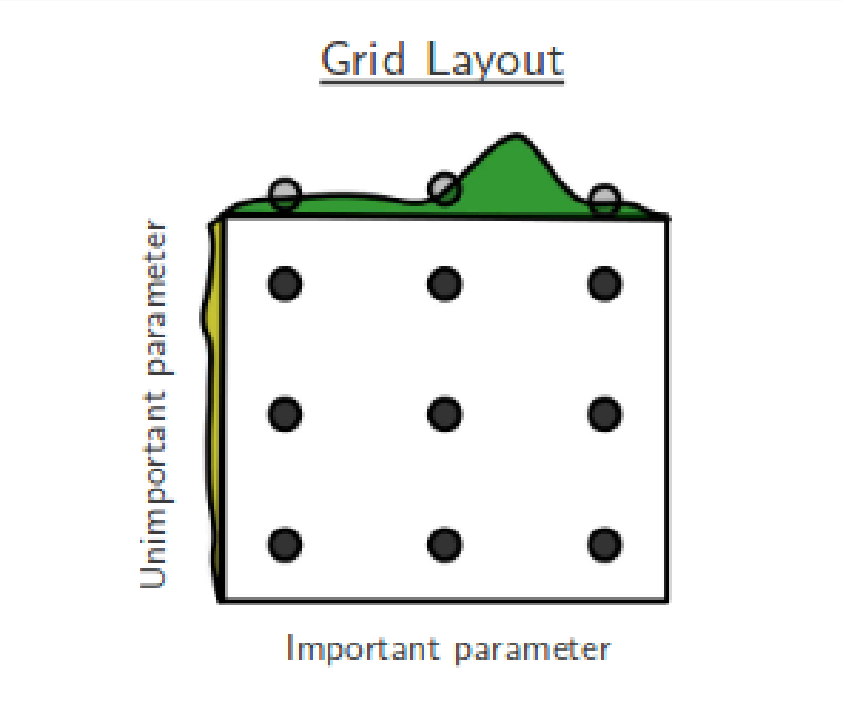
Determination of the hyperparameters

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

https://xkcd.com/1838/

Stir the pile: The gridsearch

1. Specify a list of hyperparameters to be tested.
2. For each of the parameters, specify a set of values to test
3. Train a model for each of the possible combinations of hyperparameters
4. Retain the best model (using, e.g., cross-validation)



https://medium.com/@senapati.dipak97/grid-search-vs-random-search-d34c92946318

Remarks on the gridsearch procedure

- It make an exhaustive search of the hyperparameters

Remarks on the gridsearch procedure

- It make an exhaustive search of the hyperparameters
- The procedure is easy to parallelized.

Remarks on the gridsearch procedure

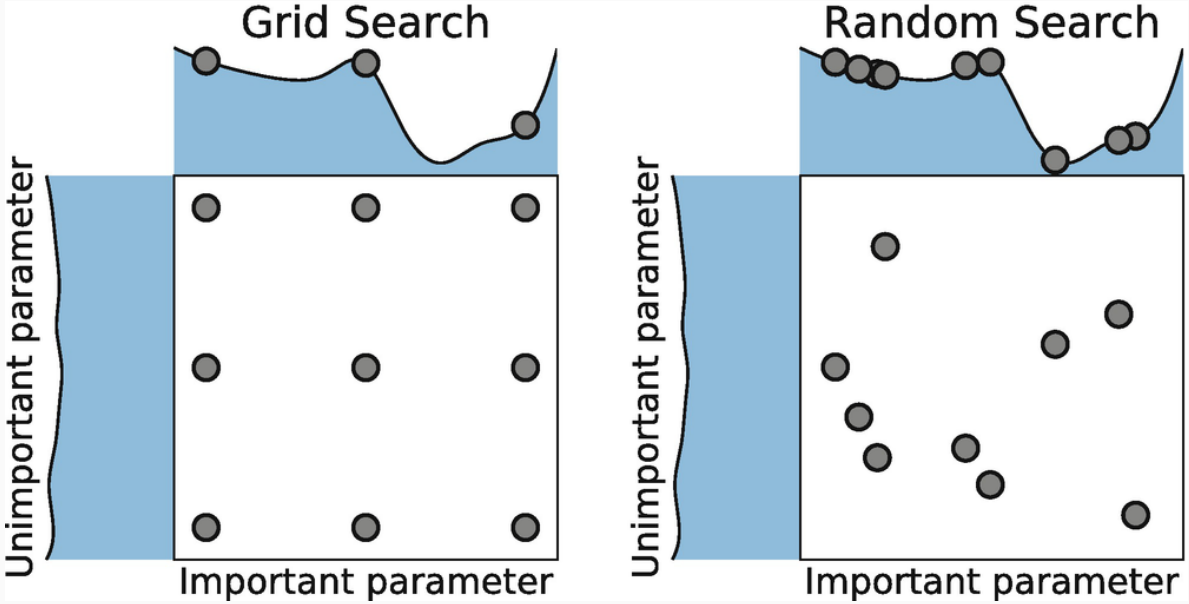
- It make an exhaustive search of the hyperparameters
- The procedure is easy to parallelized.
- it is not naturally adapted for quantitative hyperparameters.

Remarks on the gridsearch procedure

- It make an exhaustive search of the hyperparameters
- The procedure is easy to parallelized.
- it is not naturally adapted for quantitative hyperparameters.
- it can become very costly. (e.g. 8 hyperparameters with 8 values each to test = $8^8 = 16,777,216$ trainings.

Random search

1. Specify a list of hyperparameters to be tested.
2. For each of the parameters, specify a set of values to test or a law to draw a random value.
3. Draw n combinations of the hyperparameters.
4. Train a model for each of the combinations.
5. Retain the best model (using, e.g., cross-validation)



Remarks on the random search procedure

- It does not make an exhaustive search of the hyperparameters
- The procedure is easy to parallelized.
- The cost is predictable (number of draw).

Remarks on the random search procedure

- It does not make an exhaustive search of the hyperparameters
- The procedure is easy to parallelized.
- The cost is predictable (number of draw).

Both gridsearch and random search are implemented and easy to use in scikit-learn.

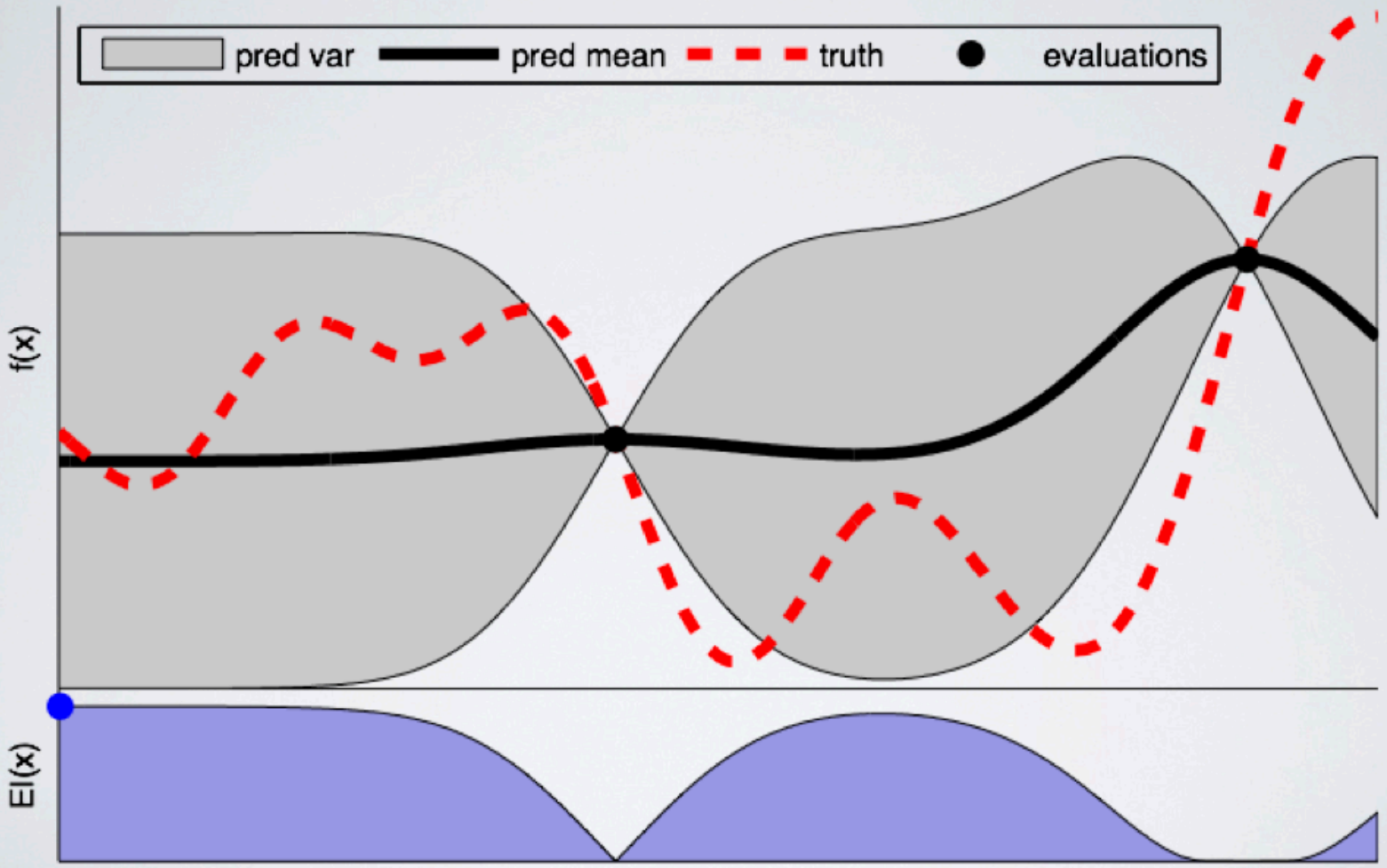
Bayesian search

A more ”advance” method it the Bayesian optimization.

The principle is to search hyperparameters where it is more likely to improve the model (based on previous attempts)

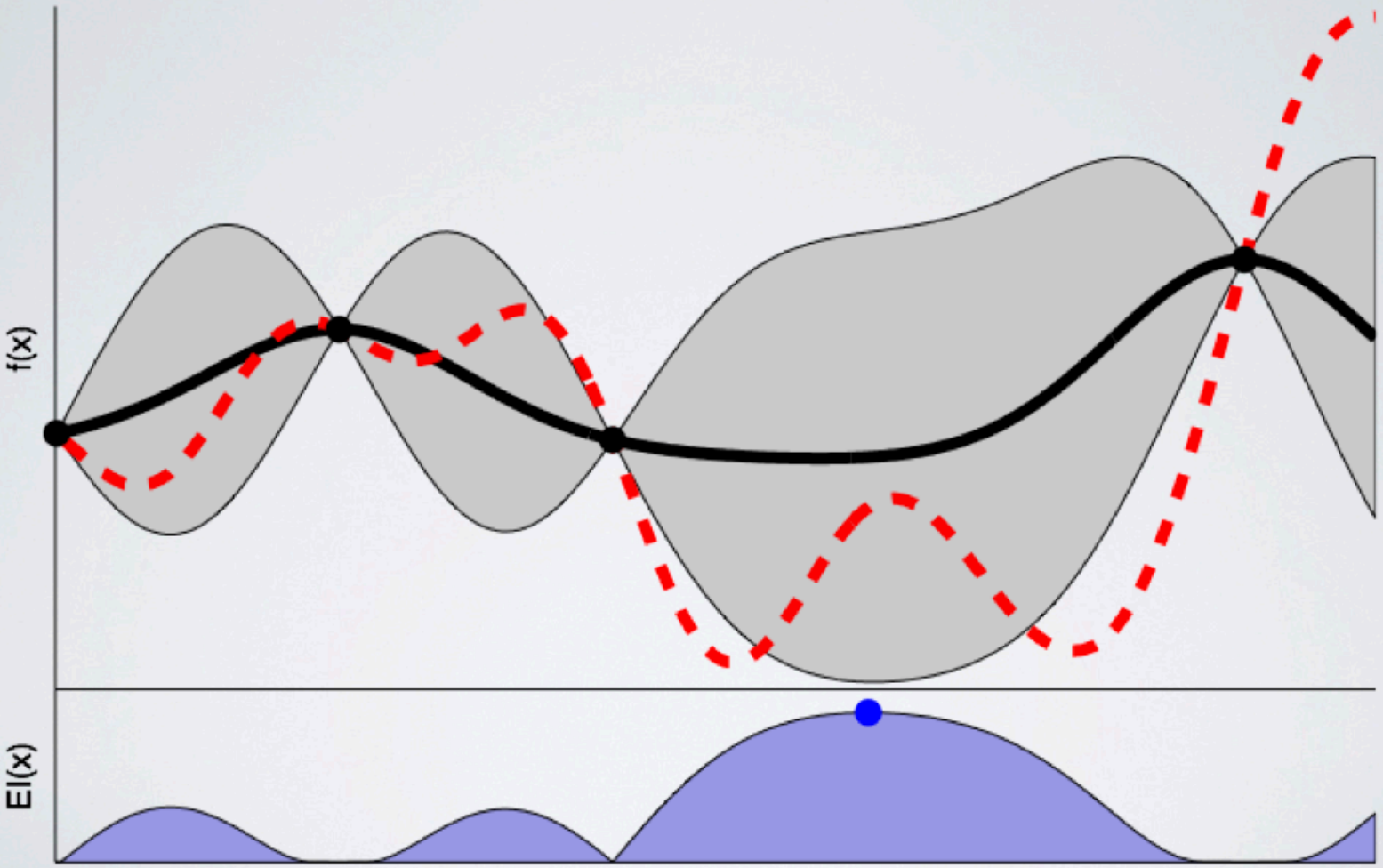
Illustration taken from Ryan P. Adams: ”A Tutorial on Bayesian Optimization for Machine Learning”

Illustrating Bayesian Optimization



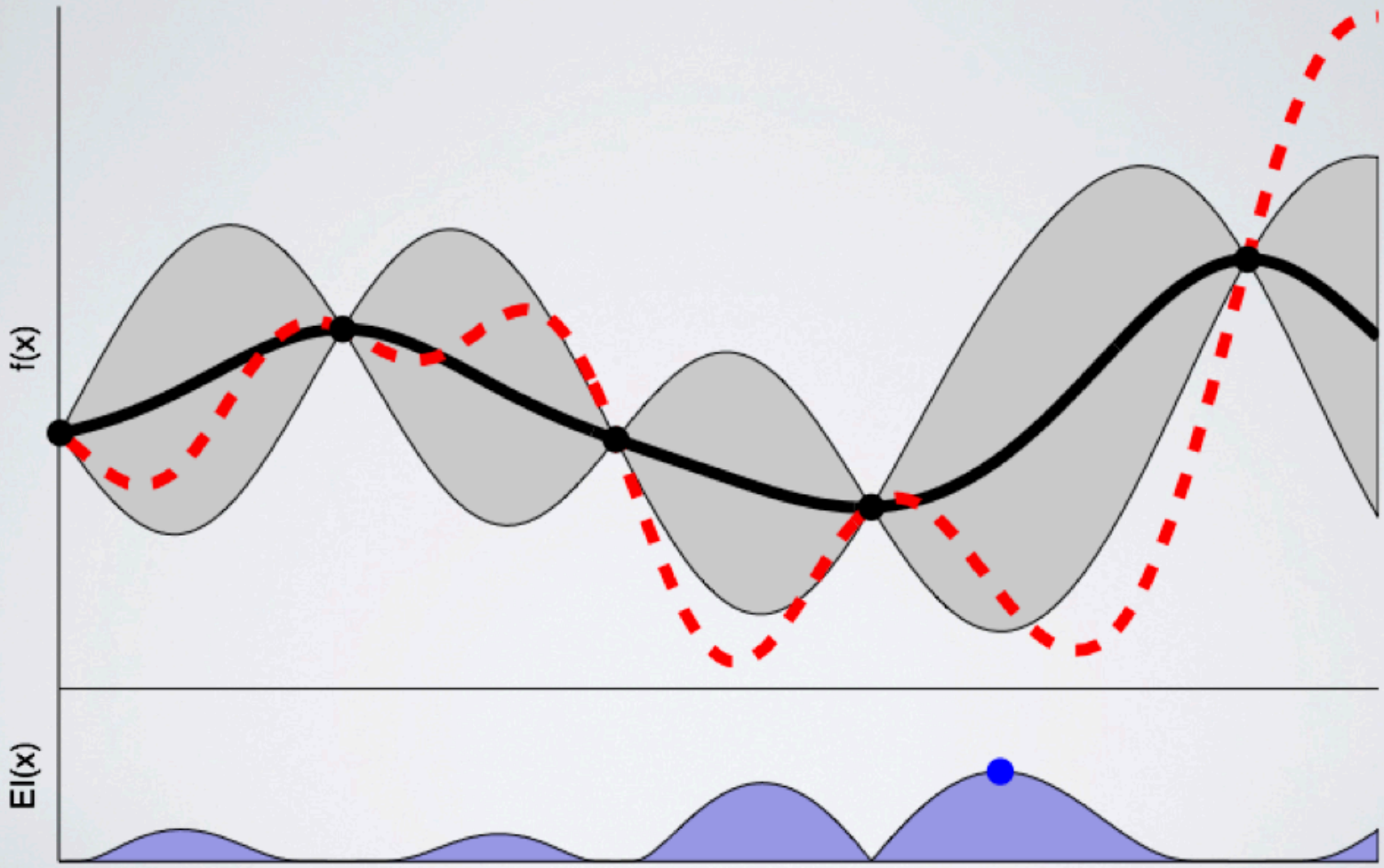
source: R.P. Adams

Illustrating Bayesian Optimization



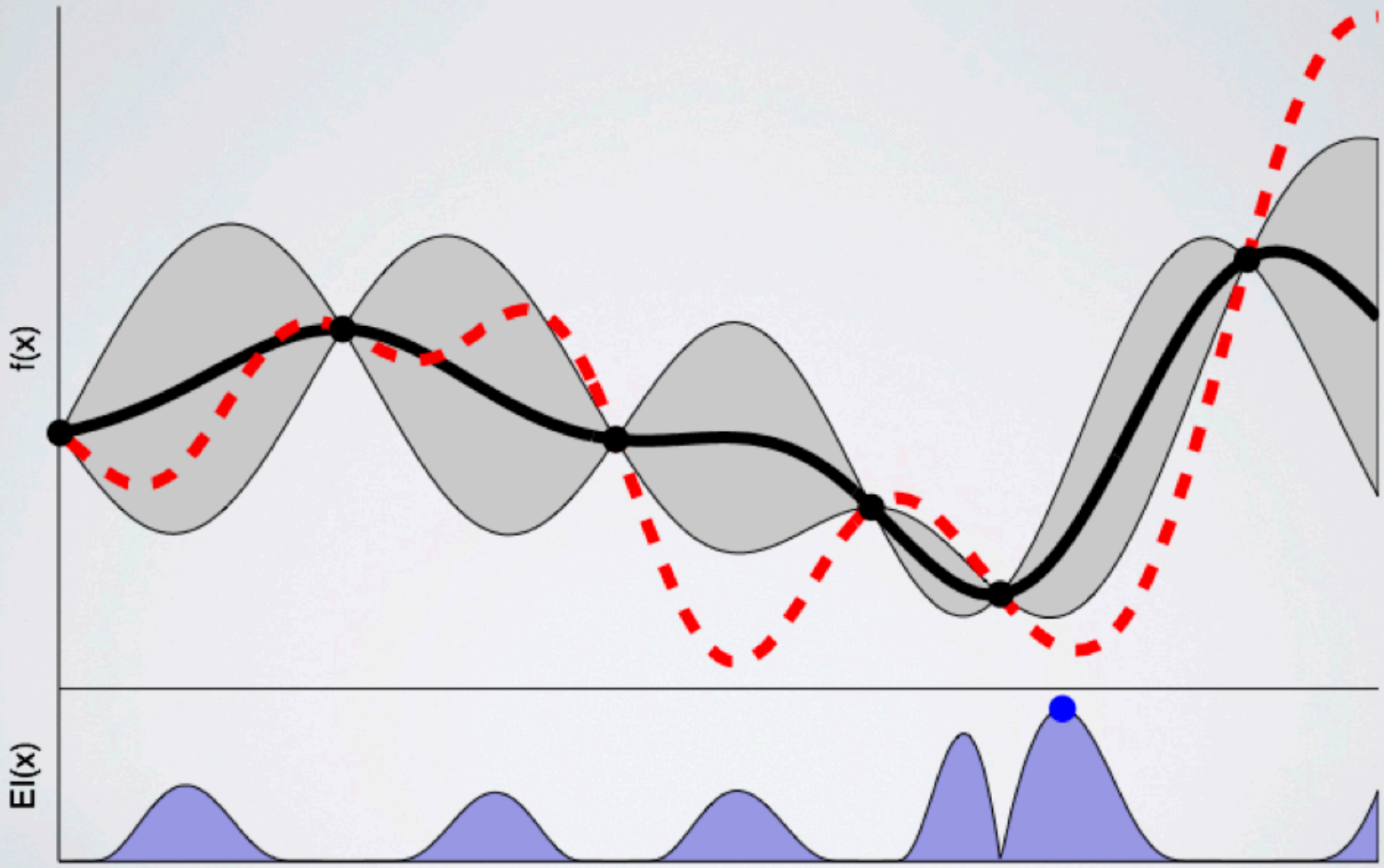
source: R.P. Adams

Illustrating Bayesian Optimization



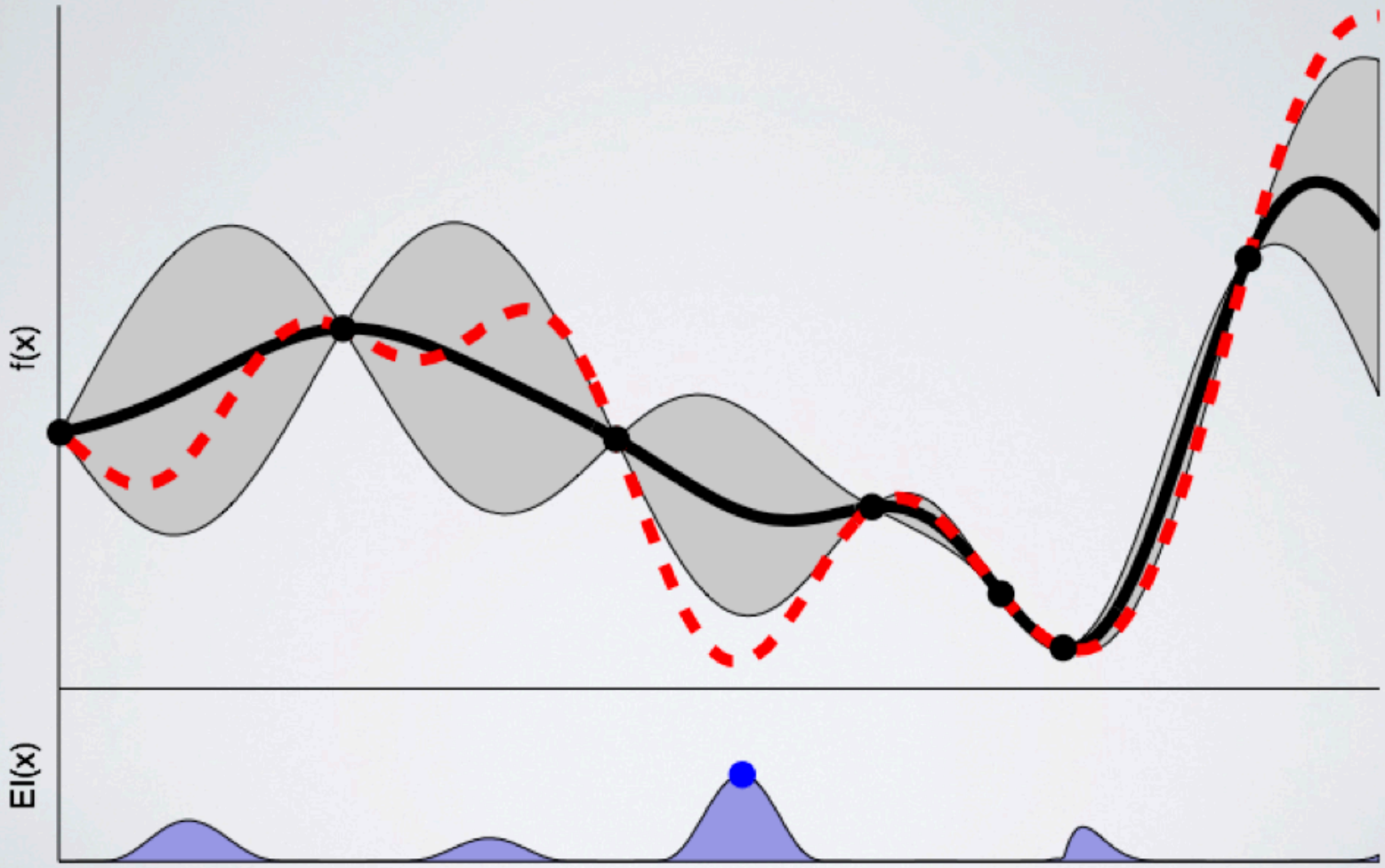
source: R.P. Adams

Illustrating Bayesian Optimization



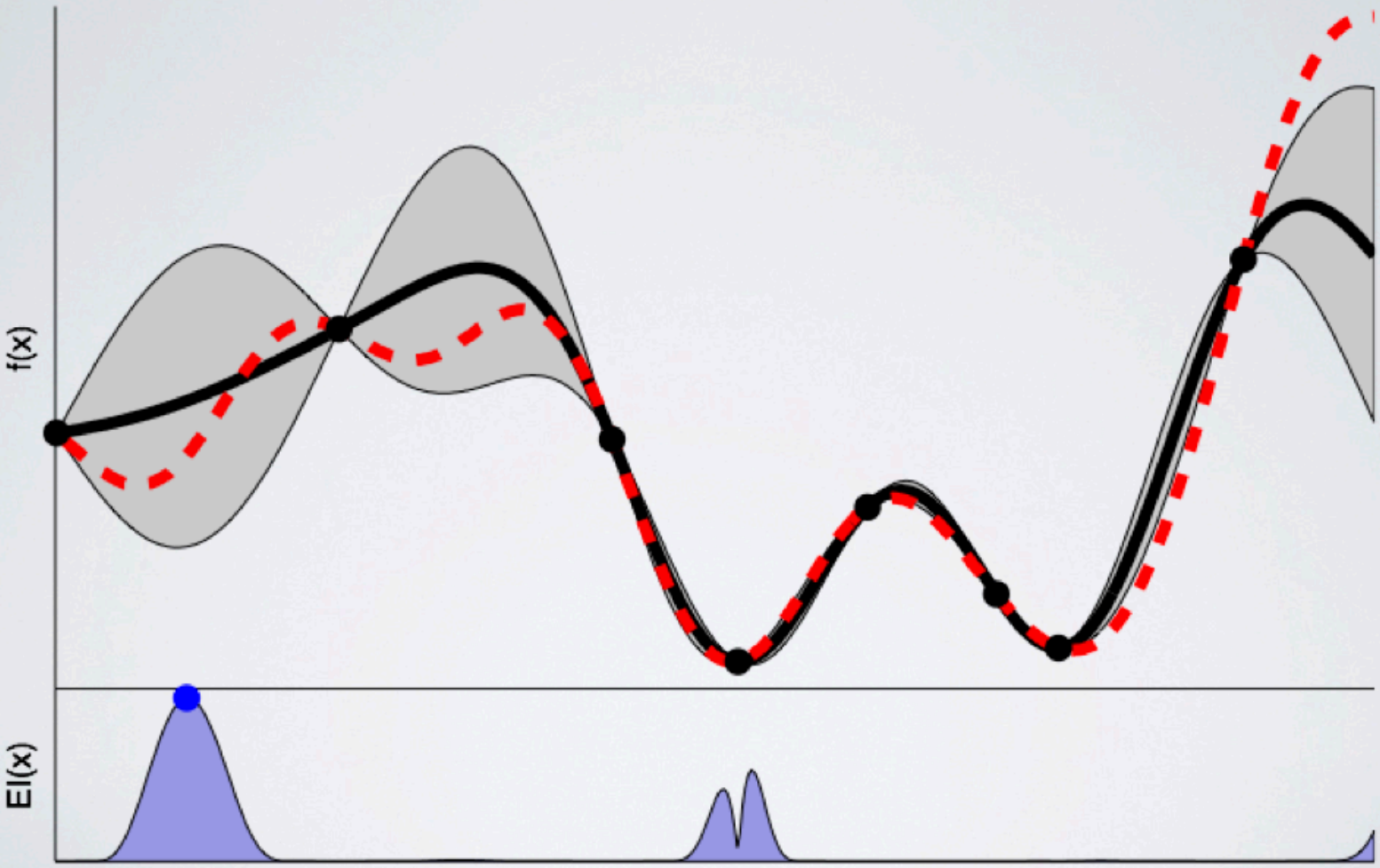
source: R.P. Adams

Illustrating Bayesian Optimization



source: R.P. Adams

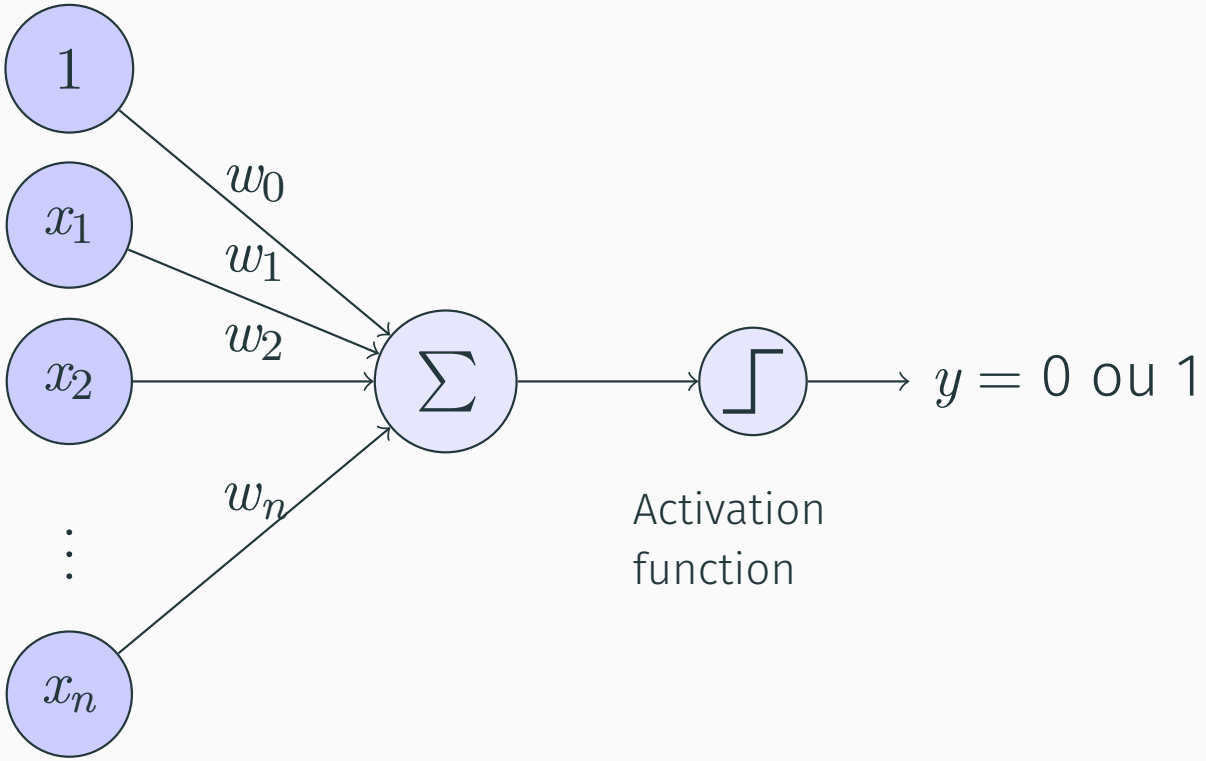
Illustrating Bayesian Optimization



source: R.P. Adams

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

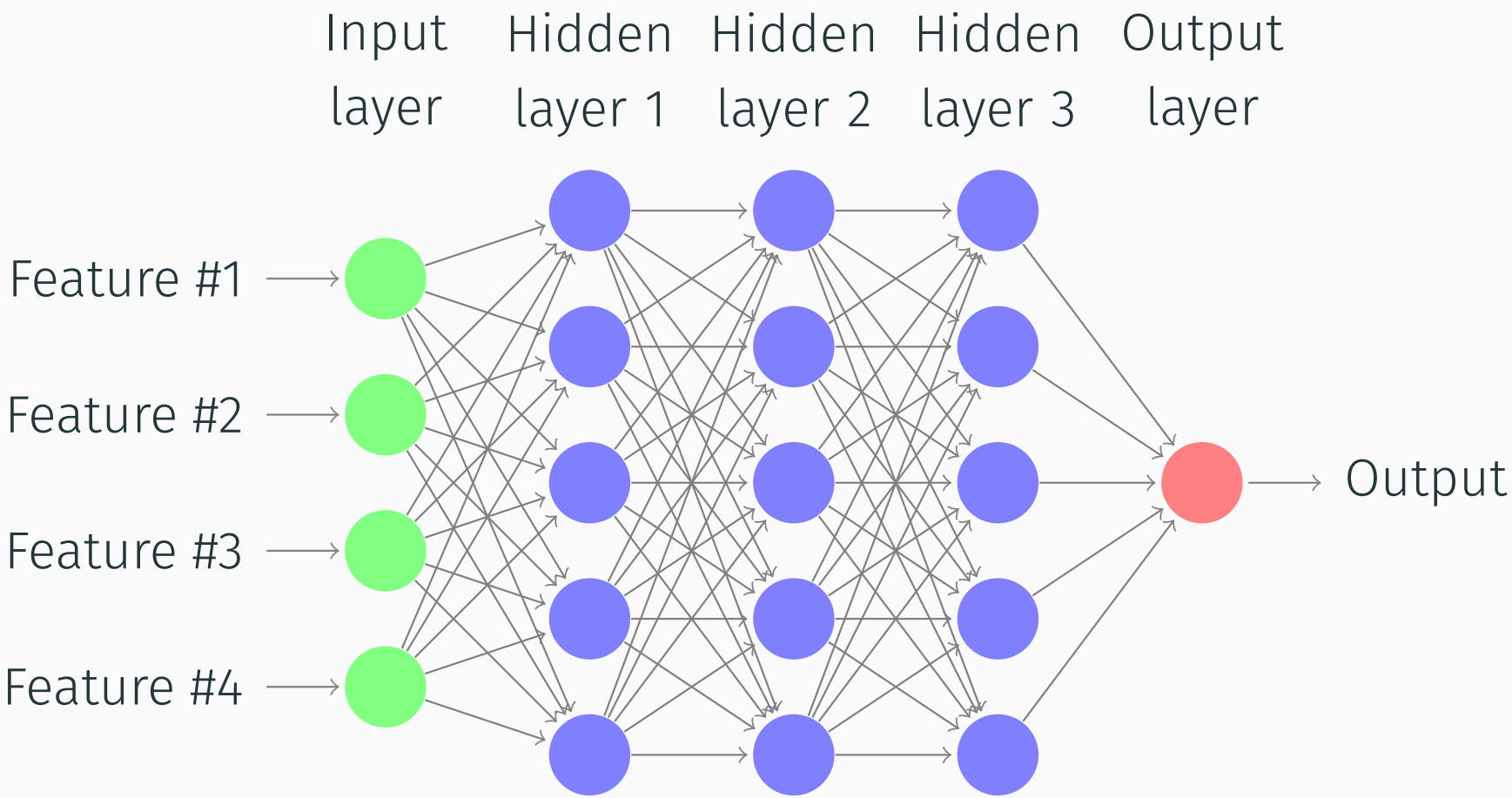
- Inputs x_i are the different features of the data
- Weight w_i are the parameters of the model to optimize
- If the activation function is identity, it is equivalent to a linear regression

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

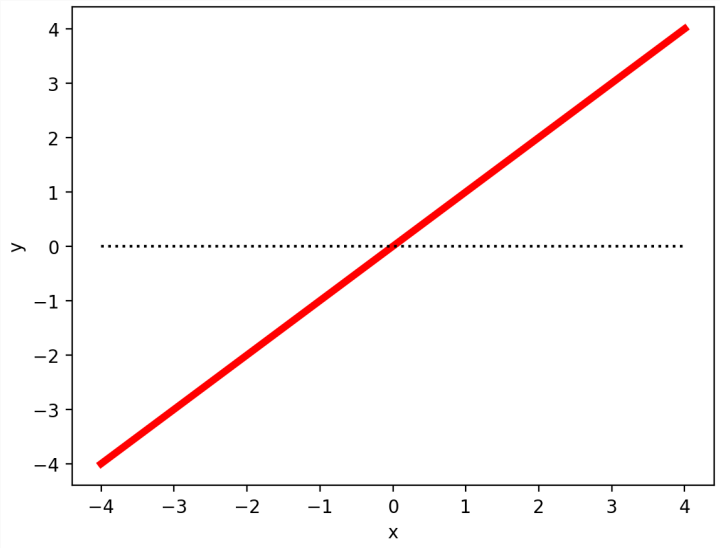
More complexe models are build by combining several perceptrons

Multi-layer perceptron (Densely connected layers)

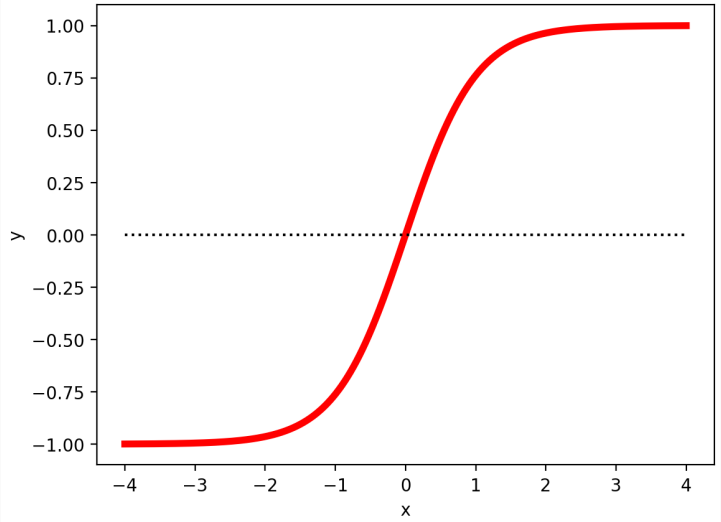


Most usual activation functions

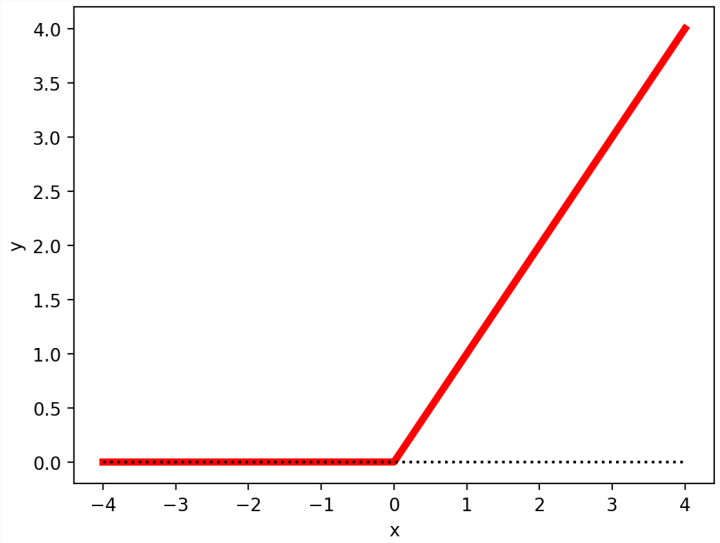
Linear



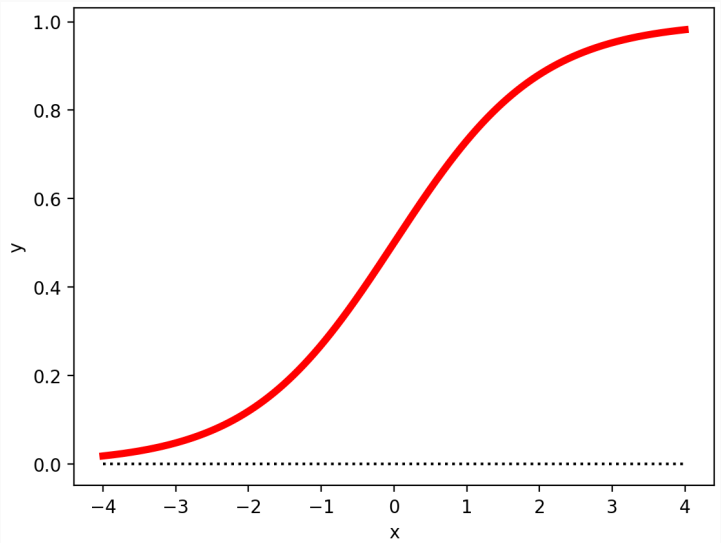
Hyperbolic tangent



ReLU

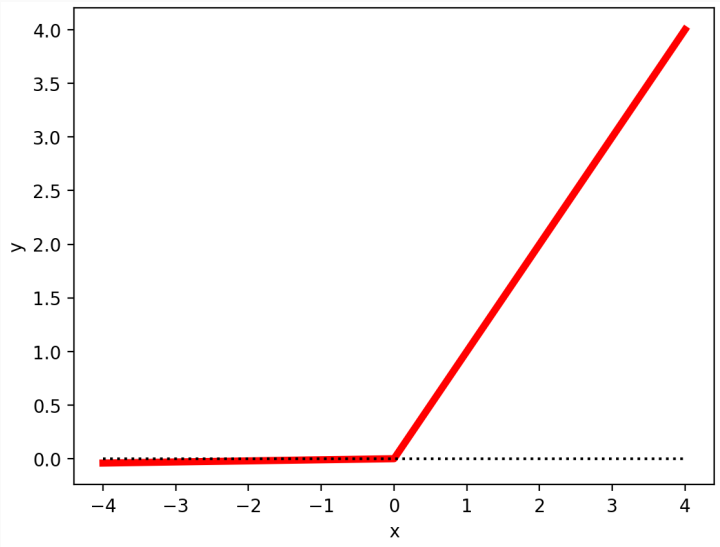


Sigmoid

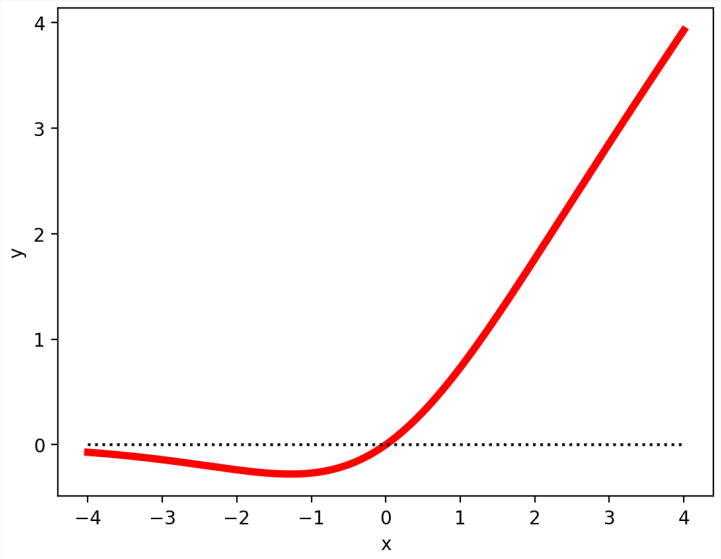


New fancy activation functions

Leaky-ReLU



Swish



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

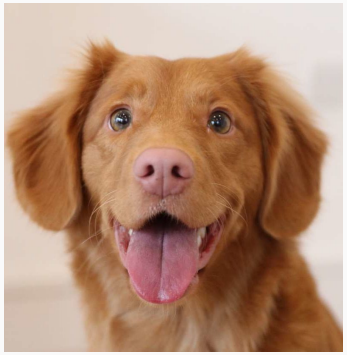
$$L(p, y) = -\sum_i \sum_j y_{i,j}. \log p_{i,j}$$

Classification loss (binary case)

Objective: binary classification (the model determine if the input feature is in a class or not)

Exemple: Try to know if an image is a cat or not.

Dataset:

x (feature)				...
y (target)	Cat	Dog	Cat	...

How to proceed?

1. Encode the targets (1 for cat, 0 otherwise)

y (target)	Cat	Dog	Cat	...
y (encoded)	1	0	1	...

How to proceed?

1. Encode the targets (1 for cat, 0 otherwise)

y (target)	Cat	Dog	Cat	...
y (encoded)	1	0	1	...

2. Design a model with one output and use the sigmoid as output activation function of your model $f(x)$, so $0 < f(x) < 1$.
   Rule: if $f(x) > \frac{1}{2}$, classify as "Cat".
   f(x) is interpreted as the probability of the image x to be a cat.

How to proceed?

1. Encode the targets (1 for cat, 0 otherwise)

y (target)	Cat	Dog	Cat	...
y (encoded)	1	0	1	...

2. Design a model with one output and use the sigmoid as output activation function of your model $f(x)$, so $0 < f(x) < 1$.
   Rule: if $f(x) > \frac{1}{2}$, classify as "Cat".
   f(x) is interpreted as the probability of the image x to be a cat.
3. Loss function to minimize is binary cross entropy:

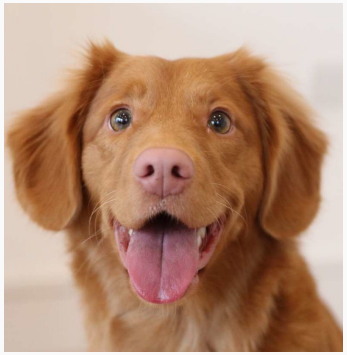
$$L = -\sum y_i \cdot \log(f(x_i)) + (1 - y_i) \cdot \log(1 - f(x_i))$$

Classification loss (multiclass)

Objective: classification (the model determine in which class the input feature (more than 2 classes))

Exemple: Try to know if an image is a cat, a dog or a duck.

Dataset:

x (feature)				...
y (target)	Cat	Dog	Duck	...

How to proceed?

1. Encode the targets using one hot encoding

y (target)	Cat	Dog	Cat	...
y (encoded)	$(1, 0, 0)$	$(0, 1, 0)$	$(0, 0, 1)$	...

How to proceed?

1. Encode the targets using one hot encoding

y (target)	Cat	Dog	Cat	...
y (encoded)	$(1,0,0)$	$(0,1,0)$	$(0,0,1)$	...

2. Design a model with N outputs (N being the numner of modalities) and use the soft-max as output activation function

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

Rule the class is attributed to the argument of the maximum of $\mathbf{p}$. Ex $\mathbf{p} = (0.1, 0, 7, 0, 2)$ is classified as "dog". p_j is interpreted as the probability of the image x to belong to the class j

How to proceed?

1. Encode the targets using one hot encoding

y (target)	Cat	Dog	Cat	...
y (encoded)	$(1,0,0)$	$(0,1,0)$	$(0,0,1)$	...

2. Design a model with N outputs (N being the numner of modalities) and use the soft-max as output activation function

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

Rule the class is attributed to the argument of the maximum of $\mathbf{p}$. Ex $\mathbf{p} = (0.1, 0, 7, 0, 2)$ is classified as "dog". p_j is interpreted as the probability of the image x to belong to the class j

3. Loss function to minimize is negative cross entropy

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

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

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

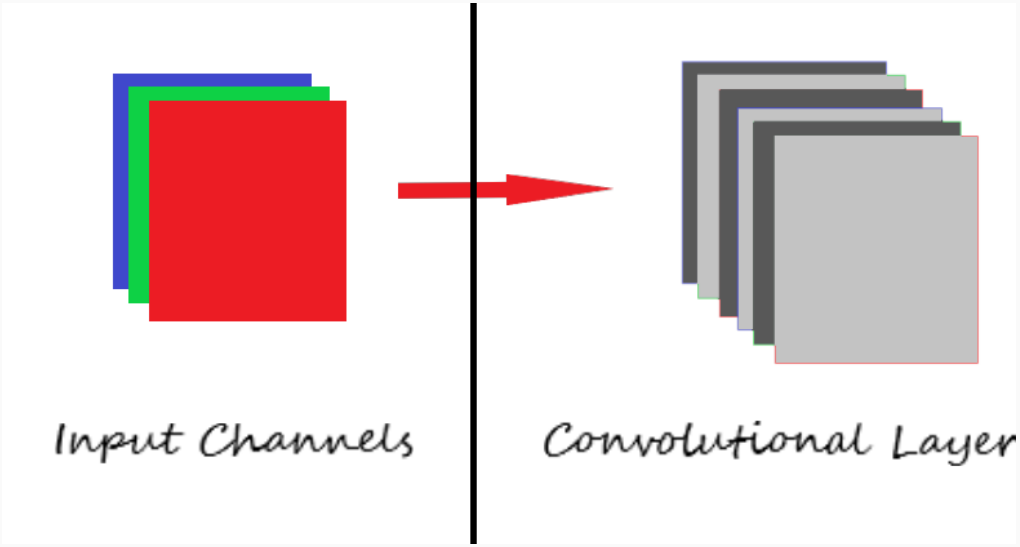
Main parameters of a convolutional layer

- Size of the filter K

Main parameters of a convolutional layer

- Size of the filter K
- Number of filters p

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

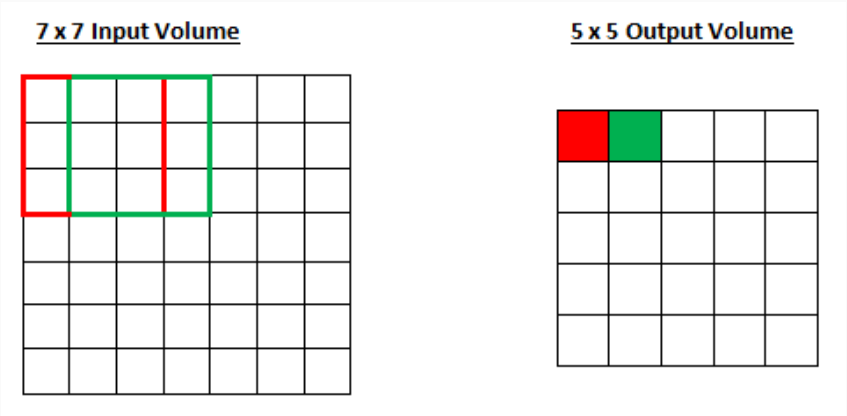


$O = \frac{W-K+2P}{S} + 1$, where O is the output size and W the input size.

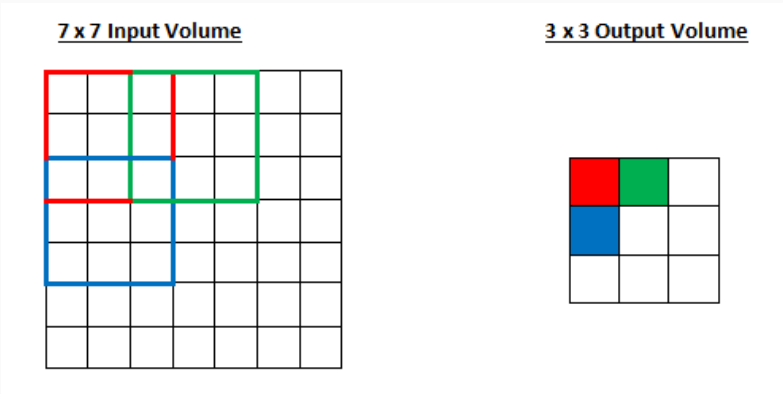
Main parameters of a convolutional layer

- Size of the filter K
- Number of filters p
- Strides S

$S = 1$



$S = 2$

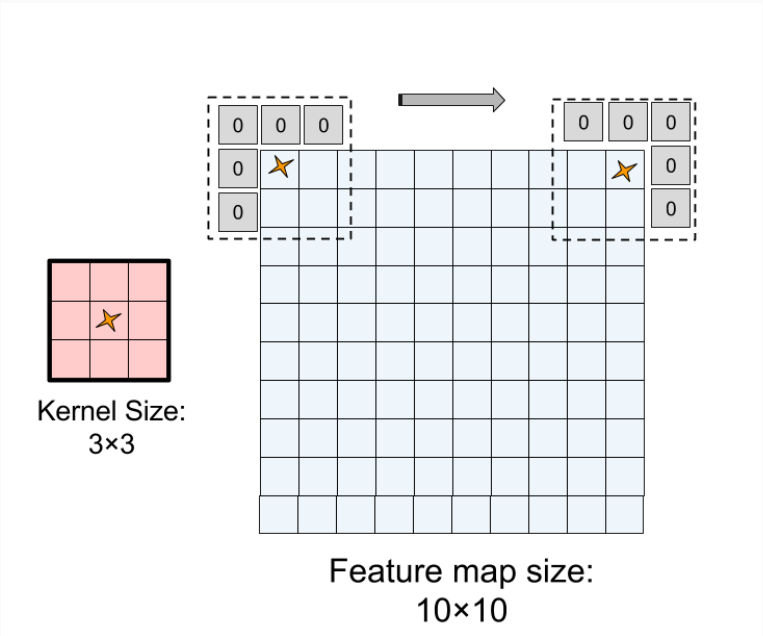


$O = \frac{W-K+2P}{S} + 1$, where O is the output size and W the input size.

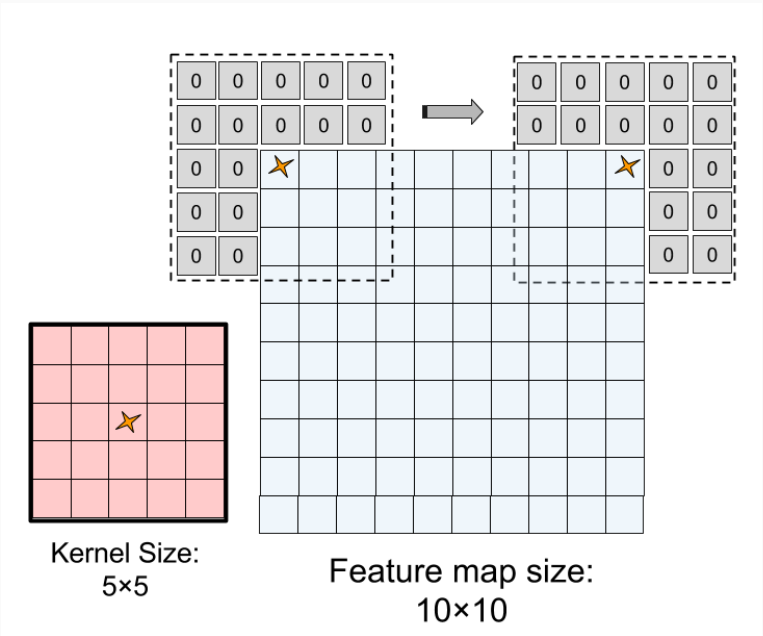
Main parameters of a convolutional layer

- Size of the filter K
- Number of filters p
- Strides S
- Padding P

$P = 1$



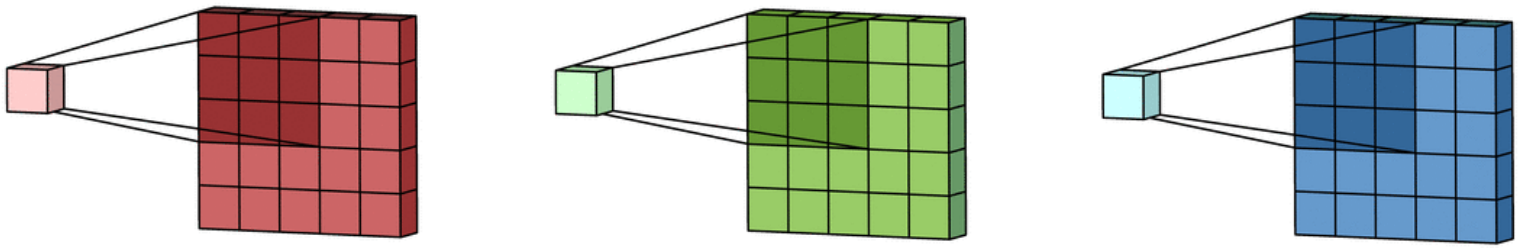
$P = 2$



$O = \frac{W-K+2P}{S} + 1$, where O is the output size and W the input size.

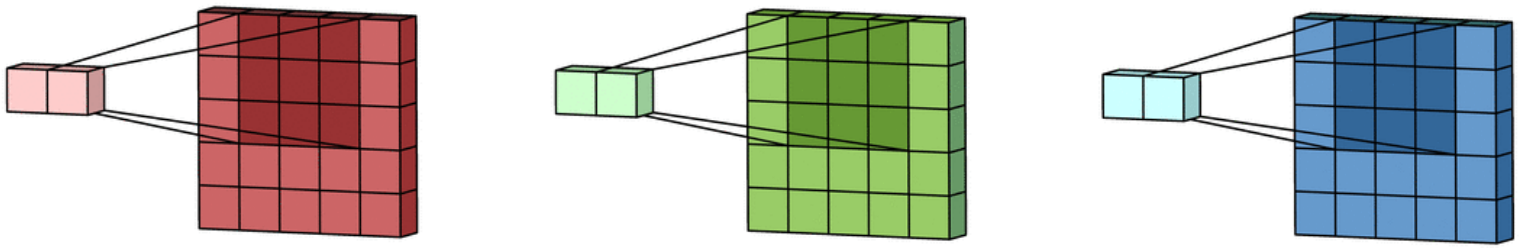
Summary of Convolutional layer steps

1. Convolution



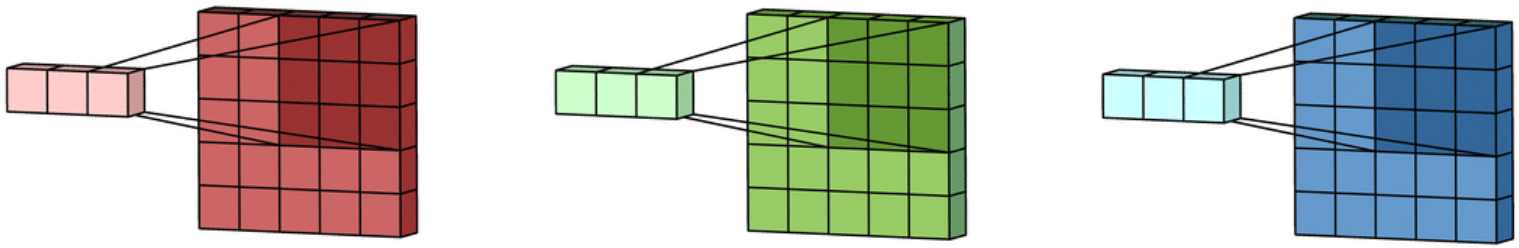
Summary of Convolutional layer steps

1. Convolution



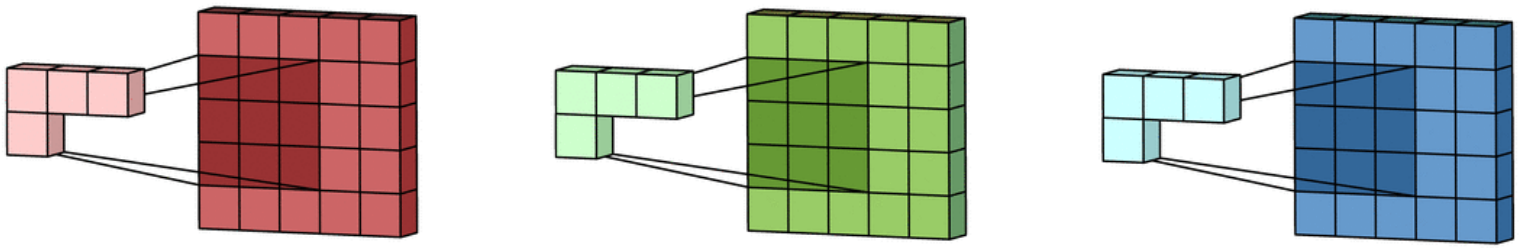
Summary of Convolutional layer steps

1. Convolution



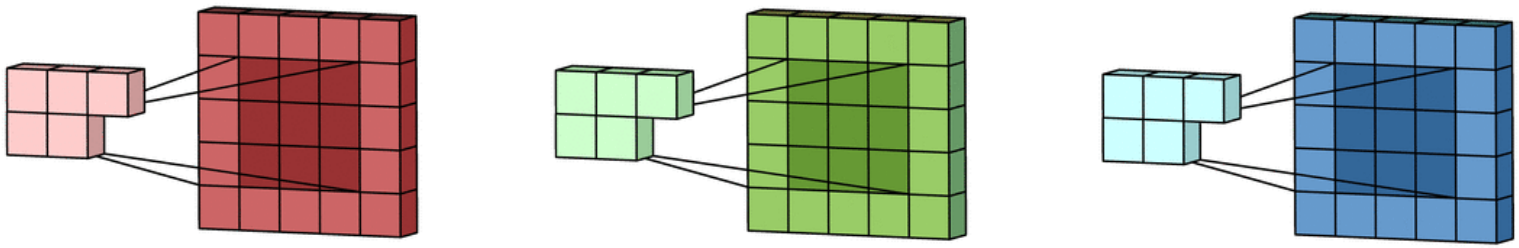
Summary of Convolutional layer steps

1. Convolution



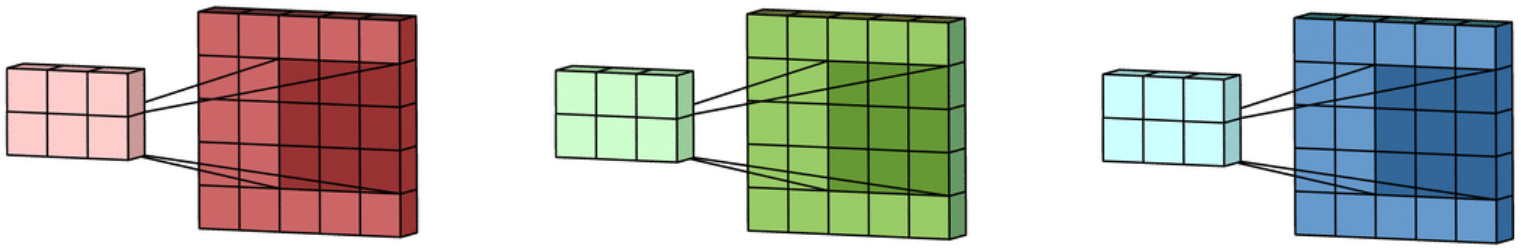
Summary of Convolutional layer steps

1. Convolution



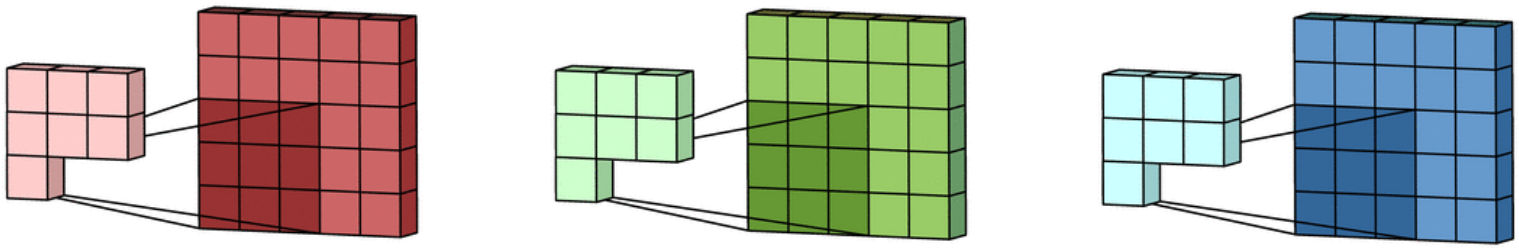
Summary of Convolutional layer steps

1. Convolution



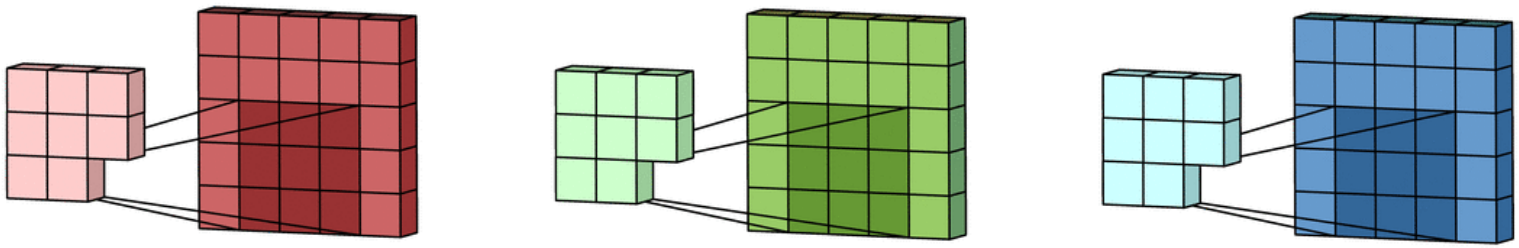
Summary of Convolutional layer steps

1. Convolution



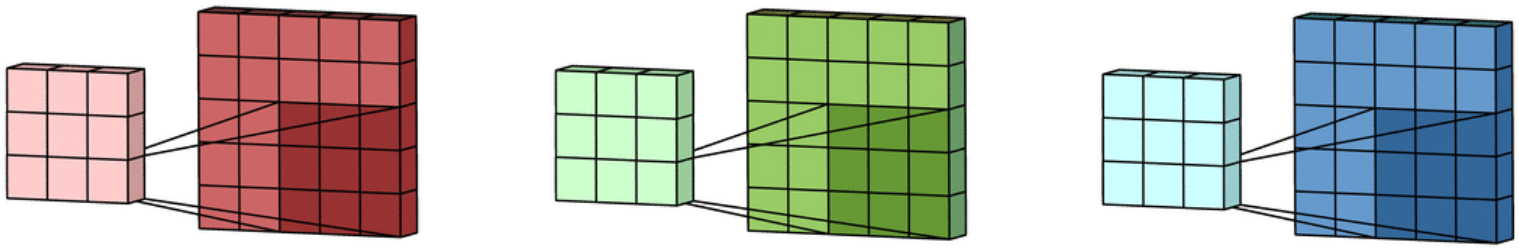
Summary of Convolutional layer steps

1. Convolution



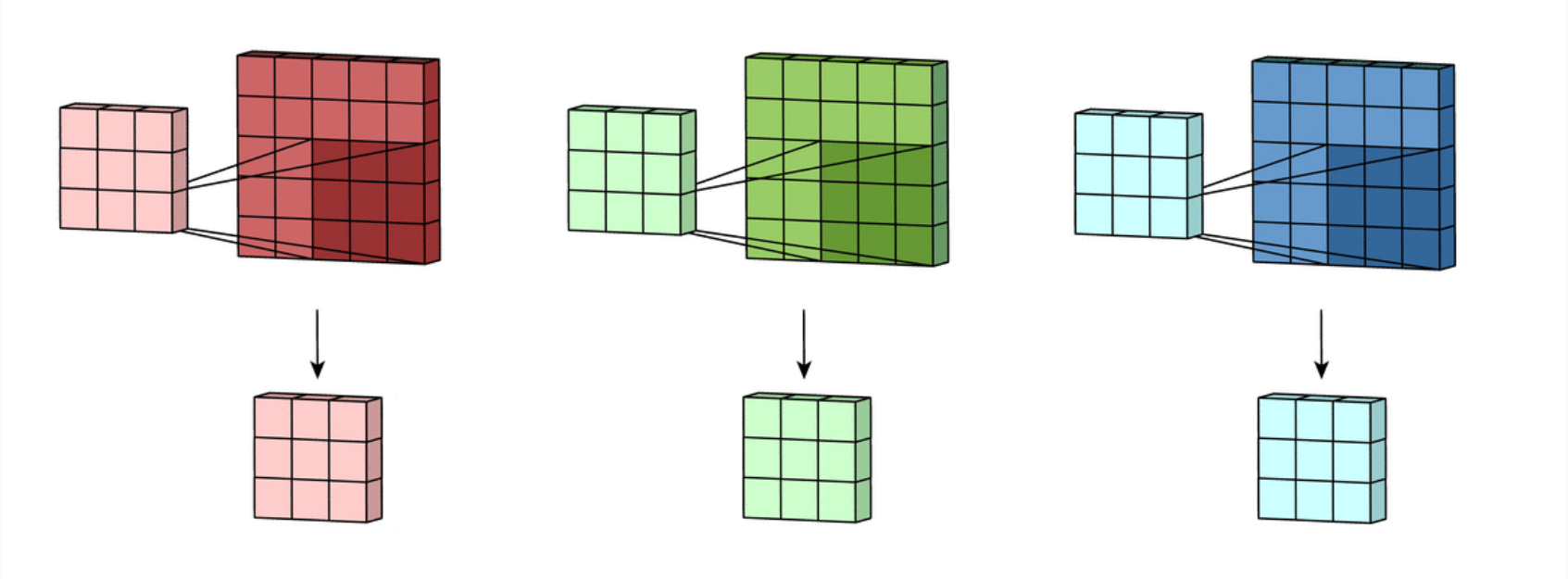
Summary of Convolutional layer steps

1. Convolution



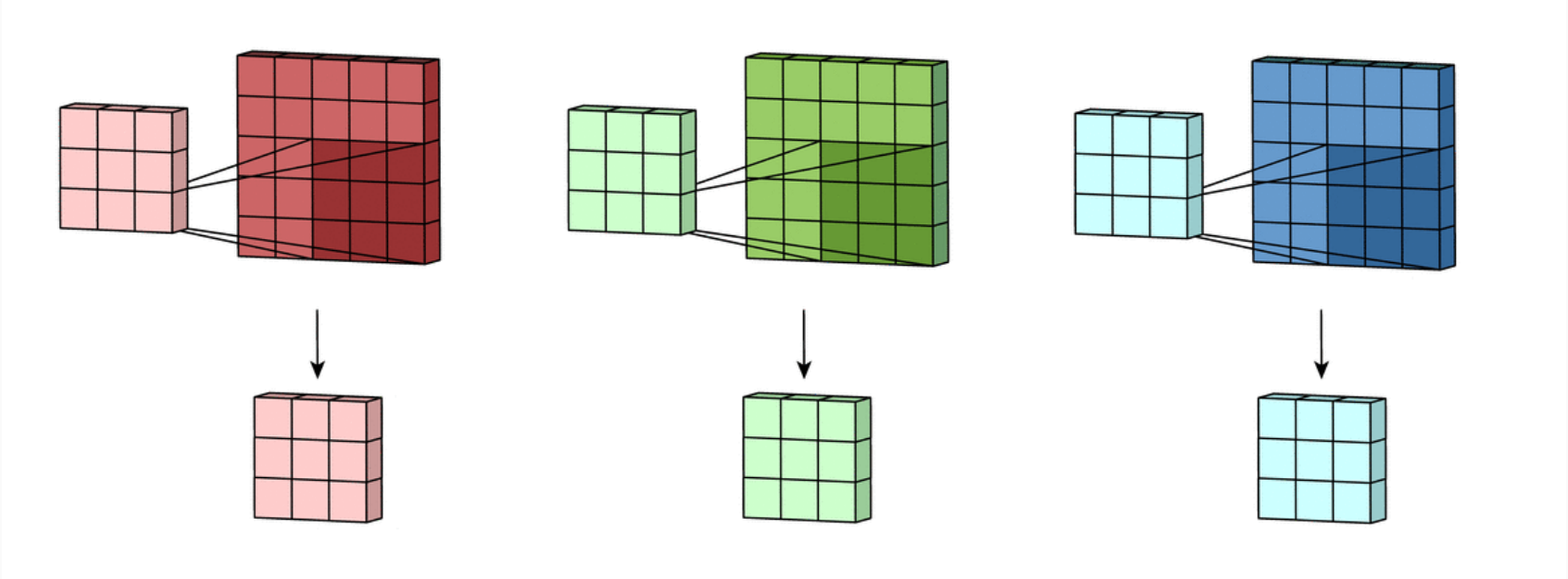
Summary of Convolutional layer steps

1. Convolution



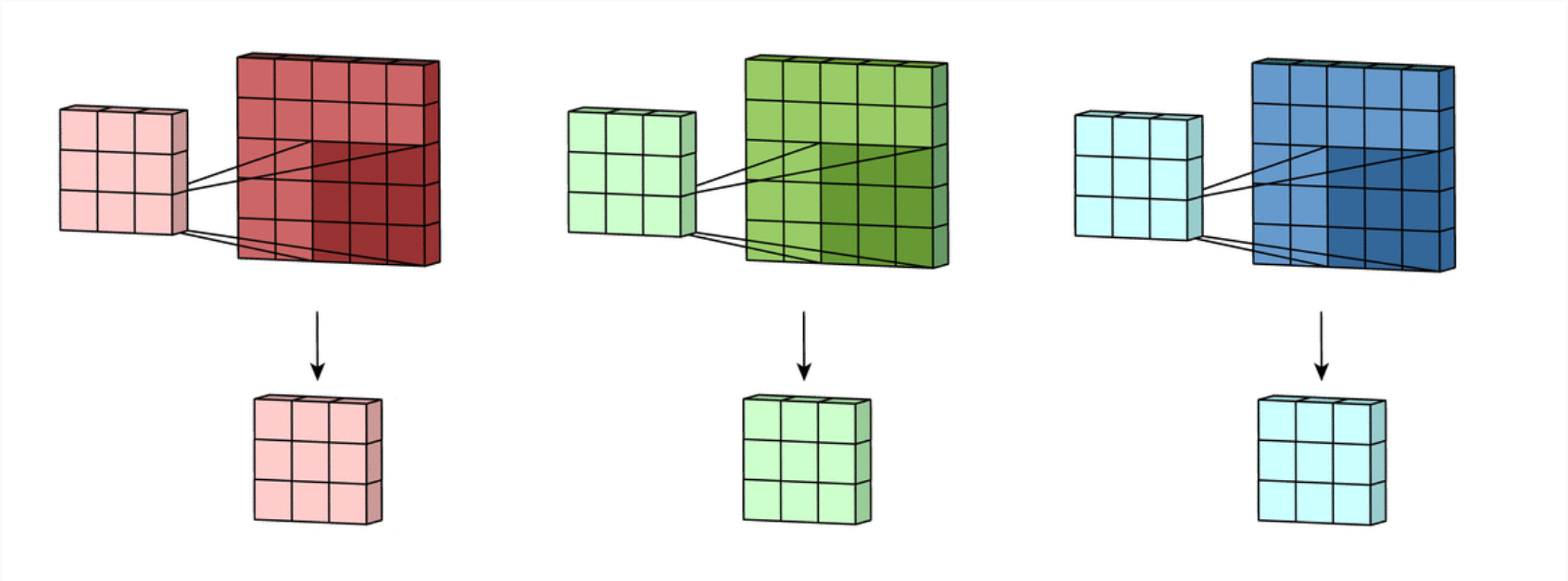
Summary of Convolutional layer steps

1. Convolution

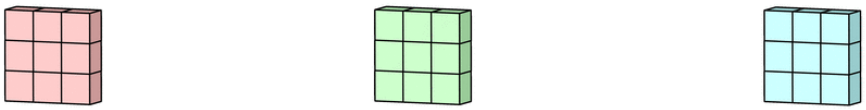


Summary of Convolutional layer steps

1. Convolution

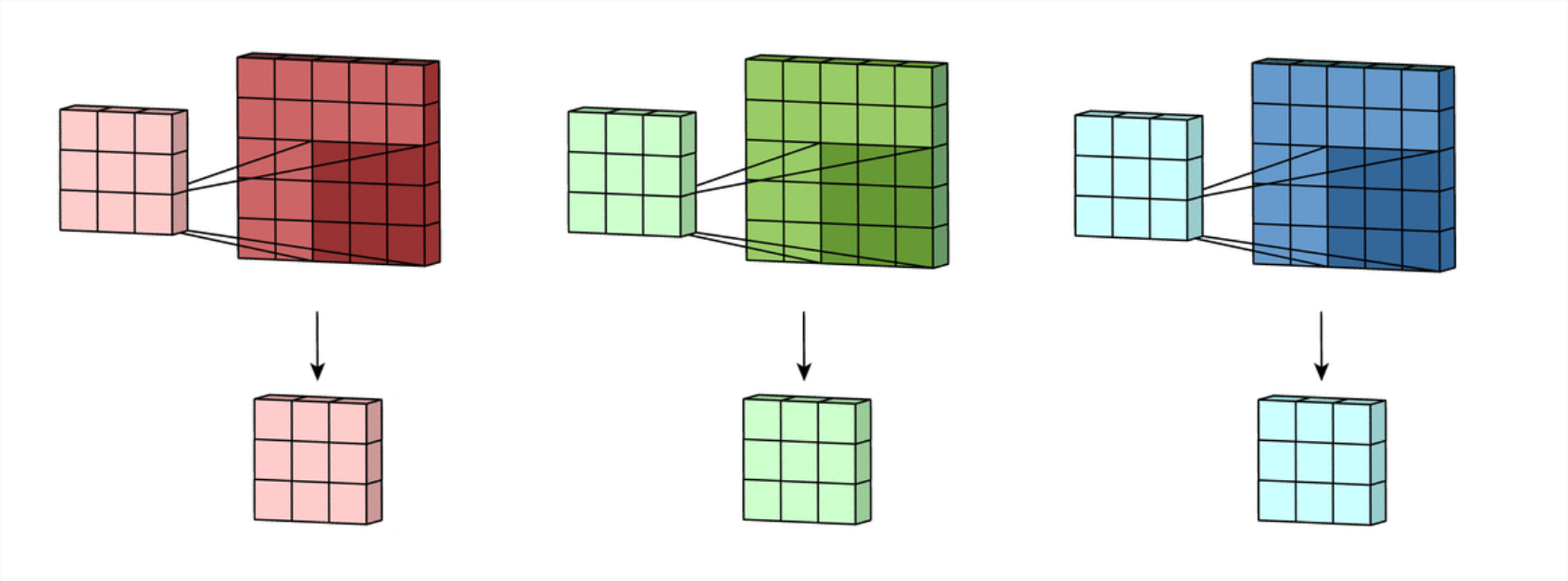


2. Addition

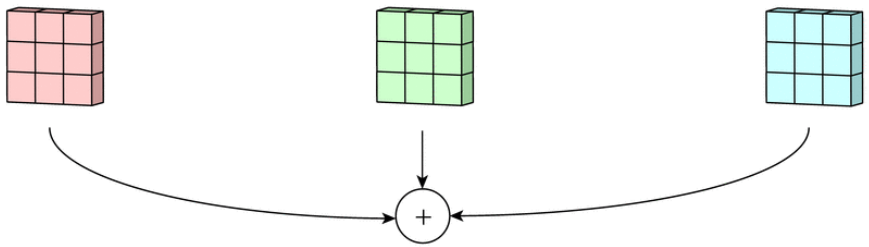


Summary of Convolutional layer steps

1. Convolution

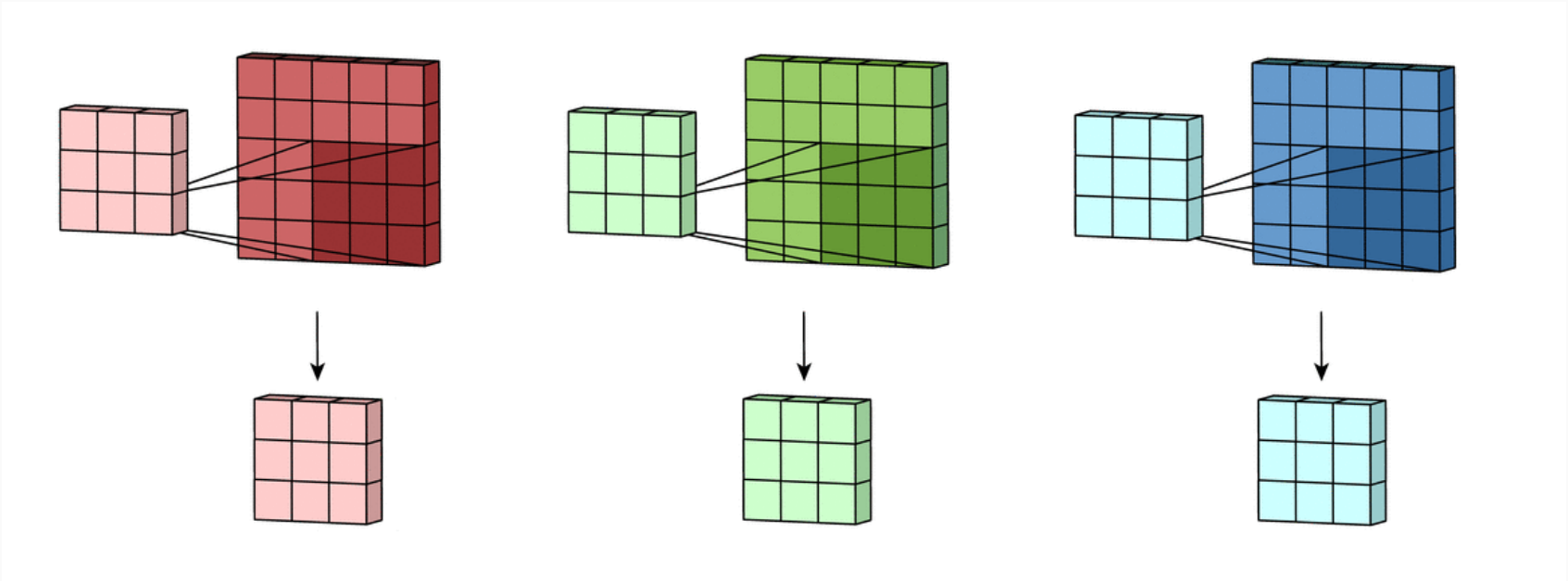


2. Addition

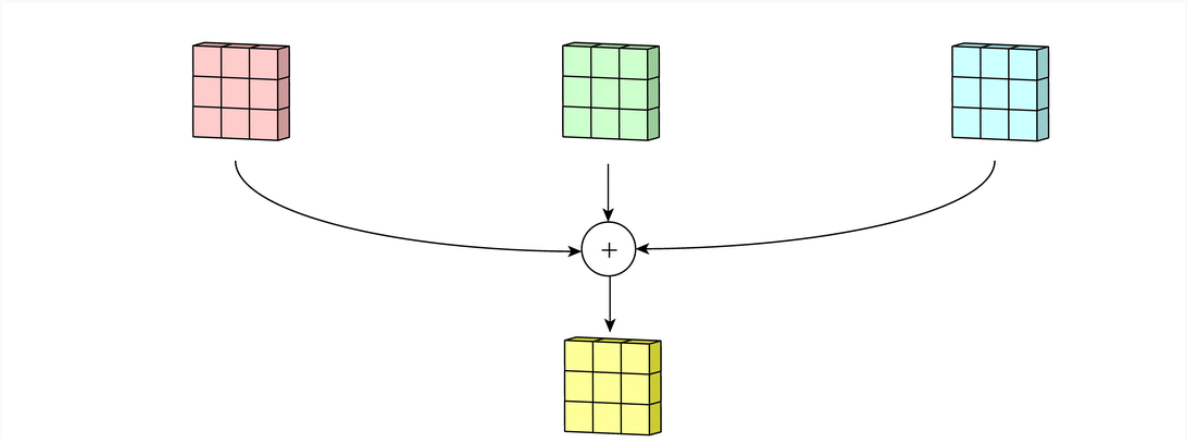


Summary of Convolutional layer steps

1. Convolution

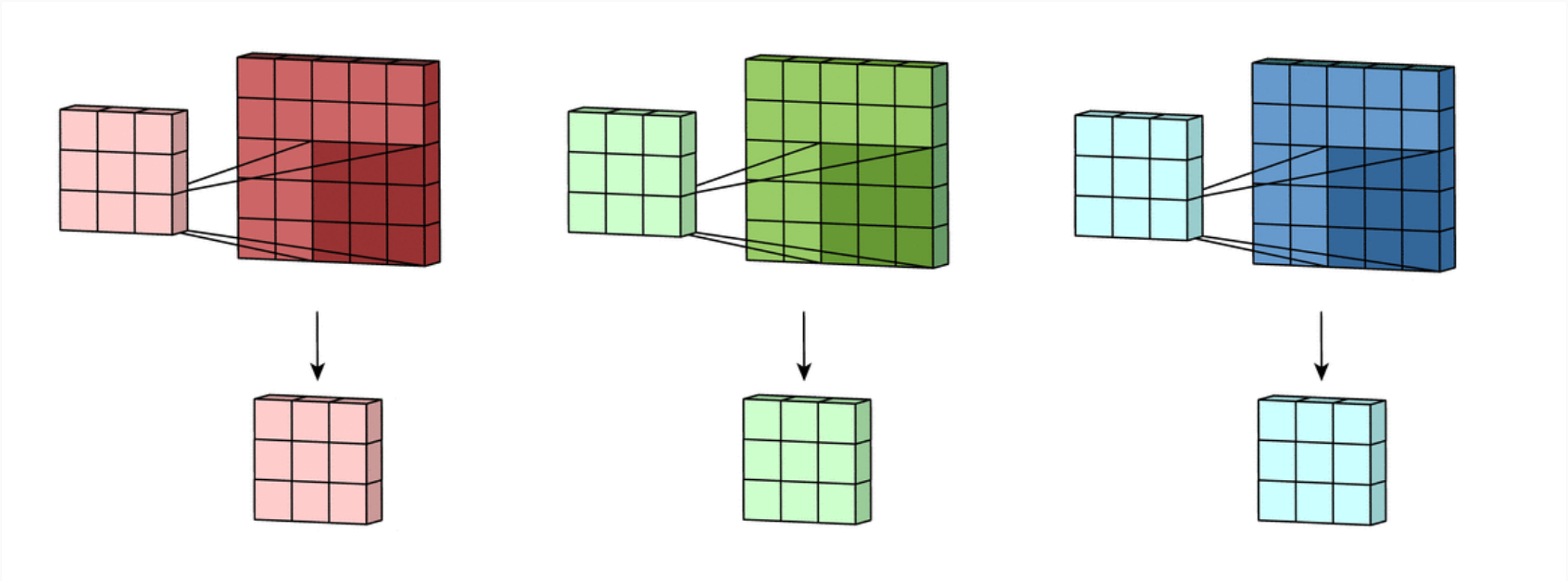


2. Addition

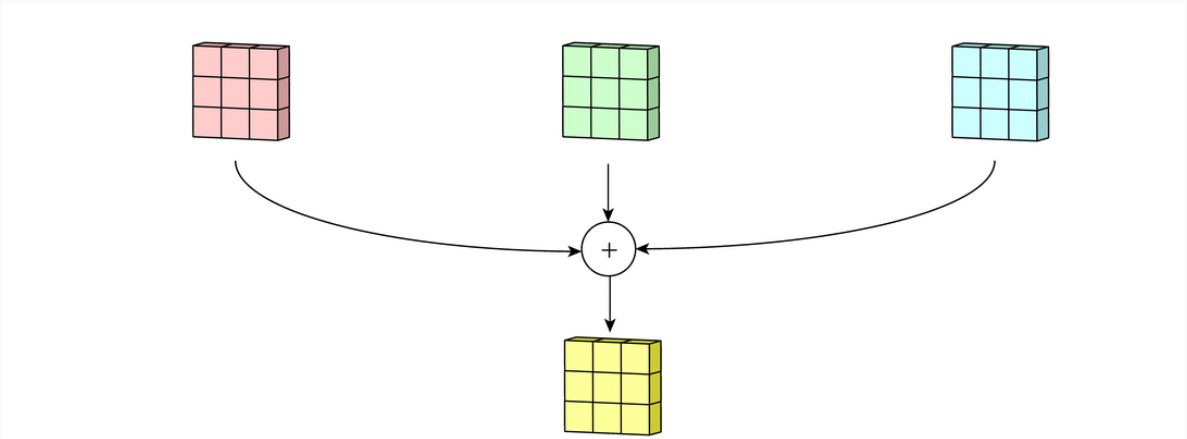


Summary of Convolutional layer steps

1. Convolution

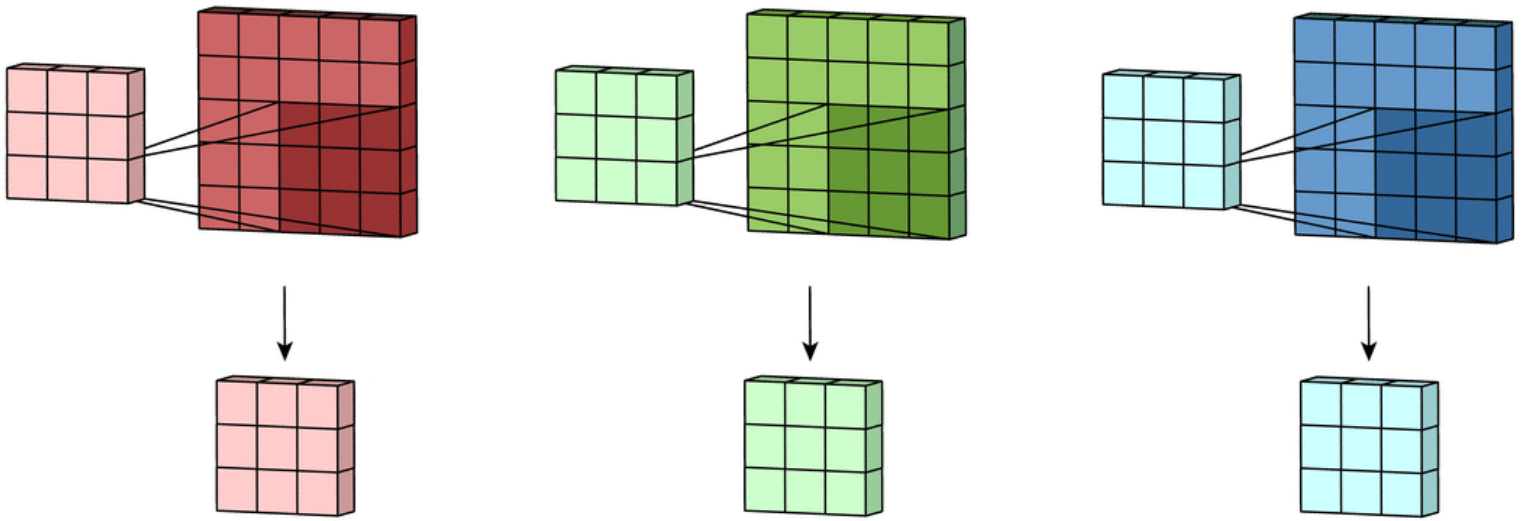


2. Addition

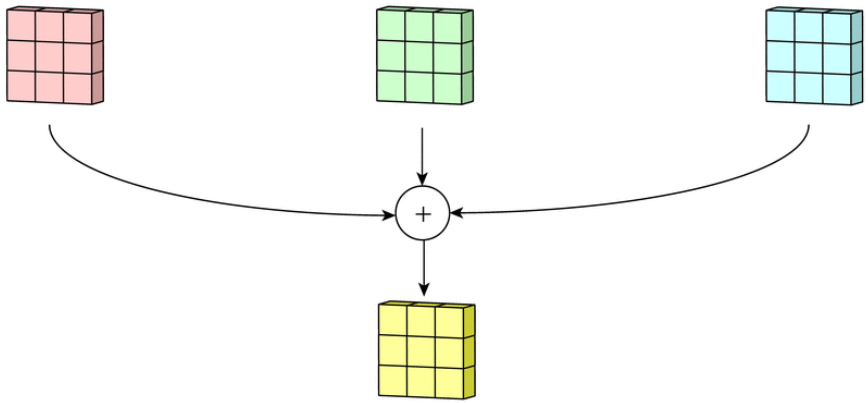


Summary of Convolutional layer steps

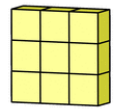
1. Convolution



2. Addition

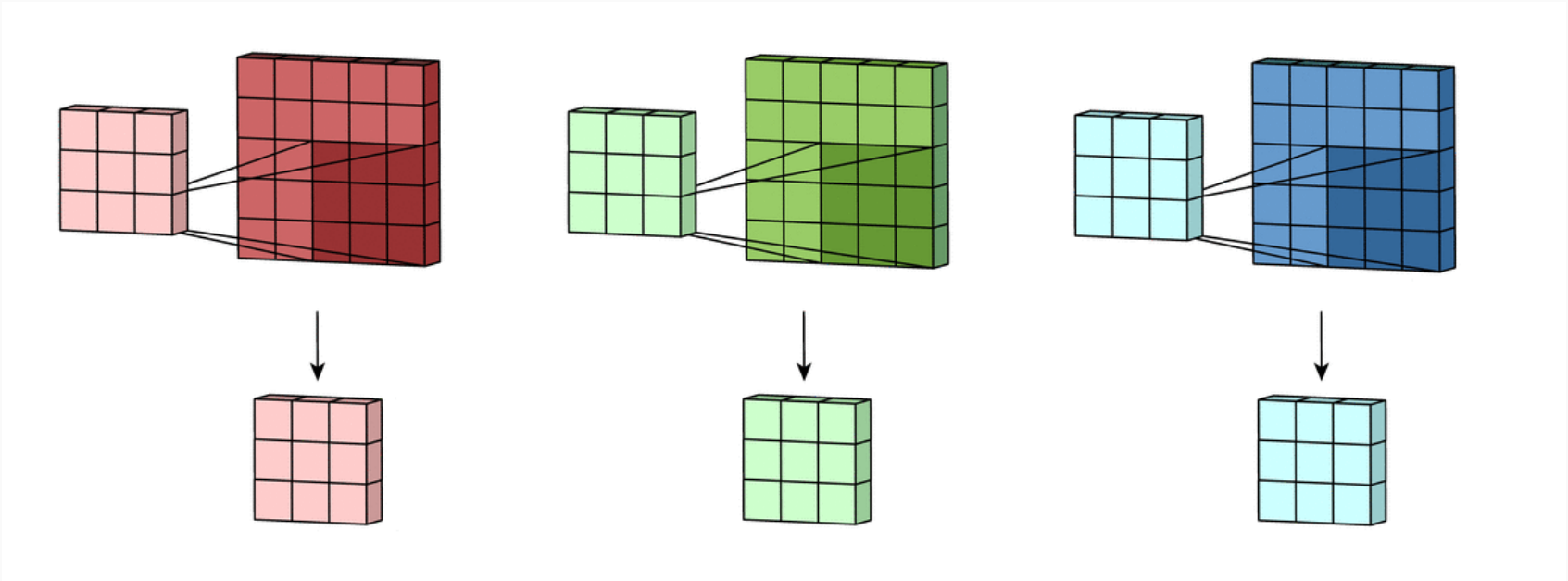


3. Bias

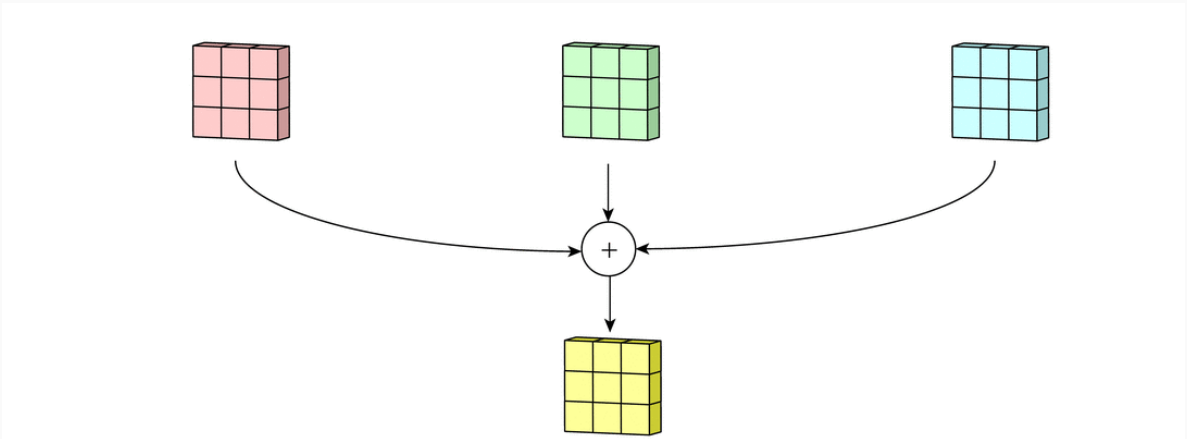


Summary of Convolutional layer steps

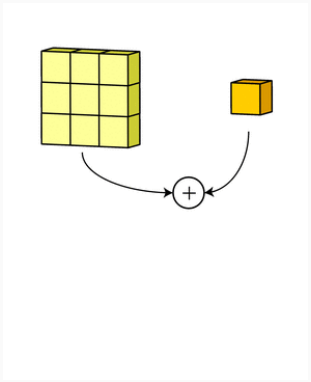
1. Convolution



2. Addition

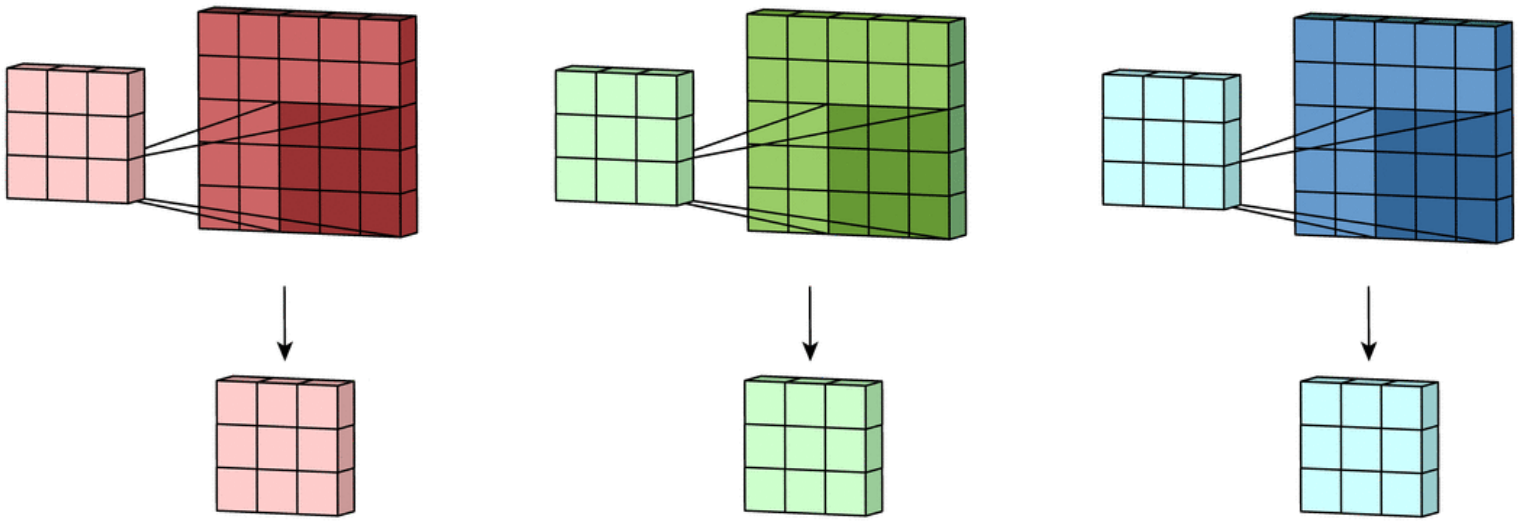


3. Bias

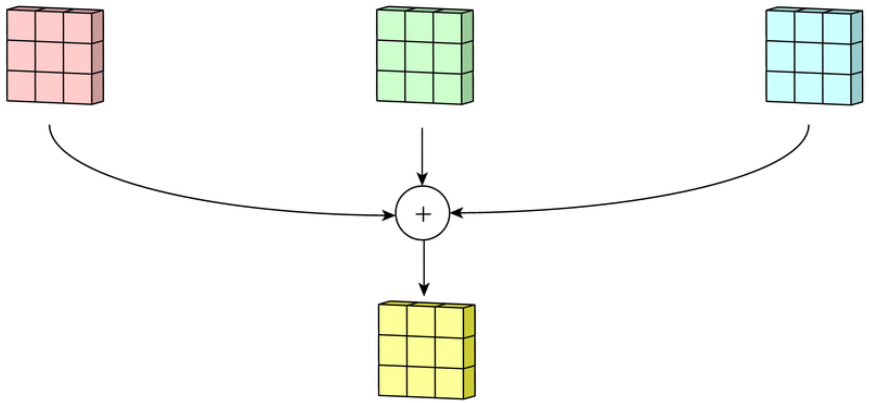


Summary of Convolutional layer steps

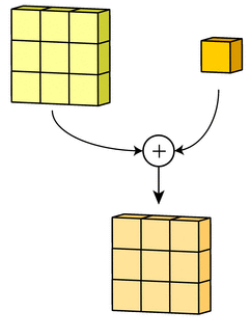
1. Convolution



2. Addition



3. Bias

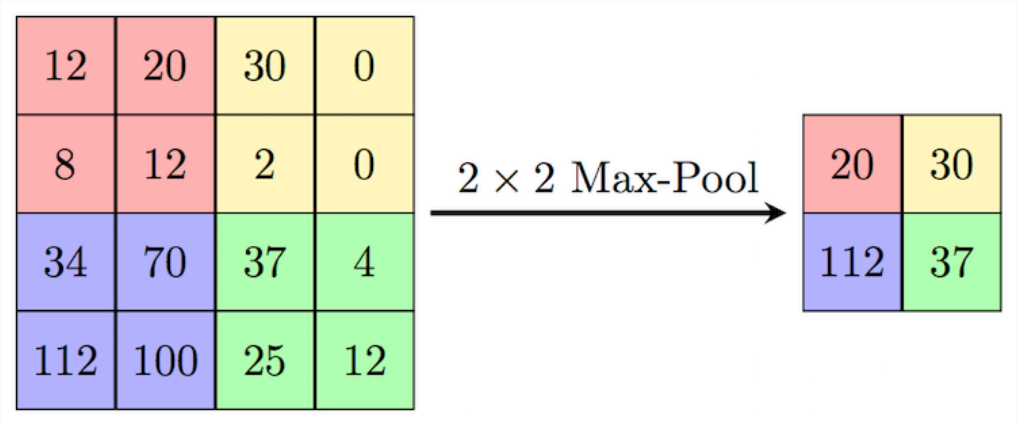


Remarks on Convolutional layers

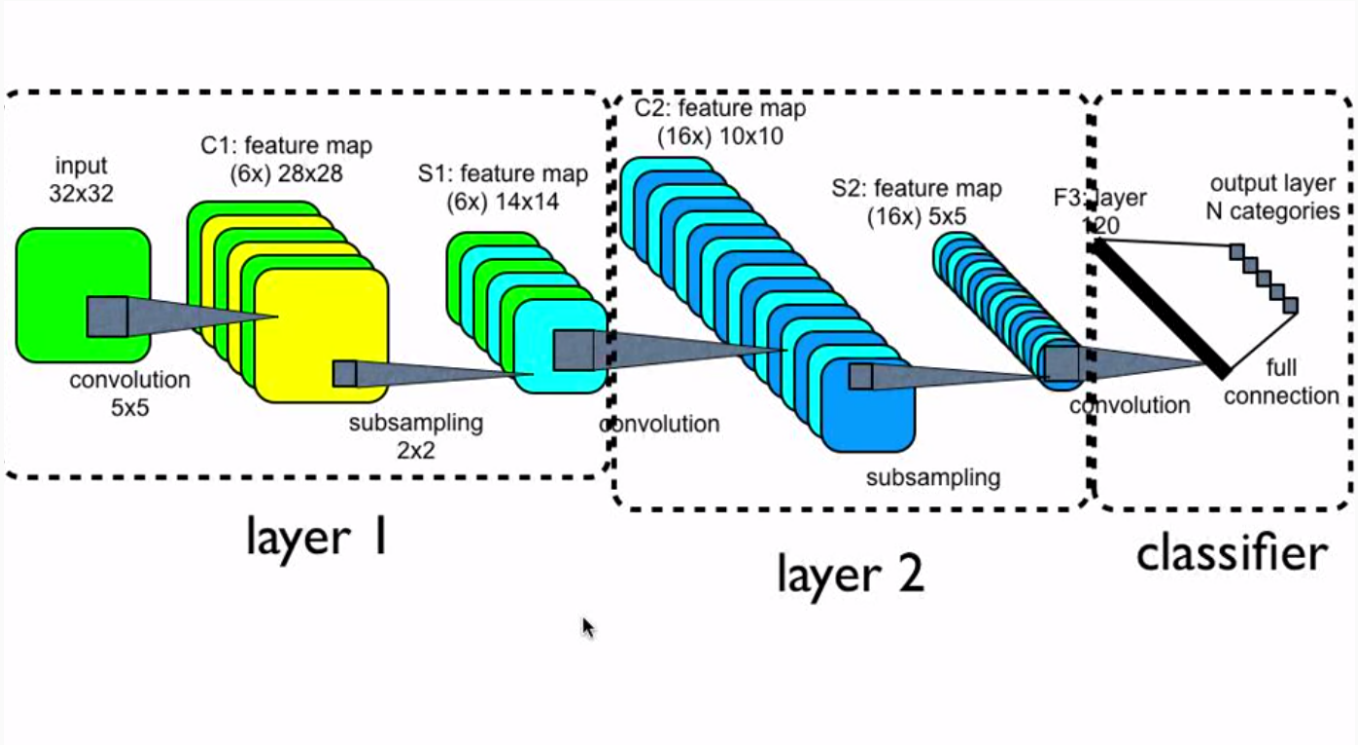
- Convolutional layers are acting locally on the image (But you can still use large scale information by adding more layers)
- Convolutions are invariant by translation (the weights do not depend on the location on the image).
- They can handle images of different sizes.

Max-Pooling

In order to reduce the size of the feature space (en to enhance the gradients), a common operation is to perform a max-pooling.



A traditionnal CNN architecture



Example of AlexNet

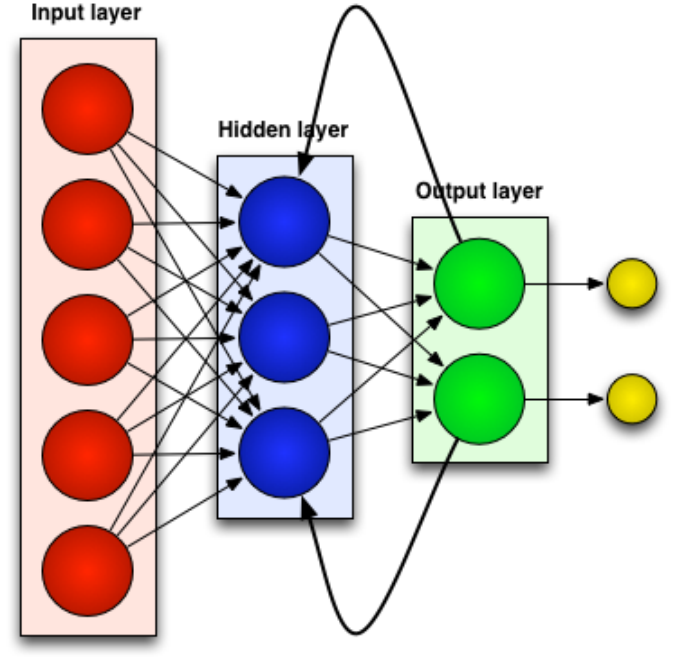
AlexNet is the first Deep architecture used on ImageNet challenge in 2012 and achieved an error of 15.3% (10% better than the previous best classifier). The paper was cited more than 34,000 times.

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

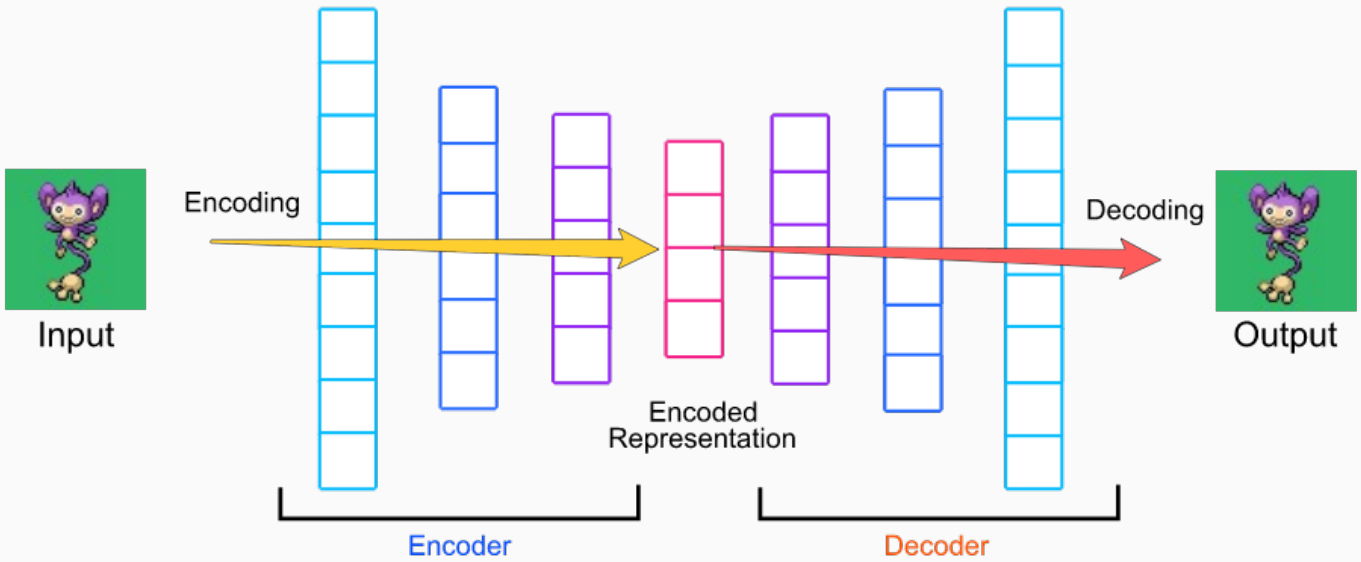


Some popular types of recurrent neural networks:

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

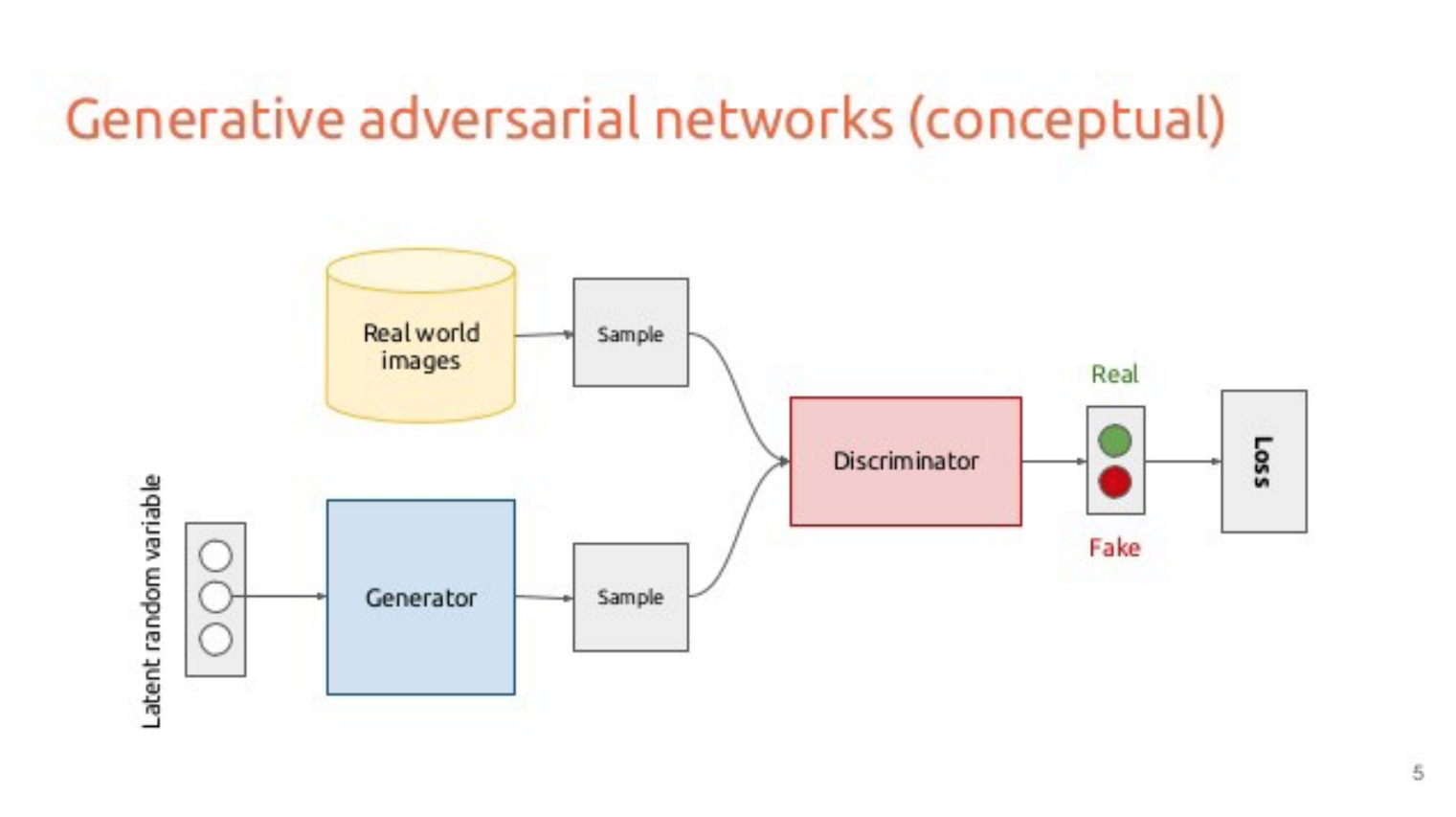
Used in machine translation and text processing

Autoencoders

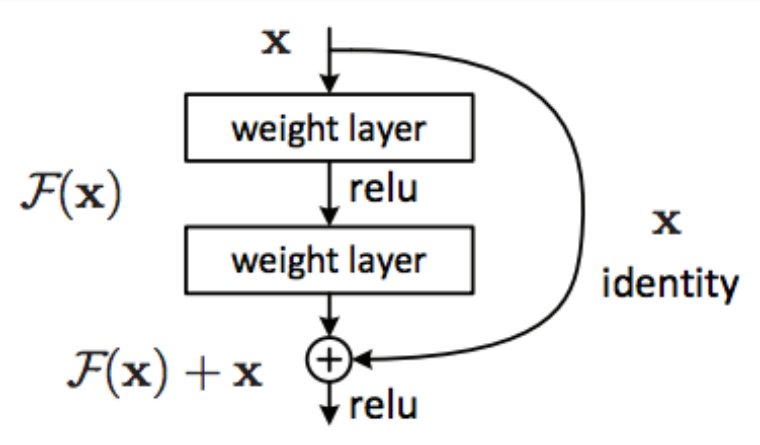


Used in image denoising, compressing, generation,...

Generative adversarial networks



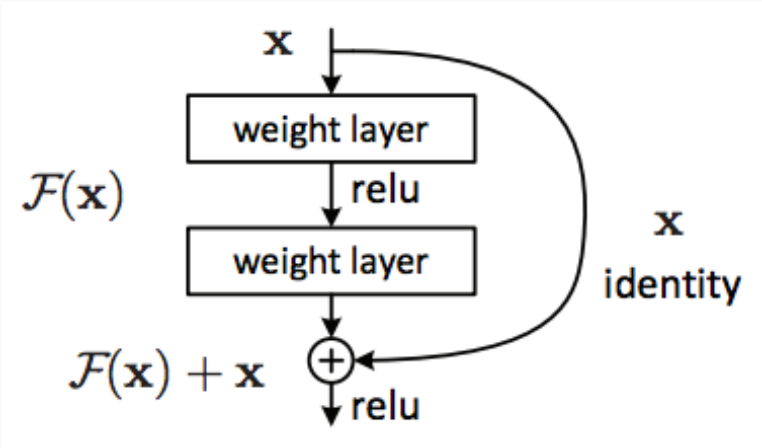
Residual Networks



x : input, y : output

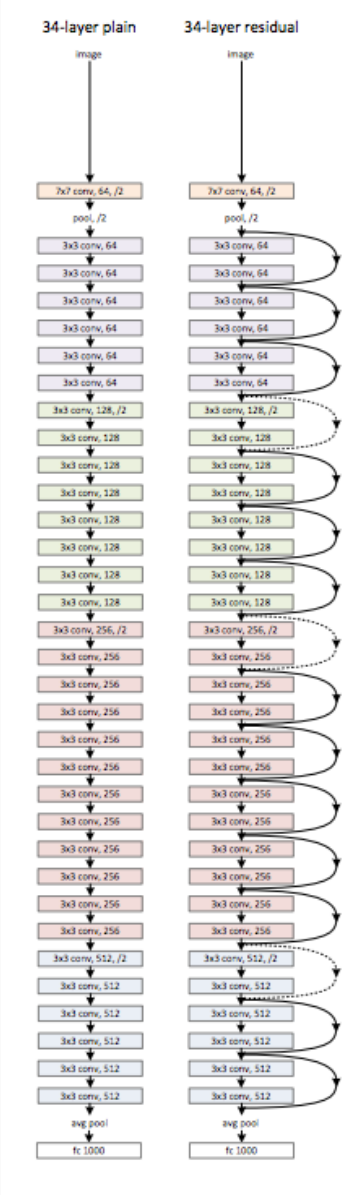
$$y = x + \mathcal{F}(x)$$

Residual Networks



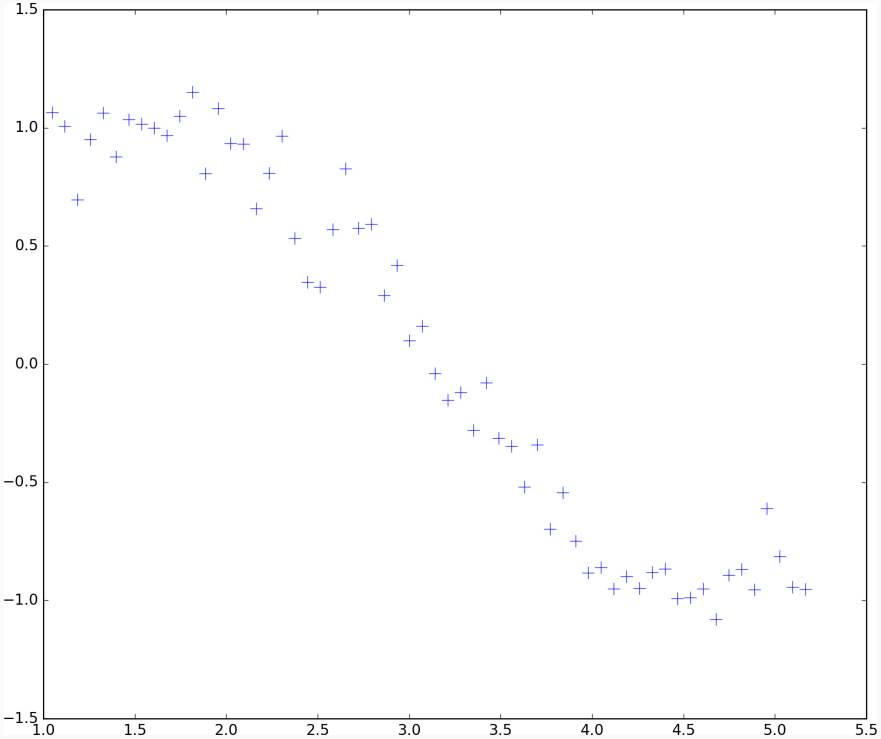
x : input, y : output

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



L1/L2 regularization

Example of a non-linear relashonship



An idea

We could take an polynomial hypothesis model:

$$h_{\boldsymbol{\theta}}(x) = \theta_0 x^0 + \theta_1 x^1 + \ldots + \theta_p x^p$$

Example

$\{(x_1, y_1), \ldots, (x_n, y_n)\}$ is the learning dataset.

For a given polynomial degree p, parameters $\boldsymbol{\theta}$ are determined minimizing the least-mean square cost function:

$$J(\boldsymbol{\theta}) = \frac{1}{n} \sum (y_i - h_{\boldsymbol{\theta}}(x_i))^2$$

with $h_{\boldsymbol{\theta}}(x) = \theta_0 x^0 + \theta_1 x^1 + \ldots + \theta_p x^p$

- It can be determined using a gradient descent method

Example

$\{(x_1, y_1), \ldots, (x_n, y_n)\}$ is the learning dataset.

For a given polynomial degree p, parameters $\boldsymbol{\theta}$ are determined minimizing the least-mean square cost function:

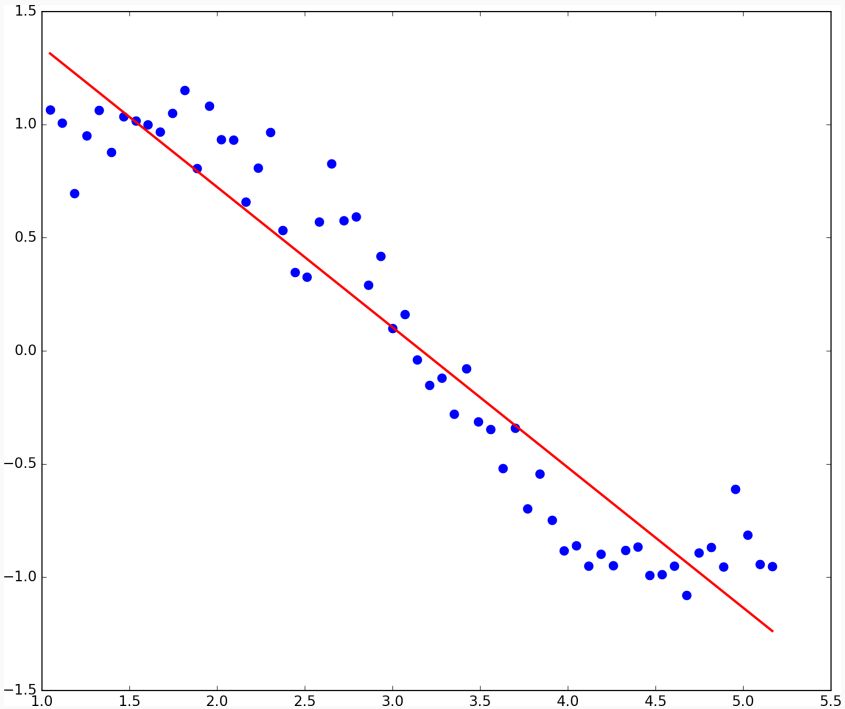
$$J(\boldsymbol{\theta}) = \frac{1}{n} \sum (y_i - h_{\boldsymbol{\theta}}(x_i))^2$$

with $h_{\boldsymbol{\theta}}(x) = \theta_0 x^0 + \theta_1 x^1 + \ldots + \theta_p x^p$

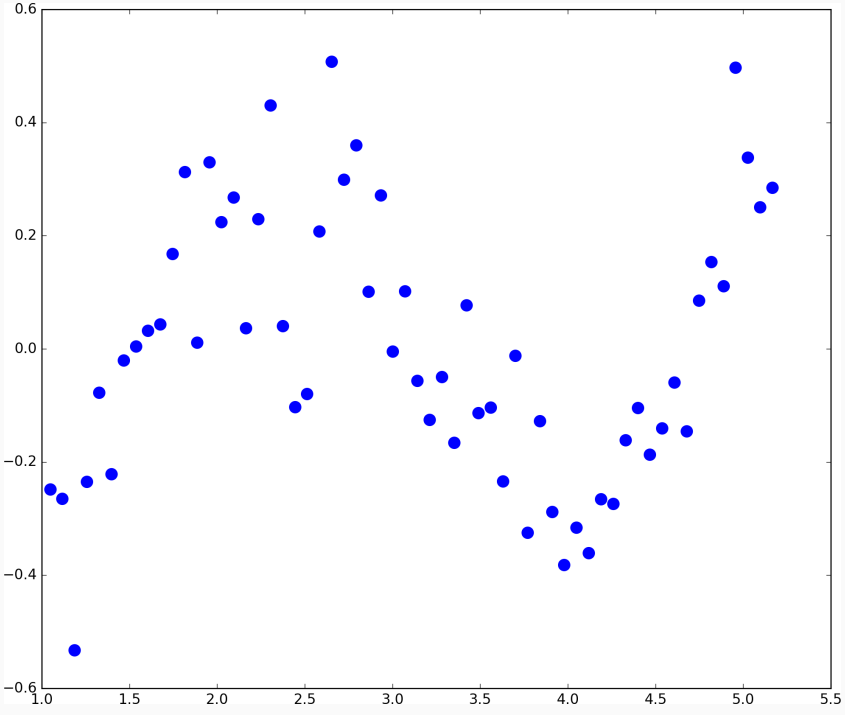
- It can be determined using a gradient descent method
- If the degree of the polynomial $p = 1$, it is a simple linear regression

A first result

$p = 1$ (linear regression)



residuals

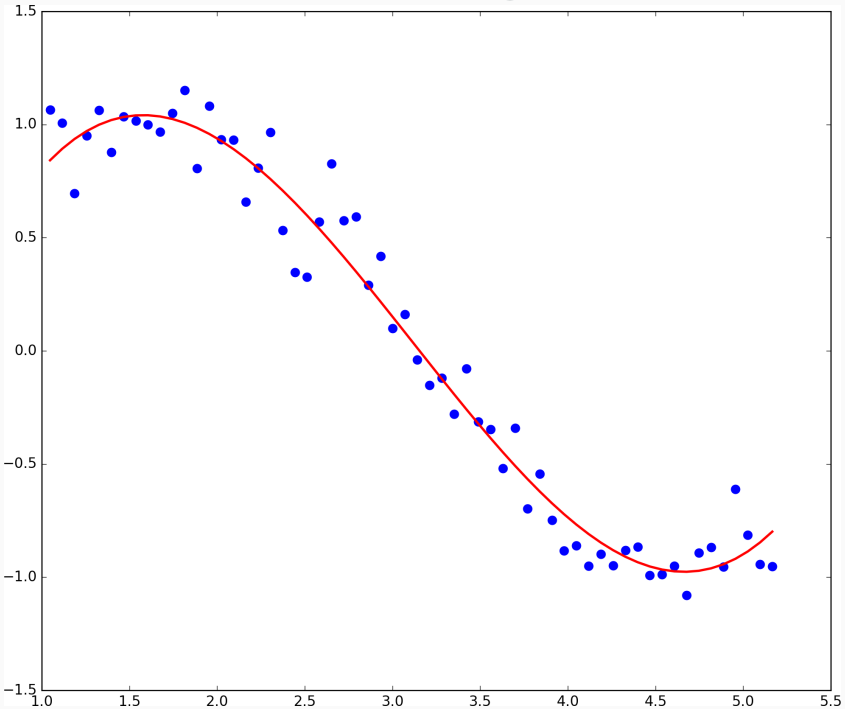


Prediction error

$$\text{err} = \frac{1}{n}\sum \text{res}^2 = 5.46e-2$$

Increasing the polynomial degree ?

$p = 3$ (cubic regression)



Prediction error

$$\text{err} = \frac{1}{n}\sum \text{res}^2 = 1.80e-2$$

Is it different from the linear regression ?

Let's consider :

$$\boldsymbol{x} = \begin{pmatrix} 1 \\ x^1 \\ x^2 \\ \vdots \\ x^p \end{pmatrix}$$

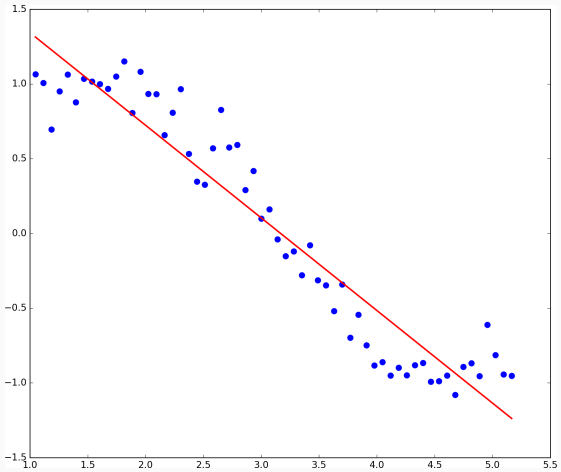
then

$$h_{\boldsymbol{\theta}}(x) = \theta_0 + \theta_1 x^1 + \ldots + \theta_p x^p = \boldsymbol{\theta}^T \boldsymbol{x}$$

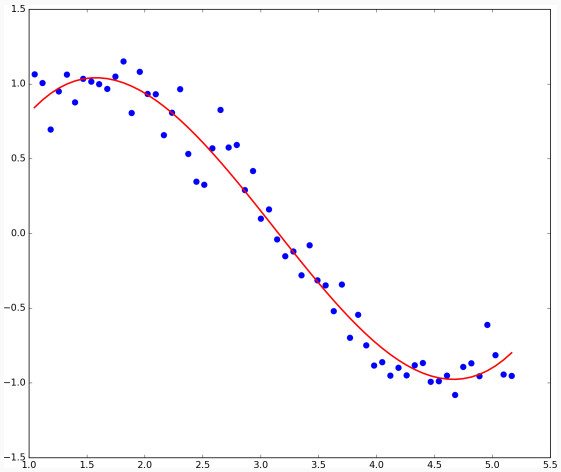
By extending a scalar predictor to a vector, polynomial regression is equivalent to linear regression.

Increasing the degree ?

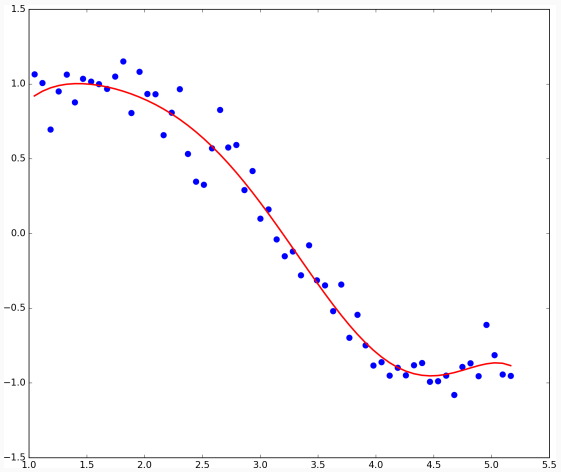
$p = 1$



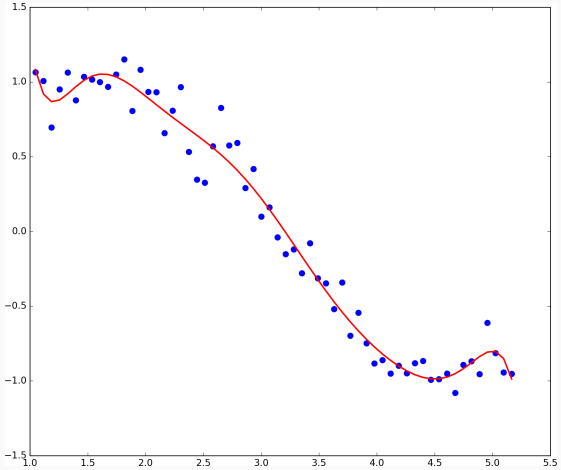
$p = 3$



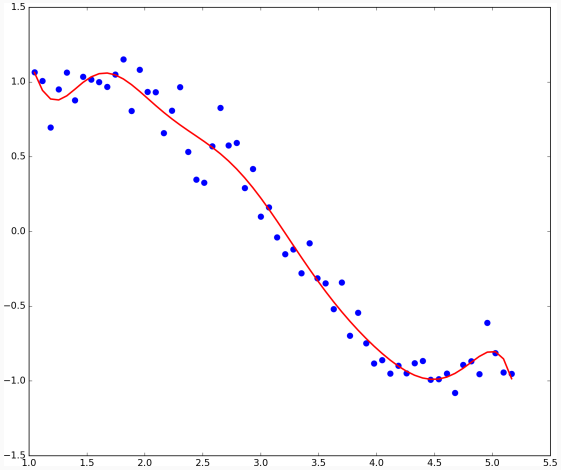
$p = 4$



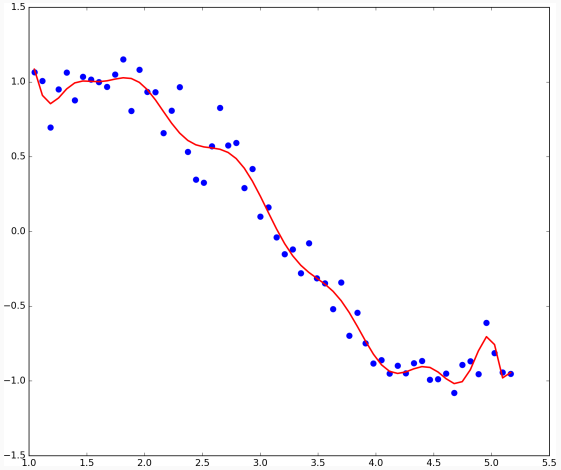
$p = 9$



$p = 12$



$p = 15$



Overfitting

When there is to many paramaters to fit, the model can reproduce a random noise, it is called overfitting

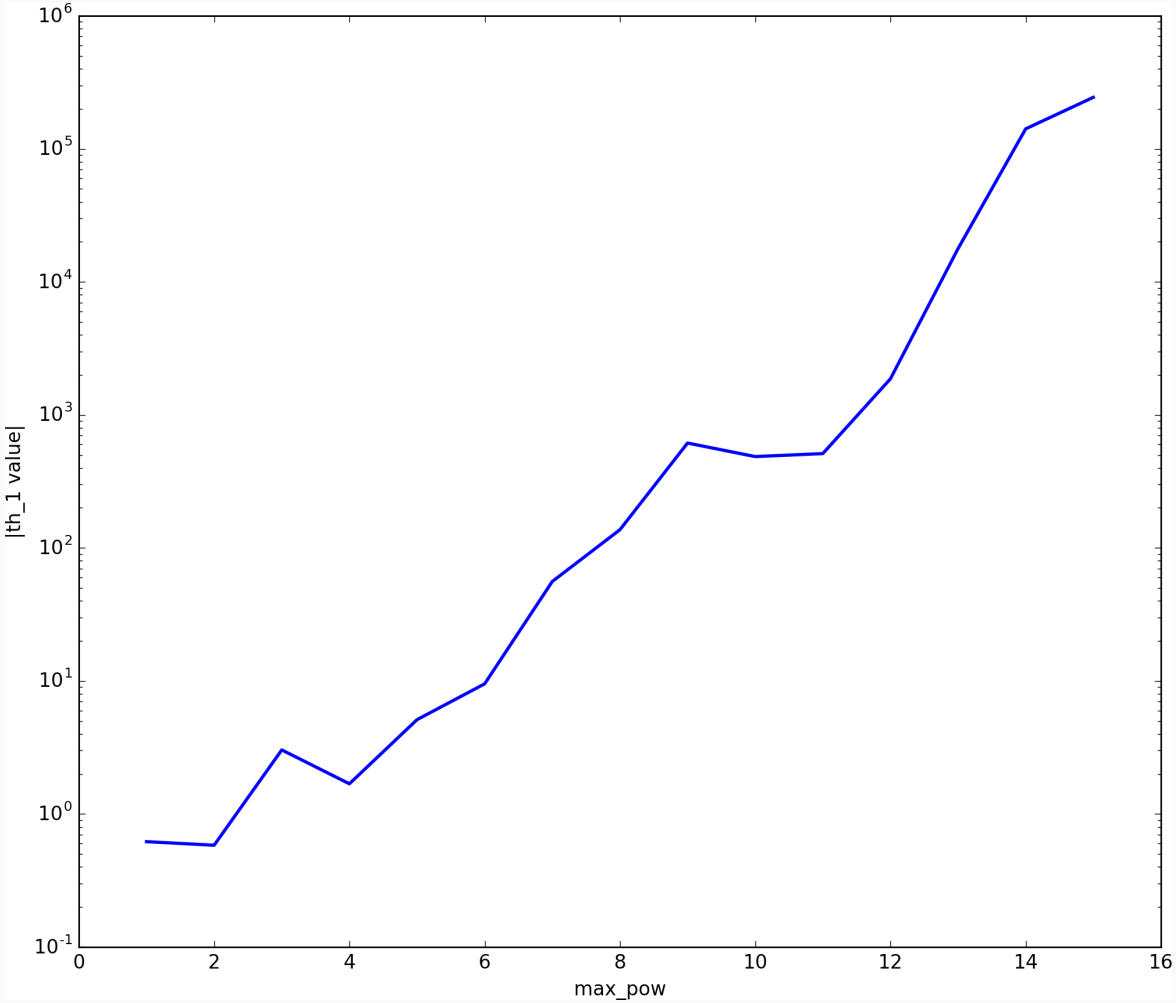
Overfitting

When there is to many paramaters to fit, the model can reproduce a random noise, it is called overfitting

	rmse	th_0	th_1	th_2	th_3	th_4	th_5	th_6	th_7	th_8
max_pow_1	+5.47e-02	+1.96e+00	-6.20e-01	NaN	NaN	NaN	NaN	NaN	NaN	NaN
max_pow_2	+5.46e-02	+1.91e+00	-5.83e-01	-5.96e-03	NaN	NaN	NaN	NaN	NaN	NaN
max_pow_3	+1.84e-02	-1.08e+00	+3.03e+00	-1.29e+00	+1.37e-01	NaN	NaN	NaN	NaN	NaN
max_pow_4	+1.80e-02	-2.66e-01	+1.69e+00	-5.32e-01	-3.57e-02	+1.39e-02	NaN	NaN	NaN	NaN
max_pow_5	+1.70e-02	+2.99e+00	-5.12e+00	+4.72e+00	-1.93e+00	+3.35e-01	-2.07e-02	NaN	NaN	NaN
max_pow_6	+1.65e-02	-2.80e+00	+9.52e+00	-9.71e+00	+5.23e+00	-1.55e+00	+2.33e-01	-1.36e-02	NaN	NaN
max_pow_7	+1.55e-02	+1.93e+01	-5.60e+01	+6.90e+01	-4.46e+01	+1.65e+01	-3.53e+00	+4.05e-01	-1.92e-02	NaN
max_pow_8	+1.53e-02	+4.32e+01	-1.37e+02	+1.84e+02	-1.33e+02	+5.77e+01	-1.53e+01	+2.42e+00	-2.10e-01	+7.68e-03
max_pow_9	+1.46e-02	+1.68e+02	-6.15e+02	+9.63e+02	-8.46e+02	+4.61e+02	-1.62e+02	+3.68e+01	-5.22e+00	+4.22e-01
max_pow_10	+1.46e-02	+1.38e+02	-4.86e+02	+7.26e+02	-5.96e+02	+2.93e+02	-8.75e+01	+1.45e+01	-8.06e-01	-1.38e-01
max_pow_11	+1.45e-02	-7.49e+01	+5.12e+02	-1.33e+03	+1.87e+03	-1.61e+03	+9.14e+02	-3.50e+02	+9.14e+01	-1.61e+01
max_pow_12	+1.45e-02	-3.39e+02	+1.87e+03	-4.42e+03	+6.01e+03	-5.25e+03	+3.12e+03	-1.30e+03	+3.84e+02	-8.03e+01
max_pow_13	+1.43e-02	+3.20e+03	-1.78e+04	+4.46e+04	-6.66e+04	+6.61e+04	-4.61e+04	+2.32e+04	-8.55e+03	+2.30e+03
max_pow_14	+1.31e-02	+2.38e+04	-1.41e+05	+3.79e+05	-6.10e+05	+6.57e+05	-5.03e+05	+2.82e+05	-1.17e+05	+3.66e+04
max_pow_15	+1.17e-02	-3.62e+04	+2.44e+05	-7.46e+05	+1.38e+06	-1.71e+06	+1.53e+06	-1.00e+06	+4.98e+05	-1.88e+05

High parameters values

Value of the parameters $|\theta_1|$ with respect with the degree of the polynomial regression



Regularization

The idea

The idea of regularization is to perform a regression minimizing a cost function that includes a term to penalize "big" values for the parameters:

$$J(\boldsymbol{\theta}) = \frac{1}{n}\sum(y_i - h_{\boldsymbol{\theta}}(x_i))^2 + \alpha P(\boldsymbol{\theta})$$

Regularization

The idea

The idea of regularization is to perform a regression minimizing a cost function that includes a term to penalize "big" values for the parameters:

$$J(\boldsymbol{\theta}) = \frac{1}{n}\sum(y_i - h_{\boldsymbol{\theta}}(x_i))^2 + \alpha P(\boldsymbol{\theta})$$

We consider two penalty terms:

- Ridge Regularization (L2): $P(\boldsymbol{\theta}) = \sum_{i=0}^{p} \theta_i^2$

Regularization

The idea

The idea of regularization is to perform a regression minimizing a cost function that includes a term to penalize "big" values for the parameters:

$$J(\boldsymbol{\theta}) = \frac{1}{n}\sum(y_i - h_{\boldsymbol{\theta}}(x_i))^2 + \alpha P(\boldsymbol{\theta})$$

We consider two penalty terms:

- Ridge Regularization (L2): $P(\boldsymbol{\theta}) = \sum_{i=0}^{p} \theta_i^2$
- Lasso Regularization (L1): $P(\boldsymbol{\theta}) = \sum_{i=0}^{p} |\theta_i|$

Regularization

The idea

The idea of regularization is to perform a regression minimizing a cost function that includes a term to penalize "big" values for the parameters:

$$J(\boldsymbol{\theta}) = \frac{1}{n}\sum(y_i - h_{\boldsymbol{\theta}}(x_i))^2 + \alpha P(\boldsymbol{\theta})$$

We consider two penalty terms:

- Ridge Regularization (L2): $P(\boldsymbol{\theta}) = \sum_{i=0}^{p} \theta_i^2$
- Lasso Regularization (L1): $P(\boldsymbol{\theta}) = \sum_{i=0}^{p} |\theta_i|$
- Elastic Net combines both regularization

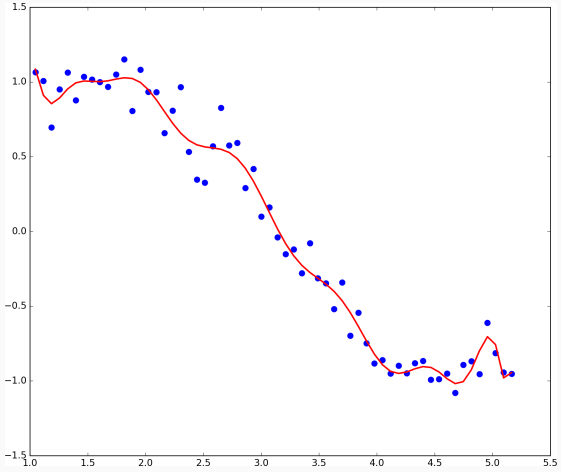
Ridge regression (L2)

Ridge regression is a linear regression with a Ridge regularization:

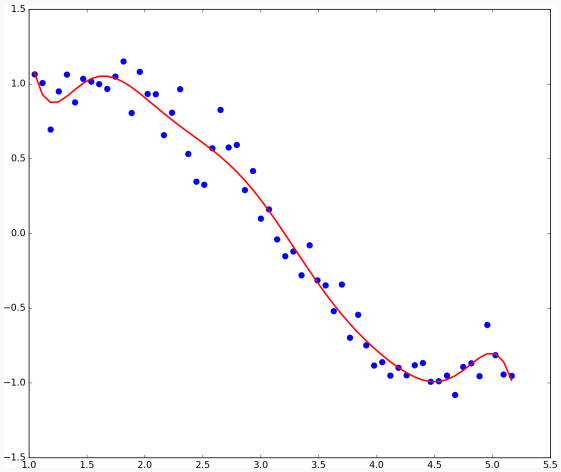
$$J(\boldsymbol{\theta}) = \frac{1}{n}\sum(y_i - h_{\boldsymbol{\theta}}(x_i))^2 + \alpha\sum_{i=0}^{p}\theta_i^2$$

Results for $p = 15$ and varying α

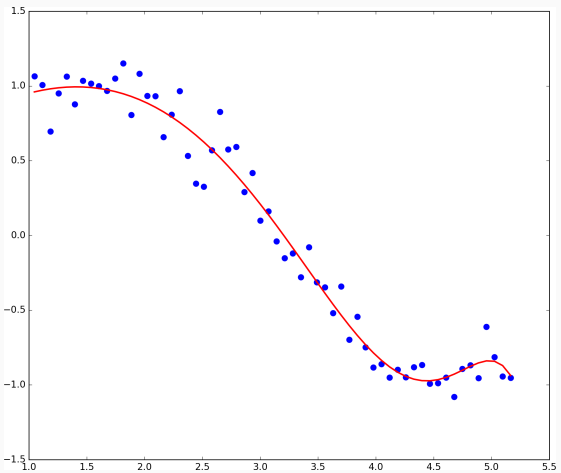
$\alpha = 0$



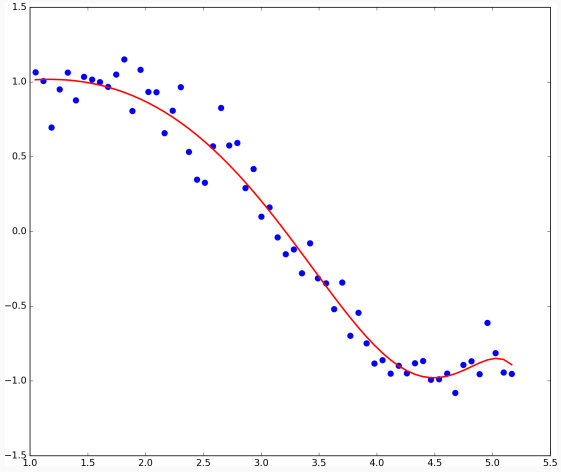
$\alpha = 1e-15$



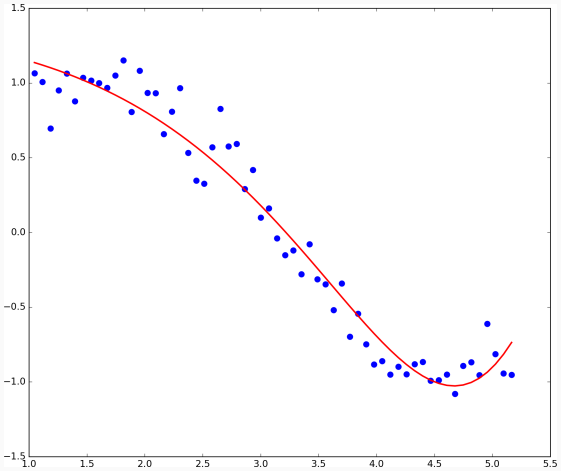
$\alpha = 1e-4$



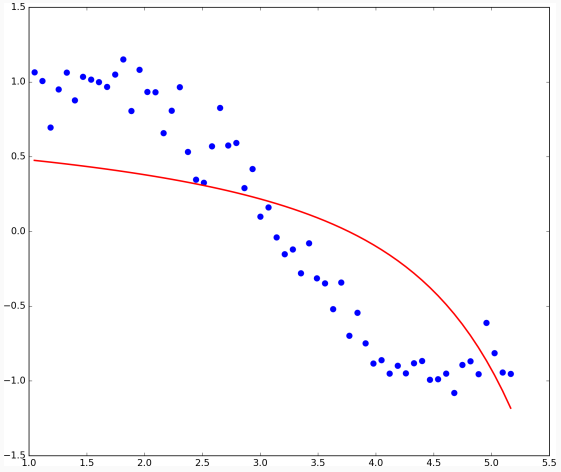
$\alpha = 1e-3$



$\alpha = 1e-2$

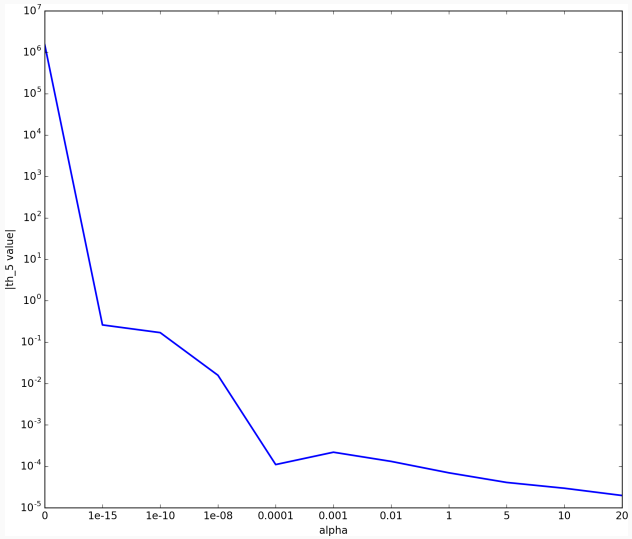


$\alpha = 5$



Values of coefficients

	rmse	th_0	th_1	th_2	th_3	th_4	th_5	th_6	th_7	th_8
alpha_0	+1.17e-02	-3.62e+04	+2.44e+05	-7.46e+05	+1.38e+06	-1.71e+06	+1.53e+06	-1.00e+06	+4.98e+05	-1.88e+05
alpha_1e-15	+1.46e-02	+9.43e+01	-2.98e+02	+3.79e+02	-2.37e+02	+6.72e+01	-2.60e-01	-4.35e+00	+5.62e-01	+1.42e-01
alpha_1e-10	+1.54e-02	+1.12e+01	-2.90e+01	+3.11e+01	-1.52e+01	+2.89e+00	+1.69e-01	-9.10e-02	-1.08e-02	+1.98e-03
alpha_1e-08	+1.58e-02	+1.34e+00	-1.53e+00	+1.75e+00	-6.80e-01	+3.88e-02	+1.58e-02	+1.59e-04	-3.60e-04	-5.37e-05
alpha_0.0001	+1.60e-02	+5.61e-01	+5.47e-01	-1.28e-01	-2.57e-02	-2.82e-03	-1.10e-04	+4.06e-05	+1.52e-05	+3.65e-06
alpha_0.001	+1.67e-02	+8.18e-01	+3.05e-01	-8.67e-02	-2.05e-02	-2.84e-03	-2.19e-04	+1.81e-05	+1.24e-05	+3.43e-06
alpha_0.01	+2.39e-02	+1.30e+00	-8.84e-02	-5.15e-02	-1.01e-02	-1.41e-03	-1.32e-04	+7.23e-07	+4.14e-06	+1.30e-06
alpha_1	+9.41e-02	+9.69e-01	-1.39e-01	-1.93e-02	-3.00e-03	-4.66e-04	-6.97e-05	-9.90e-06	-1.29e-06	-1.43e-07
alpha_5	+2.31e-01	+5.48e-01	-5.89e-02	-8.52e-03	-1.42e-03	-2.41e-04	-4.08e-05	-6.87e-06	-1.15e-06	-1.91e-07
alpha_10	+3.00e-01	+4.00e-01	-3.72e-02	-5.53e-03	-9.50e-04	-1.67e-04	-2.96e-05	-5.23e-06	-9.25e-07	-1.63e-07
alpha_20	+3.79e-01	+2.77e-01	-2.25e-02	-3.40e-03	-5.99e-04	-1.08e-04	-1.97e-05	-3.60e-06	-6.58e-07	-1.20e-07



Value of θ_5

Determination of the parameters in the ridge regression

Considering the cost function J:

$$J(\boldsymbol{\theta}) = J_{lms}(\boldsymbol{\theta}) + \alpha \sum_{i=0}^{p} \theta_i^2$$

In a gradient algorithm, update of the parameters:

$\theta_i^{k+1} = \theta_i^k - \nu.(\frac{\partial J_{lms}}{D\theta_i} - 2\alpha\theta_i^k)$

So the update rule is:

$$\theta_i^{k+1} = \theta_i^k(1 - 2\nu\alpha) - \Delta_{lms}$$

where Δ_{lms} is the update in case of non-regularized regression

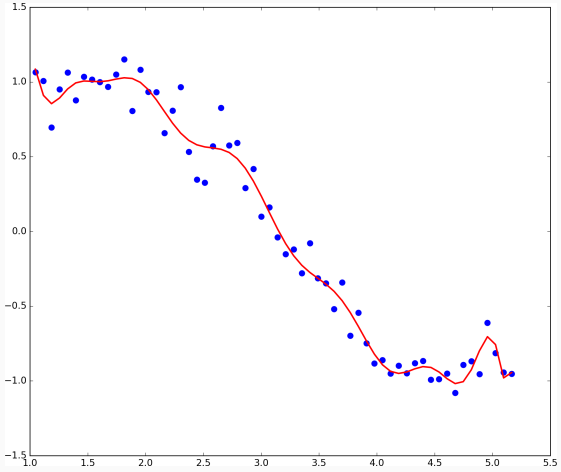
Lasso regression (L1)

Lasso regression is a linear regression with a Lasso regularization:

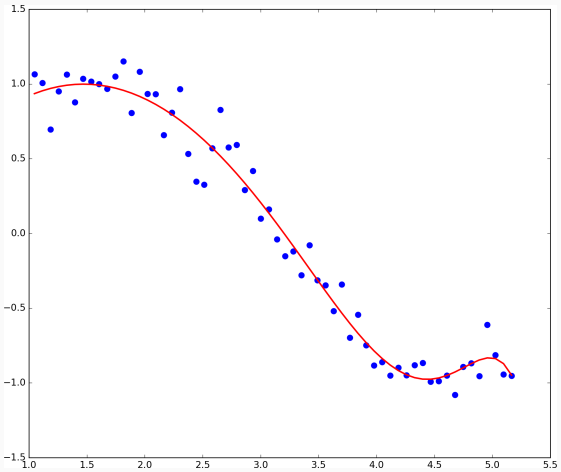
$$J(\boldsymbol{\theta}) = \frac{1}{n}\sum(y_i - h_{\boldsymbol{\theta}}(x_i))^2 + \alpha\sum_{i=0}^{p}|\theta_i|$$

Results for $p = 15$ and varying α

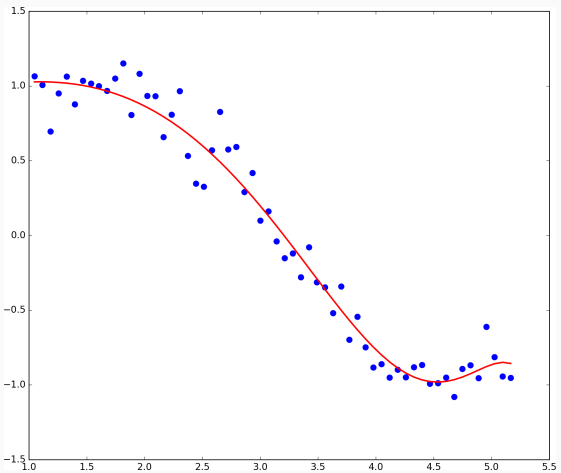
$\alpha = 0$



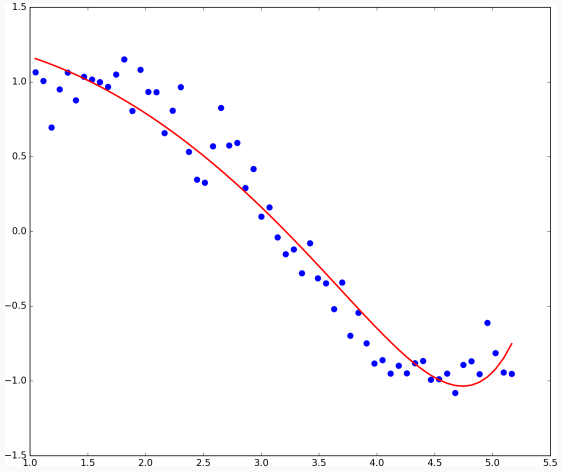
$\alpha = 1e-15$



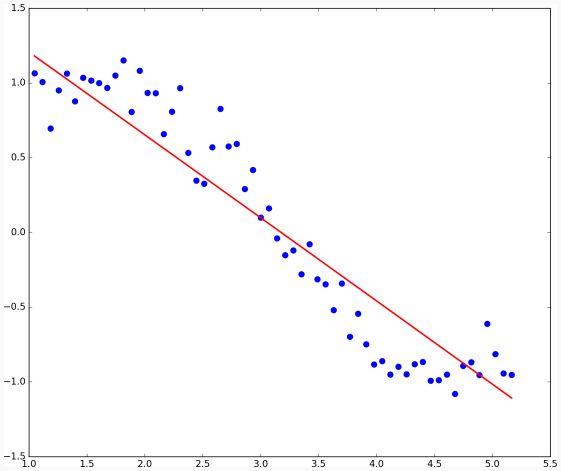
$\alpha = 1e-4$



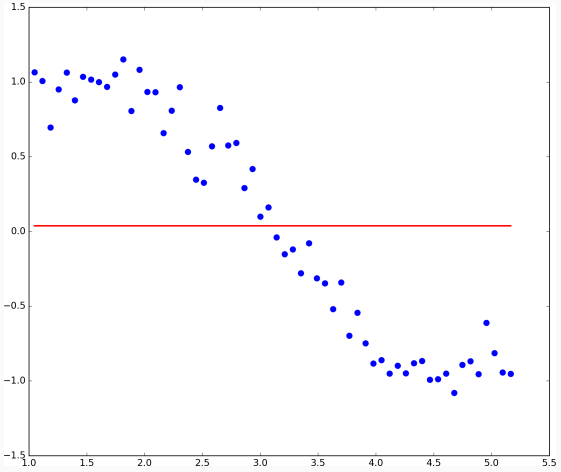
$\alpha = 1e-3$



$\alpha = 1e-2$

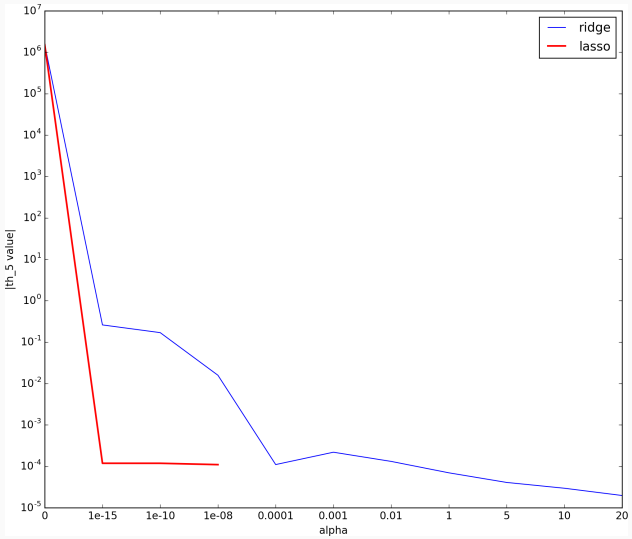


$\alpha = 5$



Values of coefficients

	rmse	th_0	th_1	th_2	th_3	th_4	th_5	th_6	th_7	th_8
alpha_0	+1.17e-02	-3.62e+04	+2.44e+05	-7.46e+05	+1.38e+06	-1.71e+06	+1.53e+06	-1.00e+06	+4.98e+05	-1.88e+05
alpha_1e-15	+1.59e-02	+2.22e-01	+1.06e+00	-3.69e-01	+8.85e-04	+1.63e-03	-1.19e-04	-6.44e-05	-6.28e-06	+1.45e-06
alpha_1e-10	+1.59e-02	+2.22e-01	+1.06e+00	-3.69e-01	+8.84e-04	+1.63e-03	-1.18e-04	-6.44e-05	-6.28e-06	+1.45e-06
alpha_1e-08	+1.59e-02	+2.22e-01	+1.06e+00	-3.69e-01	+7.69e-04	+1.62e-03	-1.10e-04	-6.45e-05	-6.32e-06	+1.43e-06
alpha_0.0001	+1.72e-02	+9.03e-01	+1.71e-01	-0.00e+00	-4.78e-02	-0.00e+00	-0.00e+00	+0.00e+00	+0.00e+00	+9.47e-06
alpha_0.001	+2.80e-02	+1.29e+00	-0.00e+00	-1.26e-01	-0.00e+00	-0.00e+00	-0.00e+00	+0.00e+00	+0.00e+00	+0.00e+00
alpha_0.01	+6.07e-02	+1.76e+00	-5.52e-01	-5.62e-04	-0.00e+00	-0.00e+00	-0.00e+00	-0.00e+00	-0.00e+00	-0.00e+00
alpha_1	+6.16e-01	+3.80e-02	-0.00e+00	-0.00e+00	-0.00e+00	-0.00e+00	-0.00e+00	-0.00e+00	-0.00e+00	-0.00e+00
alpha_5	+6.16e-01	+3.80e-02	-0.00e+00	-0.00e+00	-0.00e+00	-0.00e+00	-0.00e+00	-0.00e+00	-0.00e+00	-0.00e+00
alpha_10	+6.16e-01	+3.80e-02	-0.00e+00	-0.00e+00	-0.00e+00	-0.00e+00	-0.00e+00	-0.00e+00	-0.00e+00	-0.00e+00
alpha_20	+6.16e-01	+3.80e-02	-0.00e+00	-0.00e+00	-0.00e+00	-0.00e+00	-0.00e+00	-0.00e+00	-0.00e+00	-0.00e+00



Value of θ_5

Determination of the parameters in the lasso regression

Considering the cost function J:

$$J(\boldsymbol{\theta}) = J_{lms}(\boldsymbol{\theta}) + \alpha \sum_{i=0}^{p} |\theta_i|$$

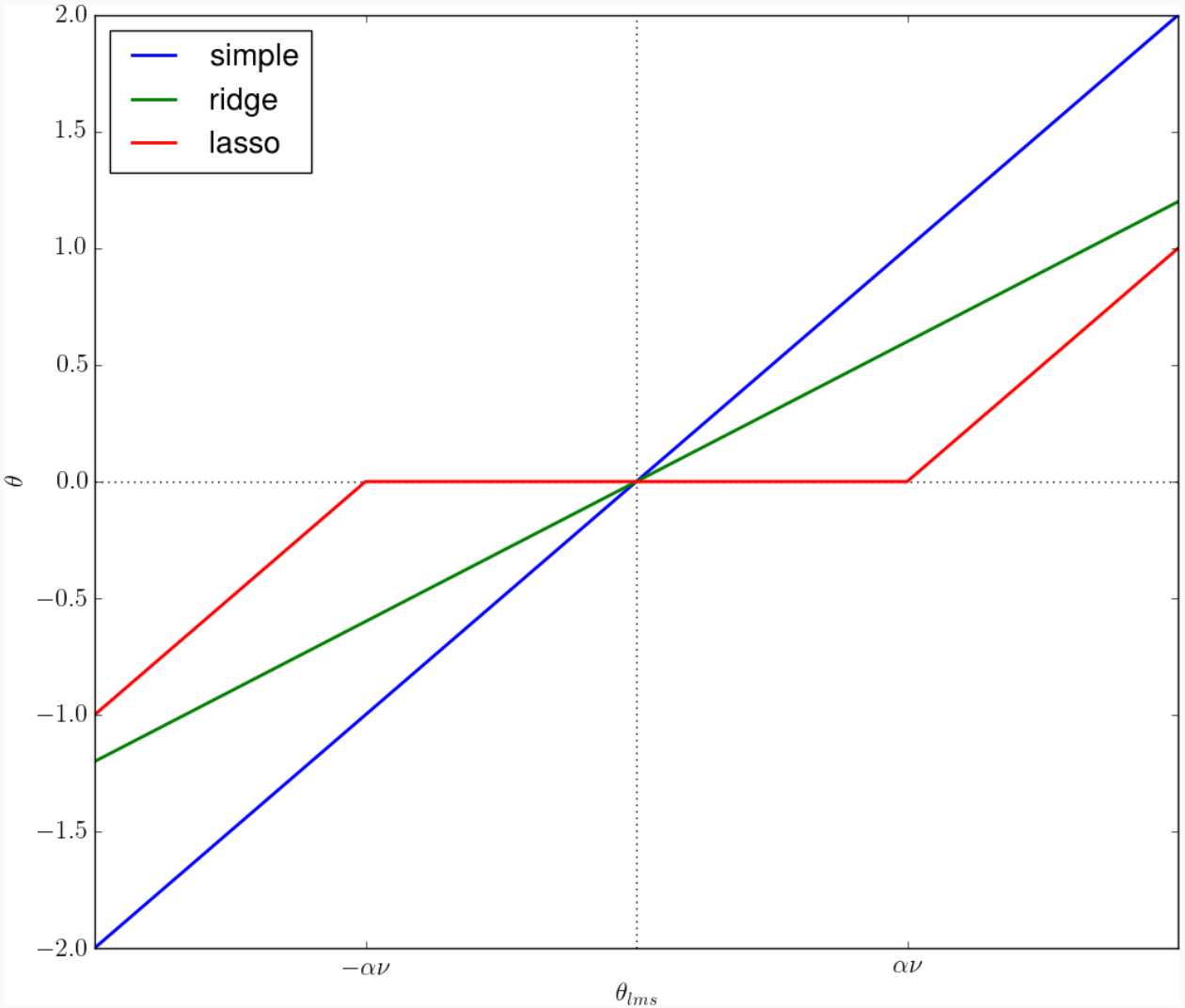
In a gradient algorithm, update of the parameters would be:

J is not differentiable. If we consider $\theta_{lms} = \theta_i^k - \nu . \frac{\partial J_{lms}}{D\theta_i}$

The update rule is:

- $\theta_i^{k+1} = \theta_{lms} + \alpha\nu$ if $\theta_{lms} < -\alpha\nu$
- $\theta_i^k = 0$ if $-\alpha\nu < \theta_{lms} < \alpha\nu$
- $\theta_i^{k+1} = \theta_{lms} - \alpha\nu$ if $\theta_{lms} > \alpha\nu$

Summary of parameters updates



Comparison Ridge/Lasso

Ridge (L2)

- Prevents the overfitting
- includes all the features (dimensions) of the predictor, so it can be useless for high dimensional predictors

Comparison Ridge/Lasso

Ridge (L2)

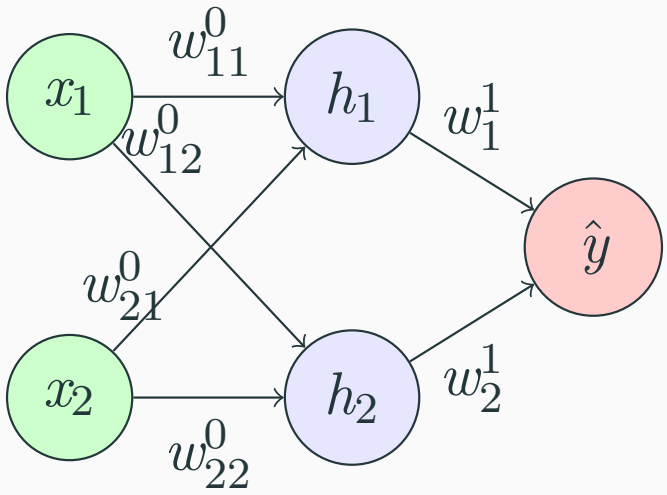
- Prevents the overfitting
- includes all the features (dimensions) of the predictor, so it can be useless for high dimensional predictors

Lasso (L1)

- Provides sparse solutions and reduce the dimension of the predictor
- If some features in the predictor are correlated, arbitrarily select one from the others.

In a neural network

L2 (Ridge) and L1 (Lasso) regularization are widely use in Neural Network architectures.



- Non-regularized loss function: $L(\mathbf{w}) = ||\hat{y}(\mathbf{w}) - y||^2$.
- L2-regularized loss function:
  $L(\mathbf{w}) = ||\hat{y}(\mathbf{w}) - y||^2 + \alpha \sum_{i=0}^{p} |w_i| \ .$
- L1-regularized loss function:
  $L(\mathbf{w}) = ||\hat{y}(\mathbf{w}) - y||^2 + \alpha \sum_{i=0}^{p} w_i^2.$

Wrapping-up

Advantage

L1/L2 regularization prevents overfitting even for large architecture (a big weight vector to optimize)

Wrapping-up

Advantage

L1/L2 regularization prevents overfitting even for large architecture (a big weight vector to optimize)

But...

It comes with extra hyperparameter to tune (using, e.g., cross-validation): α

Questions addressed in this lecture

- What is the principle of the Random Forests? [Van16,5.8]
- How to determine the hyperparamters? [Van16,5.3]
- What is a artificial neural network? [GBC16,6]
- What is a convolutive layers [GBC16,9]
- What is L1(Lasso)/L2(Ridge) regularization? [Van16,5.6; GBC16,8] https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1
- What are the main types of neural networks? [GBC16,10, 14, 20]

Refs

[Van16,n]: Jake VanderPlas, *Python Data Science Handbook*, section n
[GBC16,n]: Goodfellow etal., Deep Learning, chapter n