Machine learning and physical modelling-2

julien.brajard@nersc.no

October 2022

NERSC

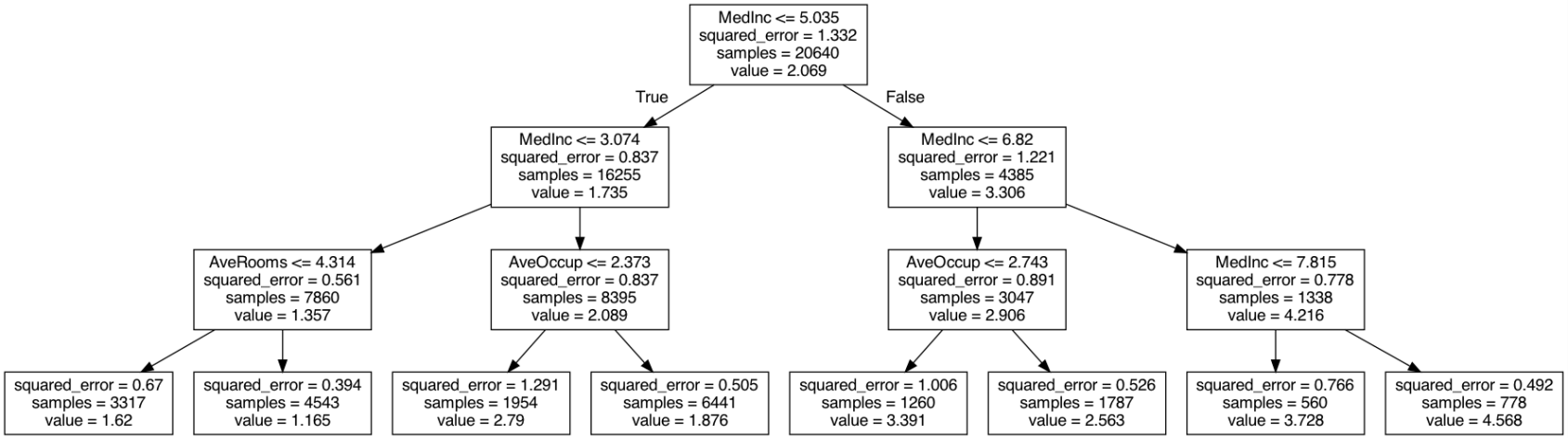
`https://github.com/brajard/MAT330`

Table of contents

A standard Machine learning model: Random Forests

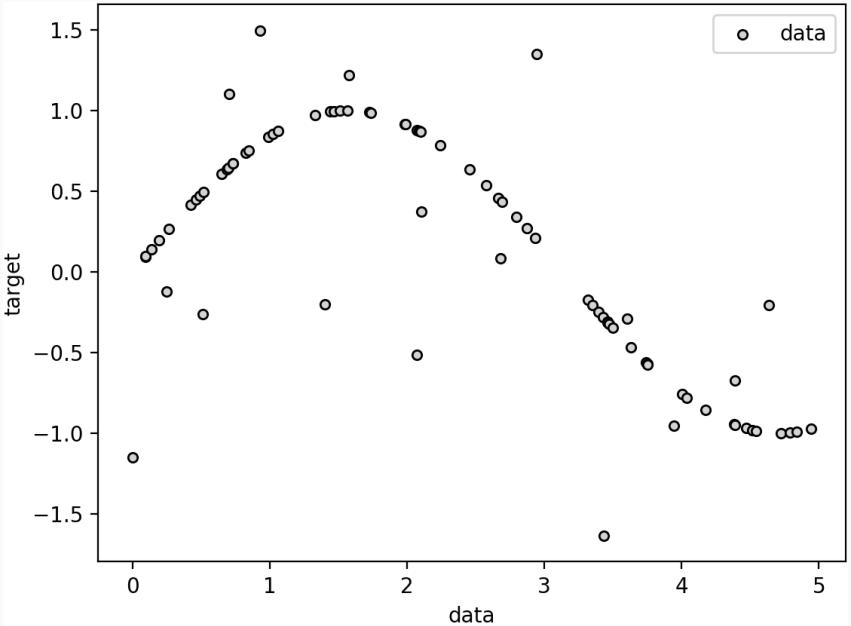
A decision tree

Predict house price (in $100,000) from 8 features:

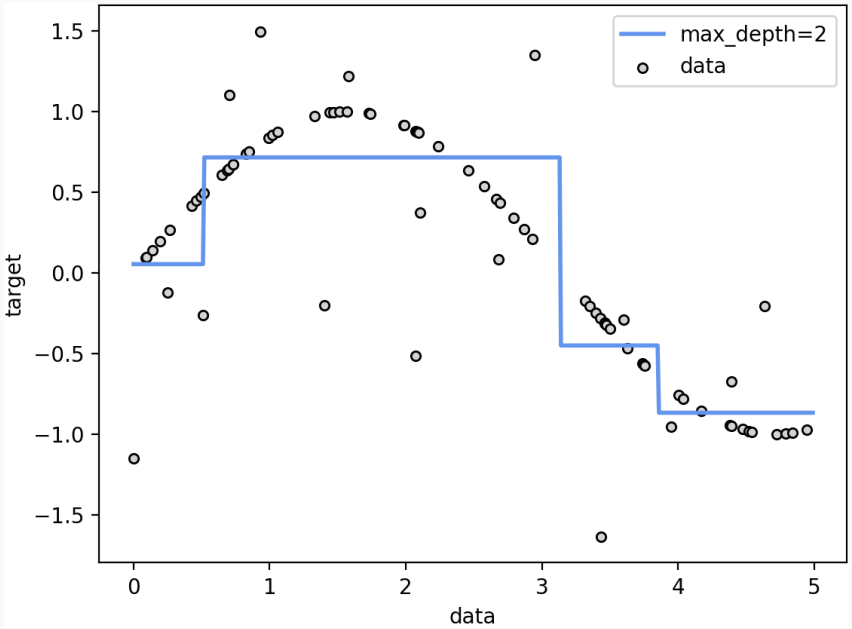


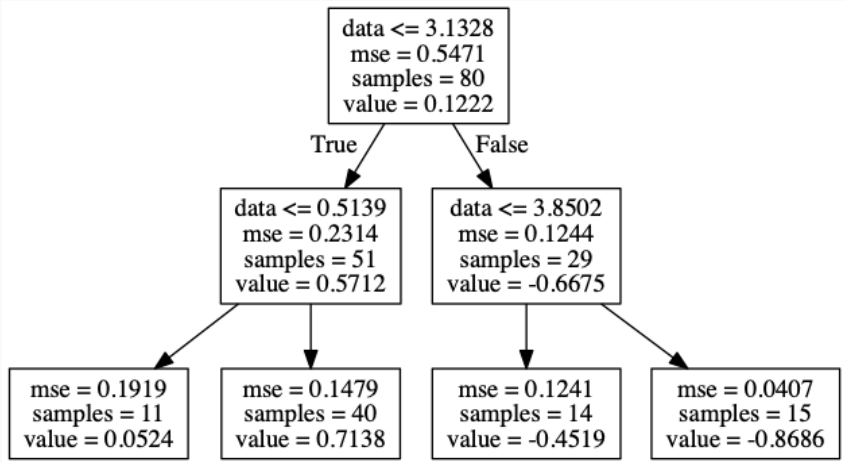
MedInc	median income in block group
AveRooms	average number of rooms per household
AveOccup	average number of household members
...	...

Uni-variate example

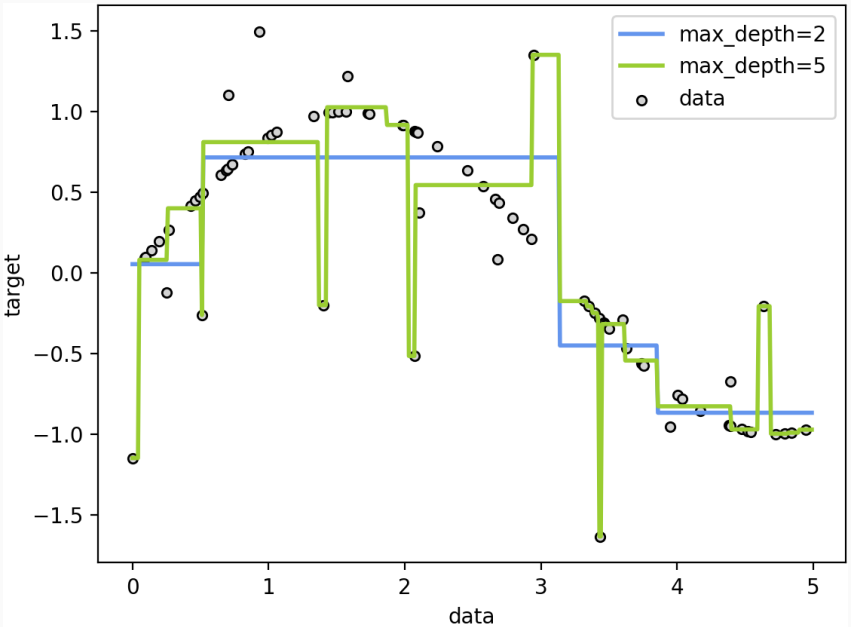


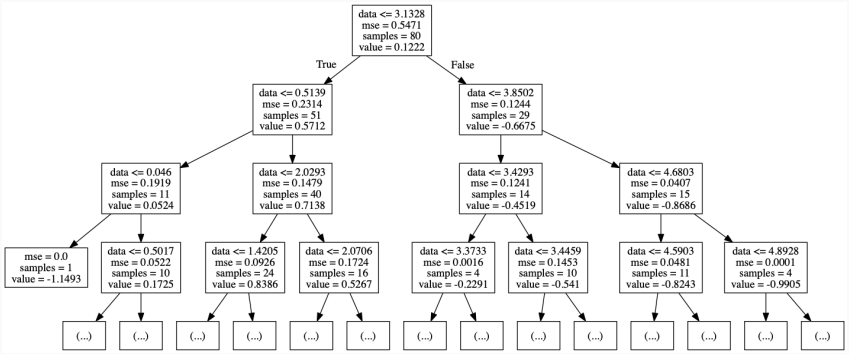
Uni-variate example





Uni-variate example





From tree to forest

Disadvantages of regression tree:

- Can overfit the data

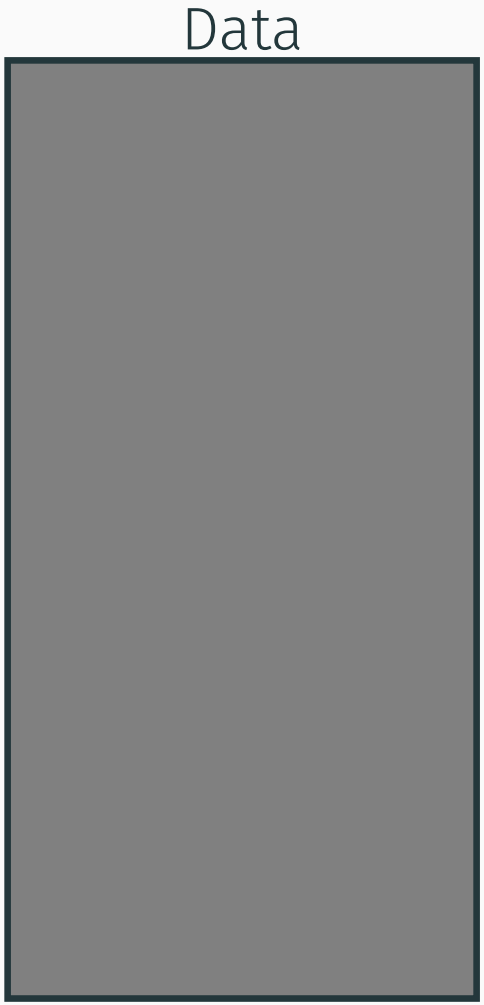
From tree to forest

Disadvantages of regression tree:

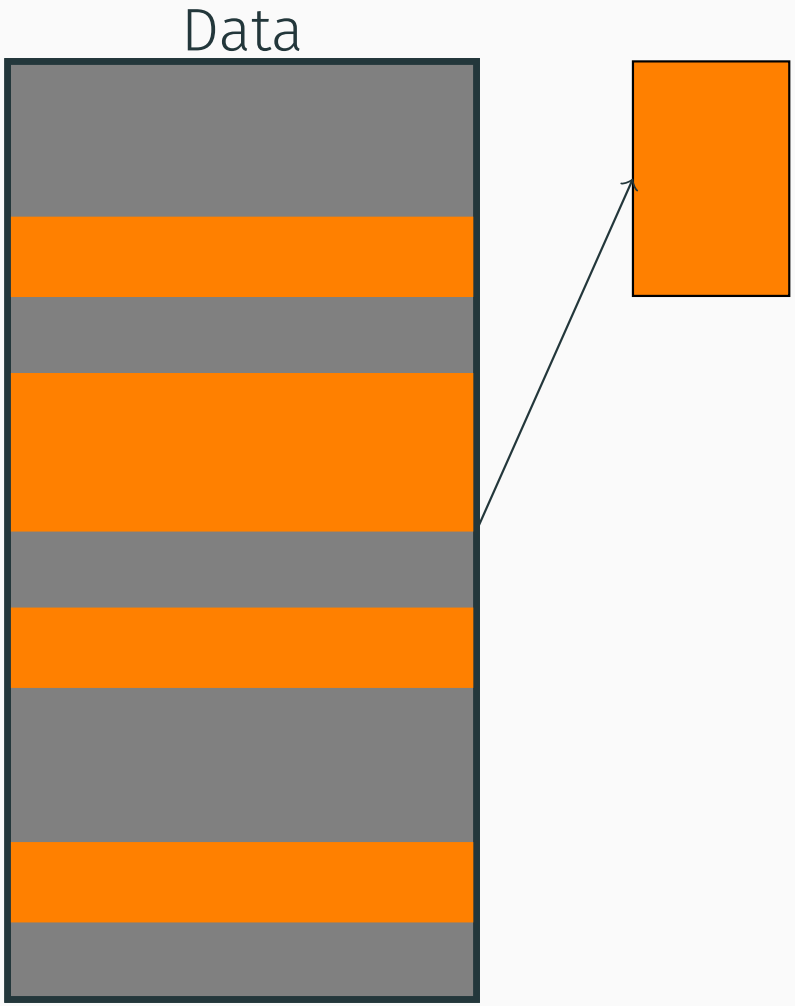
- Can overfit the data

One extension of Regression Tree: Random Forest

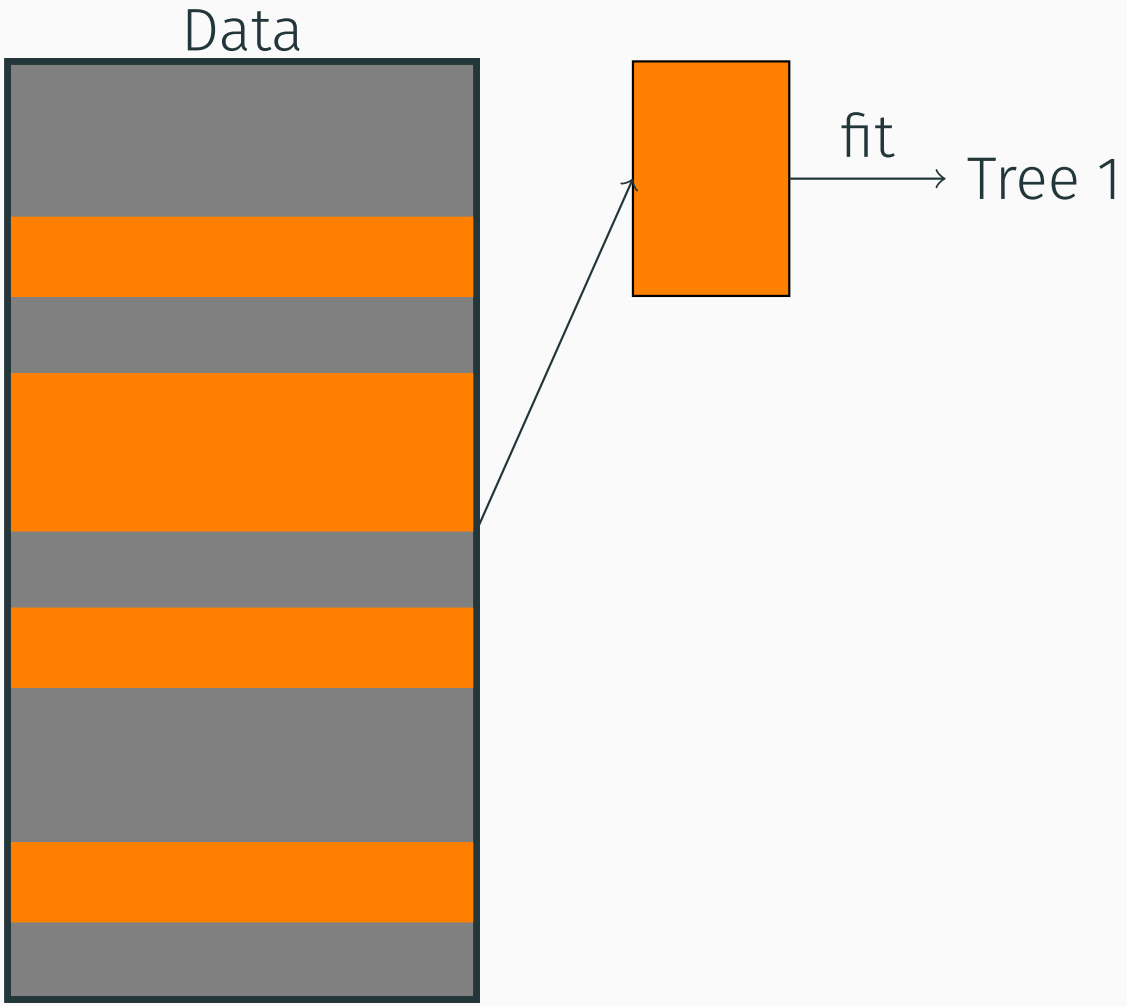
The (over simplified) principle of Random Forest



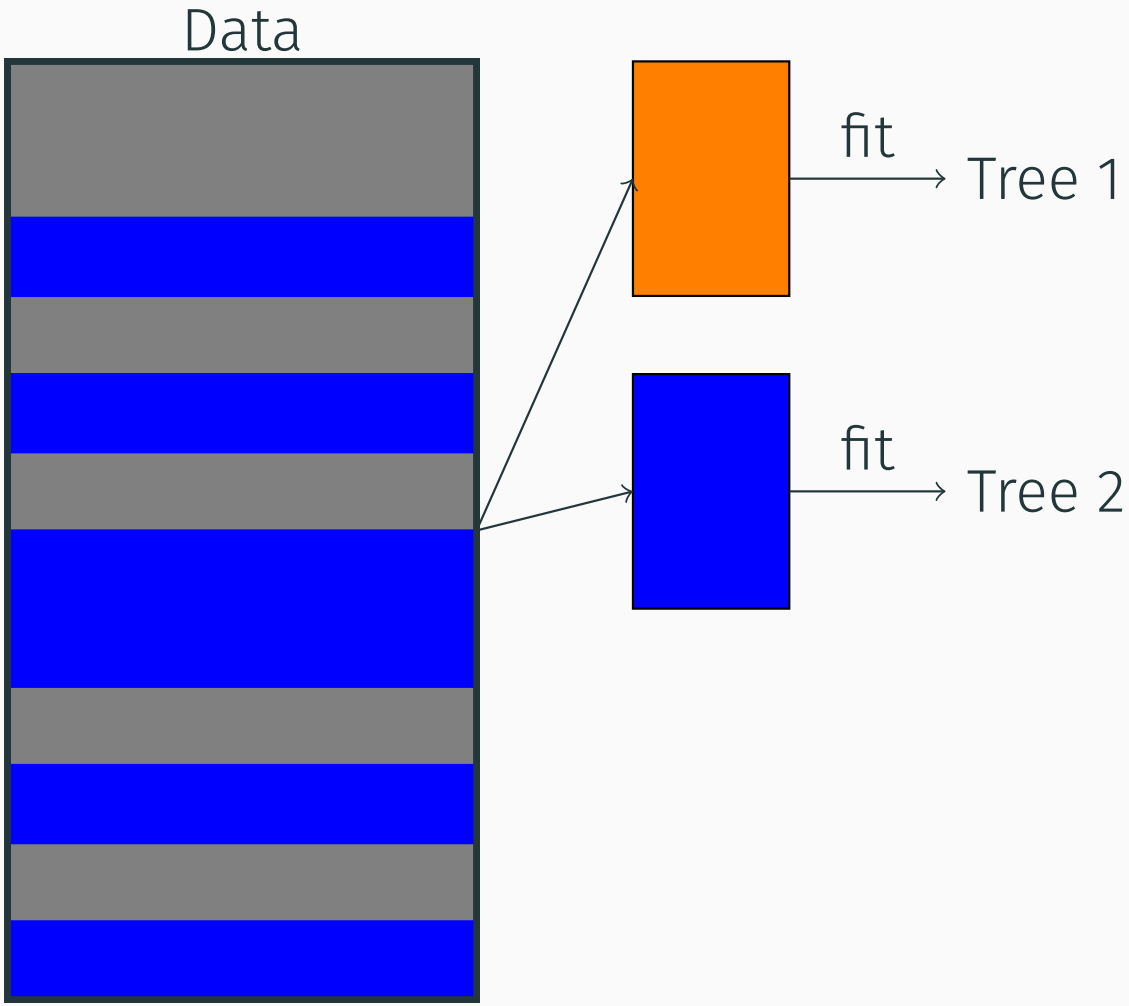
The (over simplified) principle of Random Forest



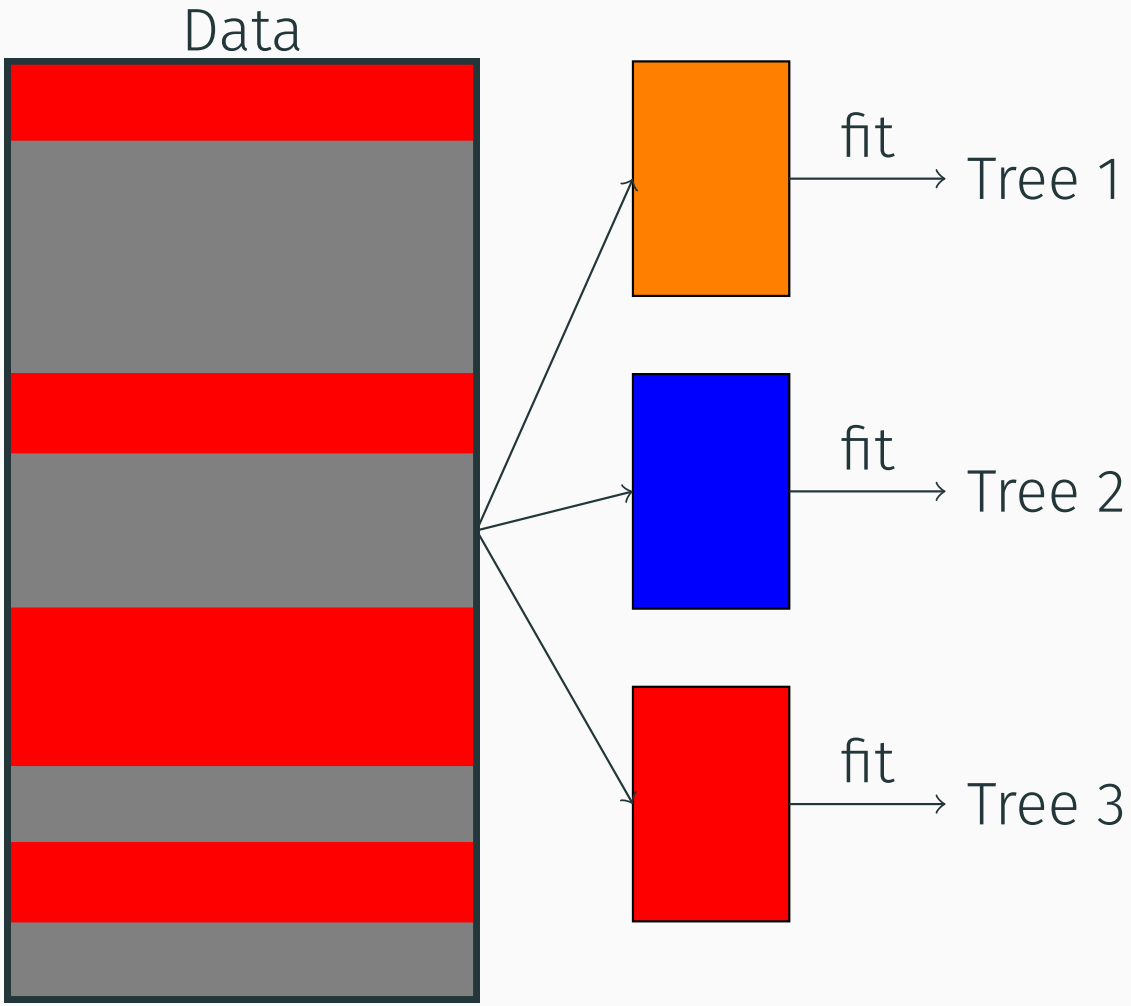
The (over simplified) principle of Random Forest



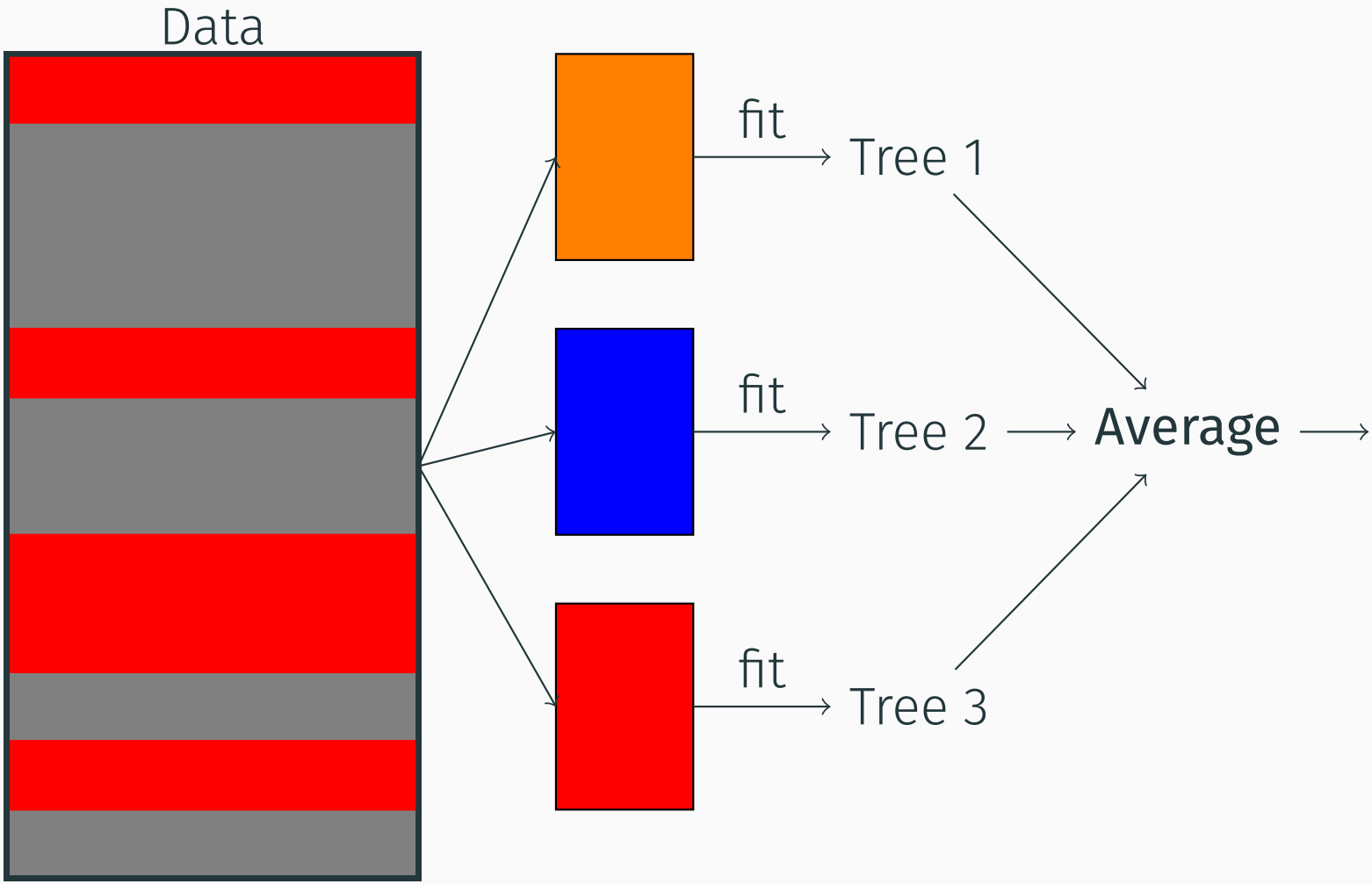
The (over simplified) principle of Random Forest



The (over simplified) principle of Random Forest

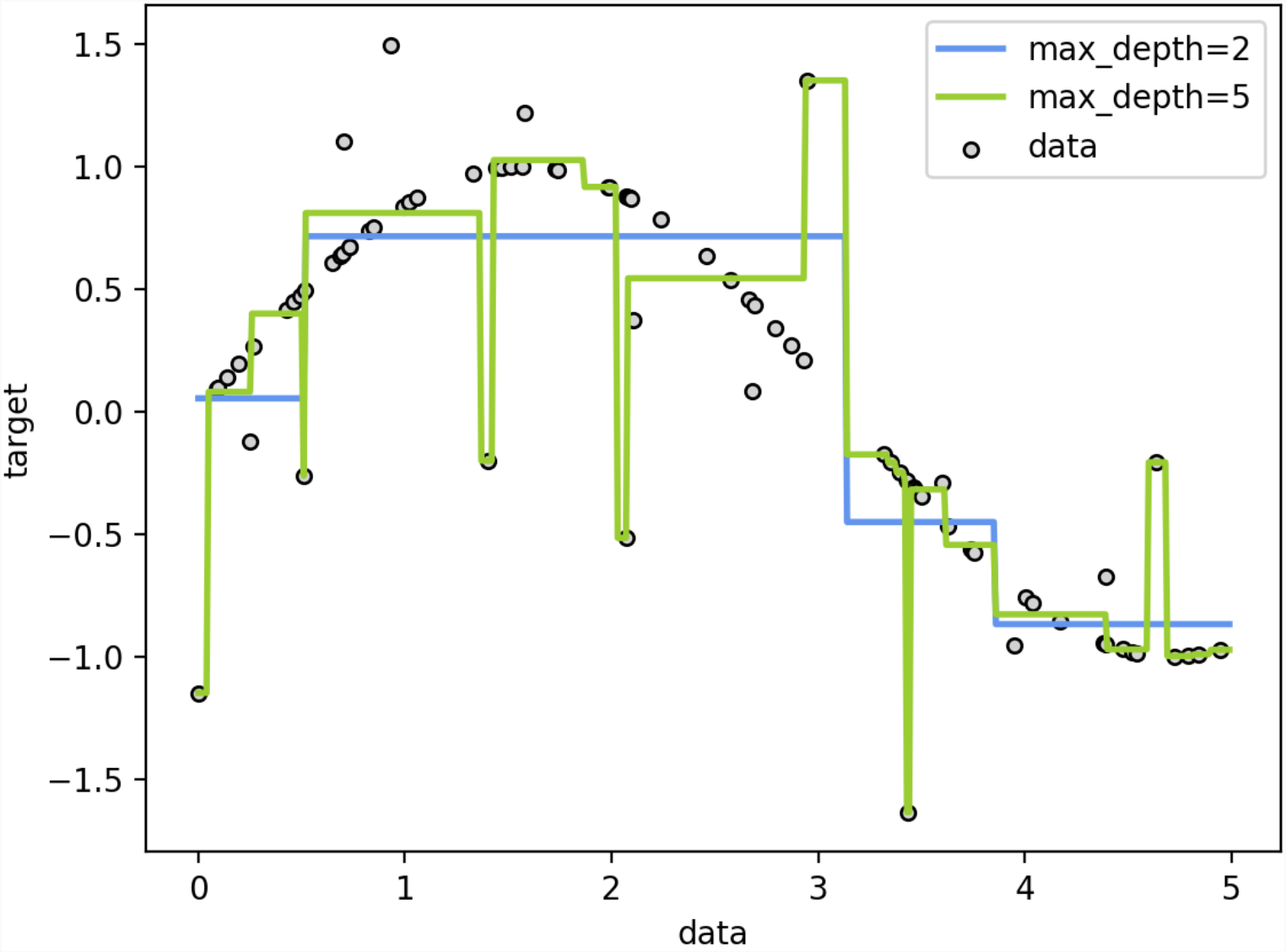


The (over simplified) principle of Random Forest



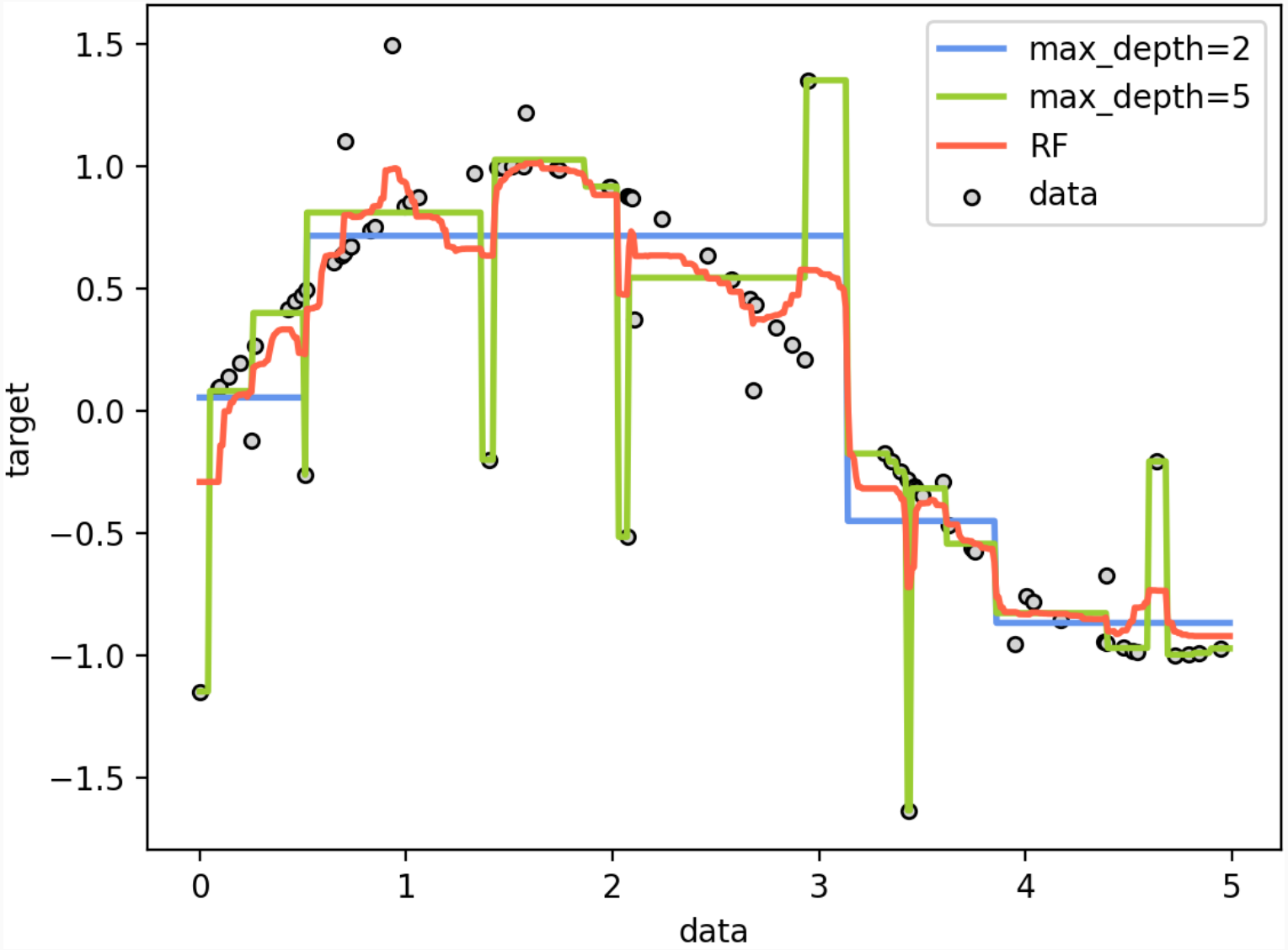
Results on the univariate experiment

Prediction of Randoms trees



Results on the univariate experiment

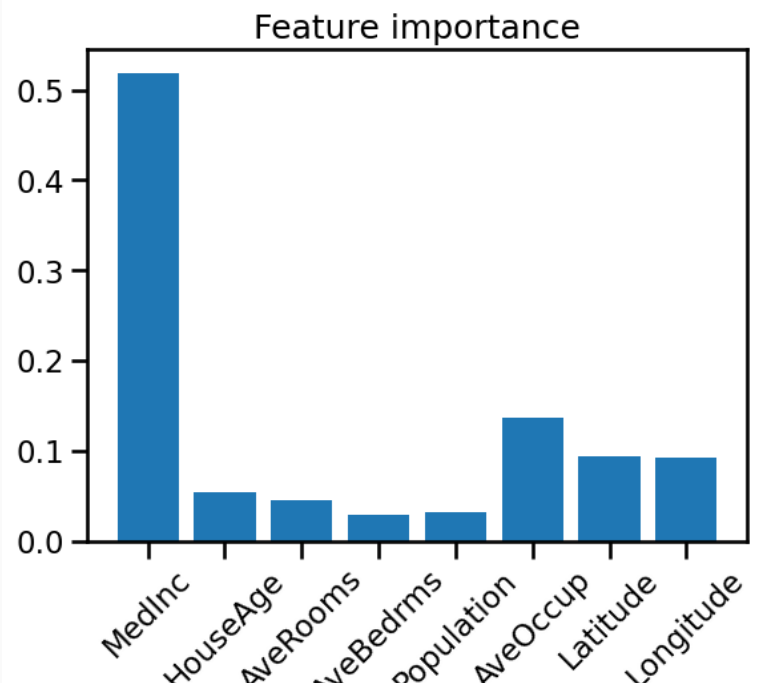
Prediction of a Random Forest



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.



MedInc	median income in block grou
AveRooms	average number of rooms household
AveOccup	average number of househ members
...	...

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`: minimum 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.

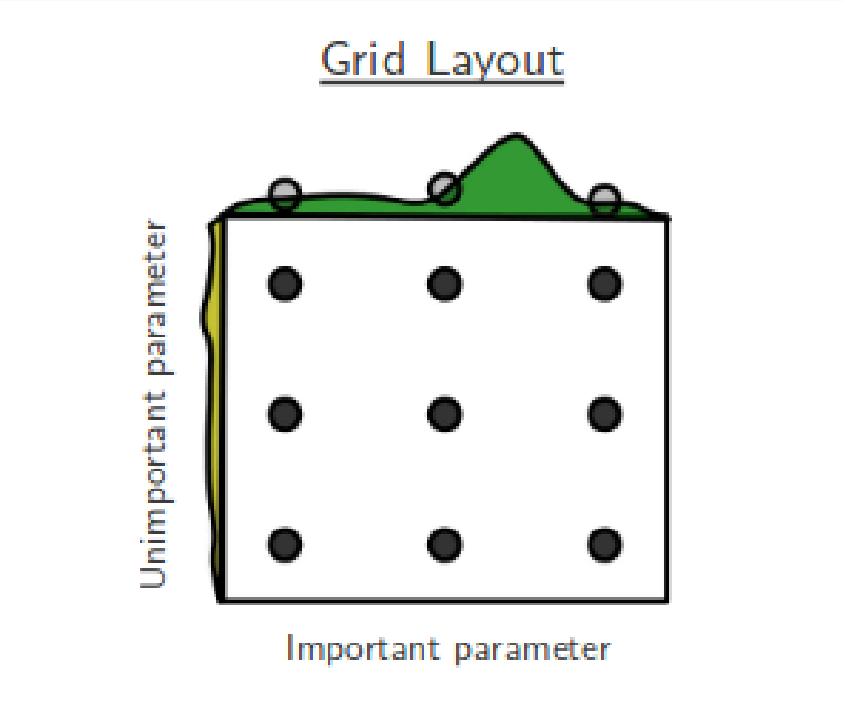
Determination of the hyperparameters

- 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

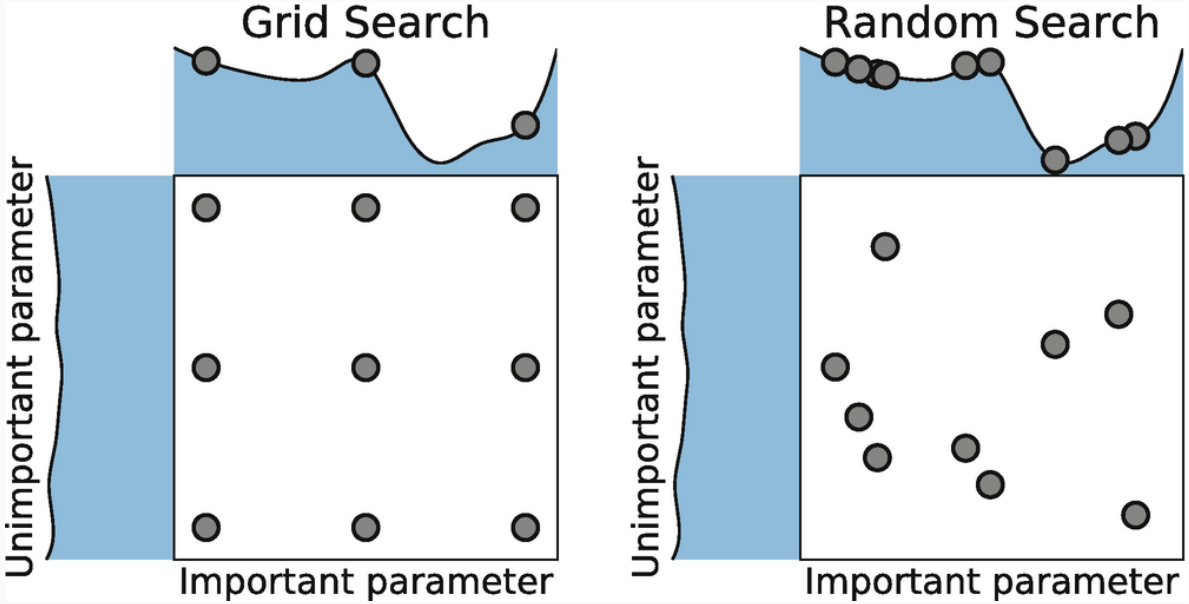
- It make an exhaustive search of the hyperparameters
- The procedure is easy to parallelized.
- it is not naturally adapted for quantitative hyperparameters.

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.

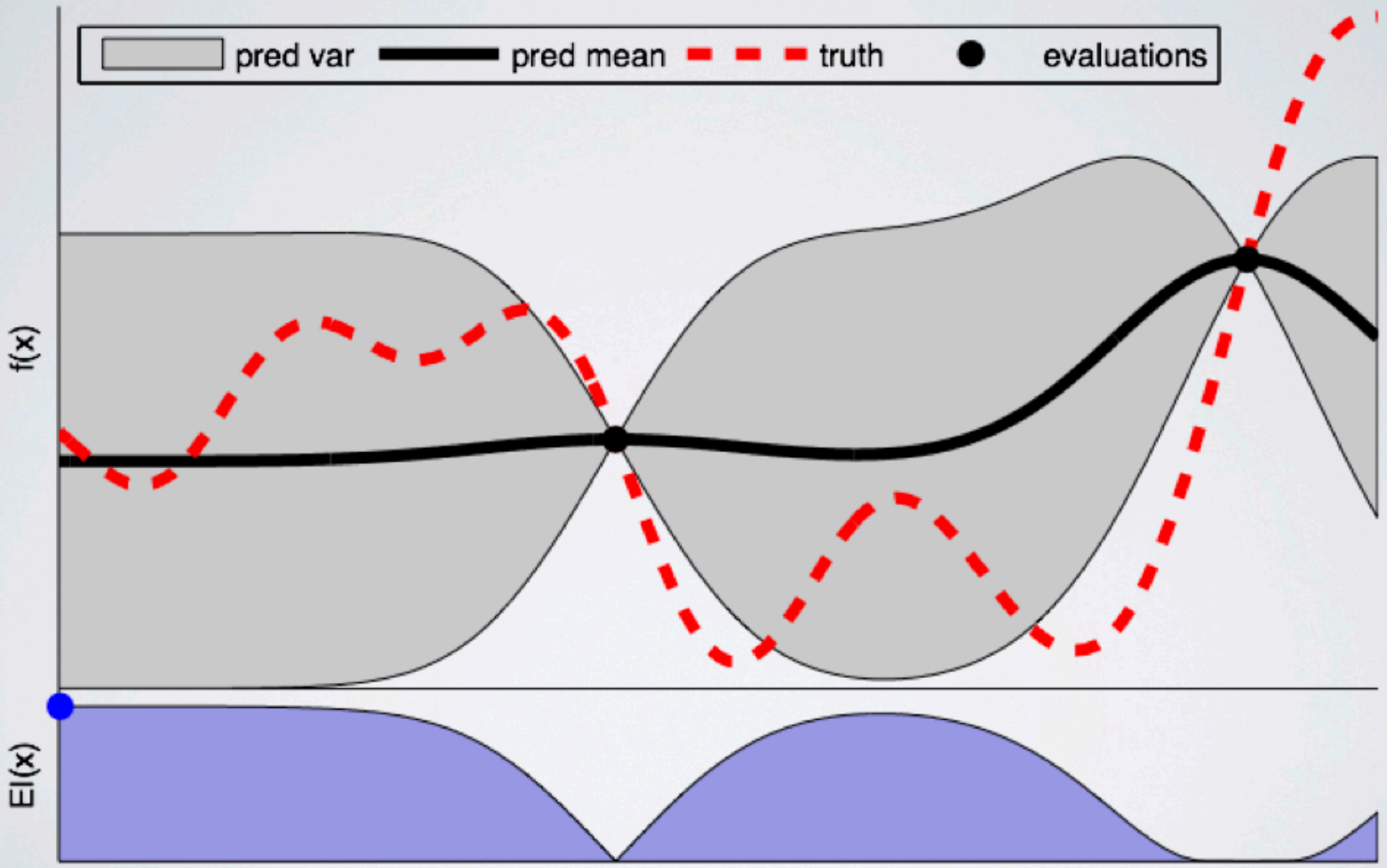
Bayesian search

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)

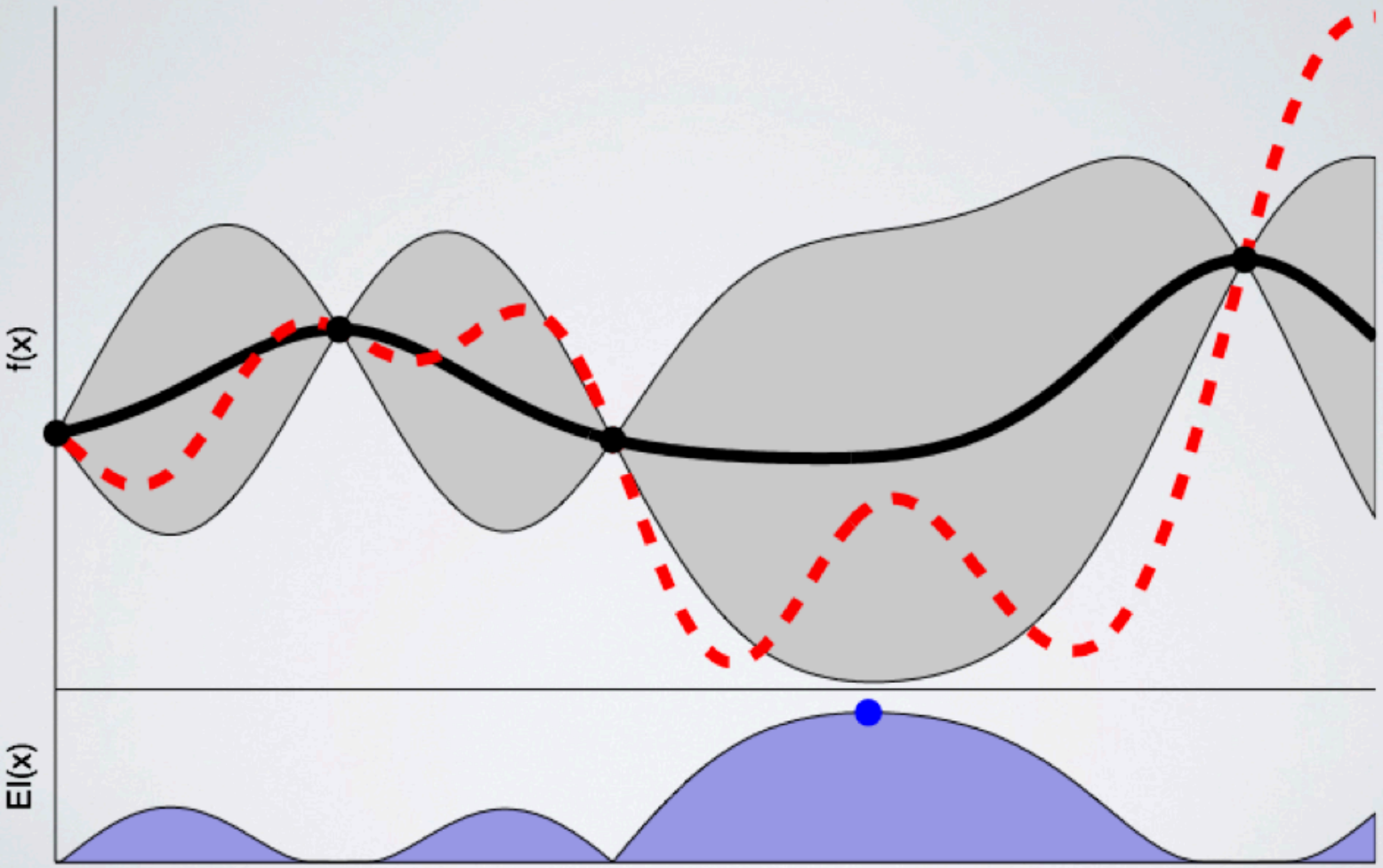
Illustration taken from Ryan P. Adams: ”A Tutorial on Bayesian Optimization for Machine Learning”

Illustrating Bayesian Optimization



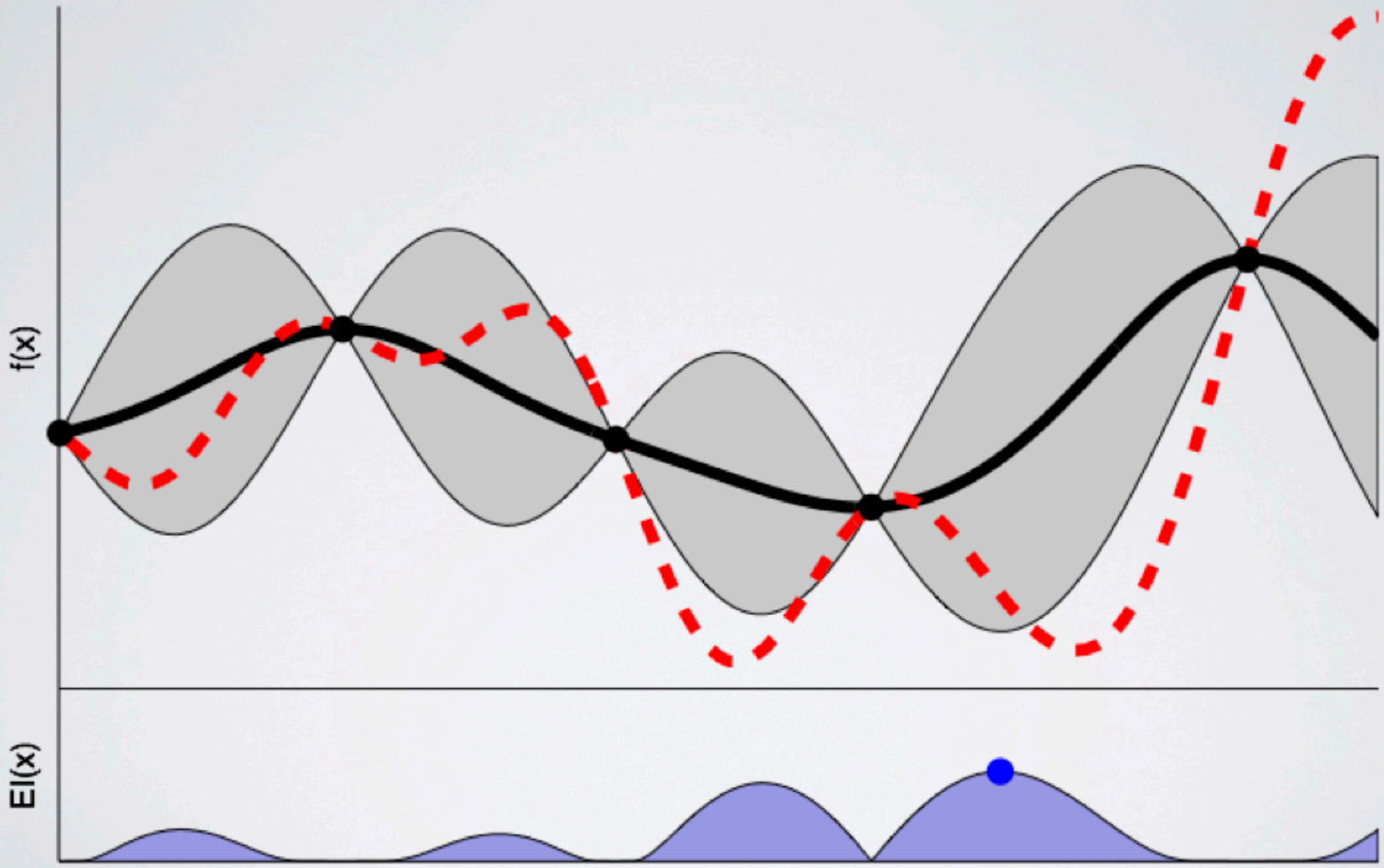
source: R.P. Adams

Illustrating Bayesian Optimization



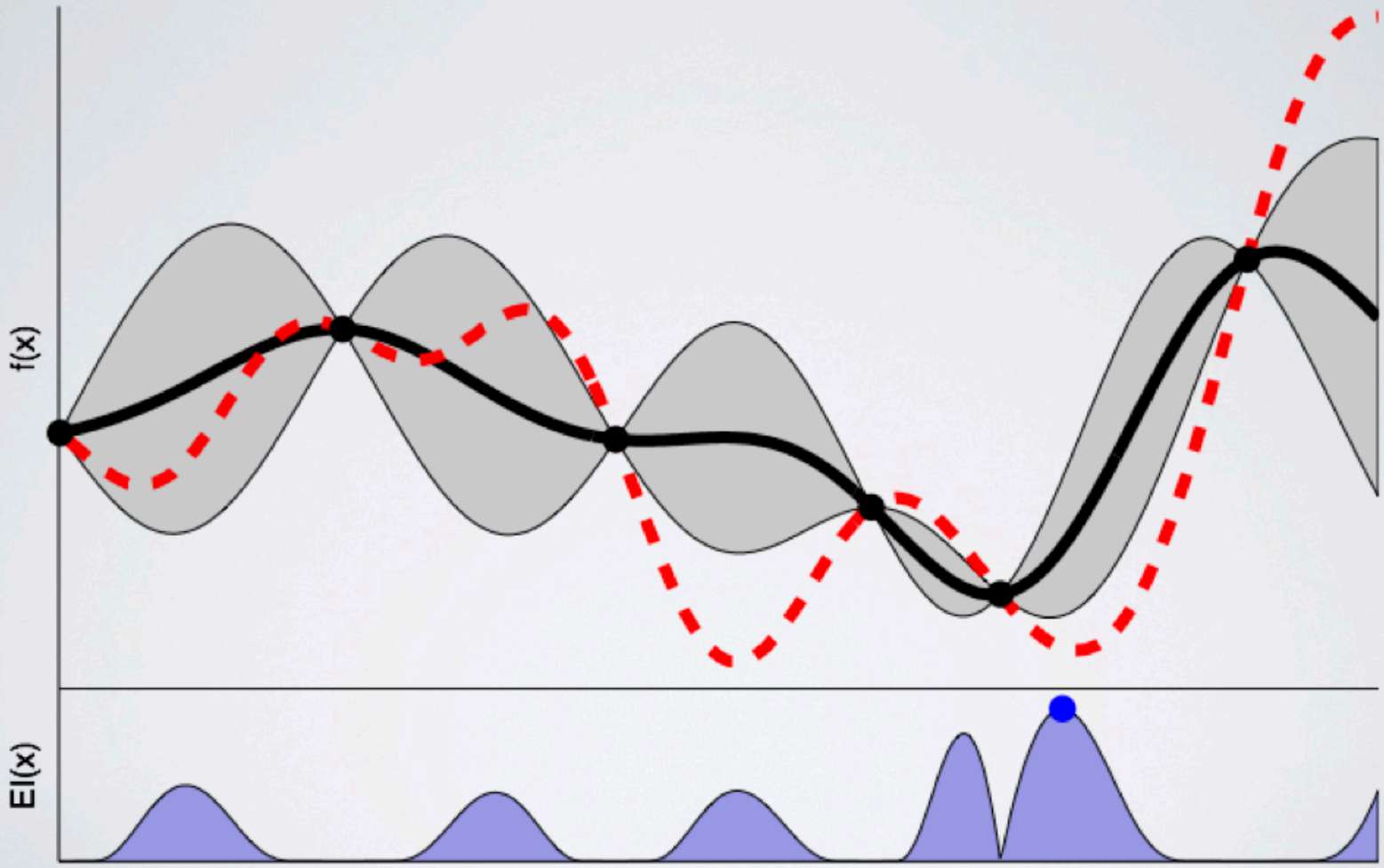
source: R.P. Adams

Illustrating Bayesian Optimization



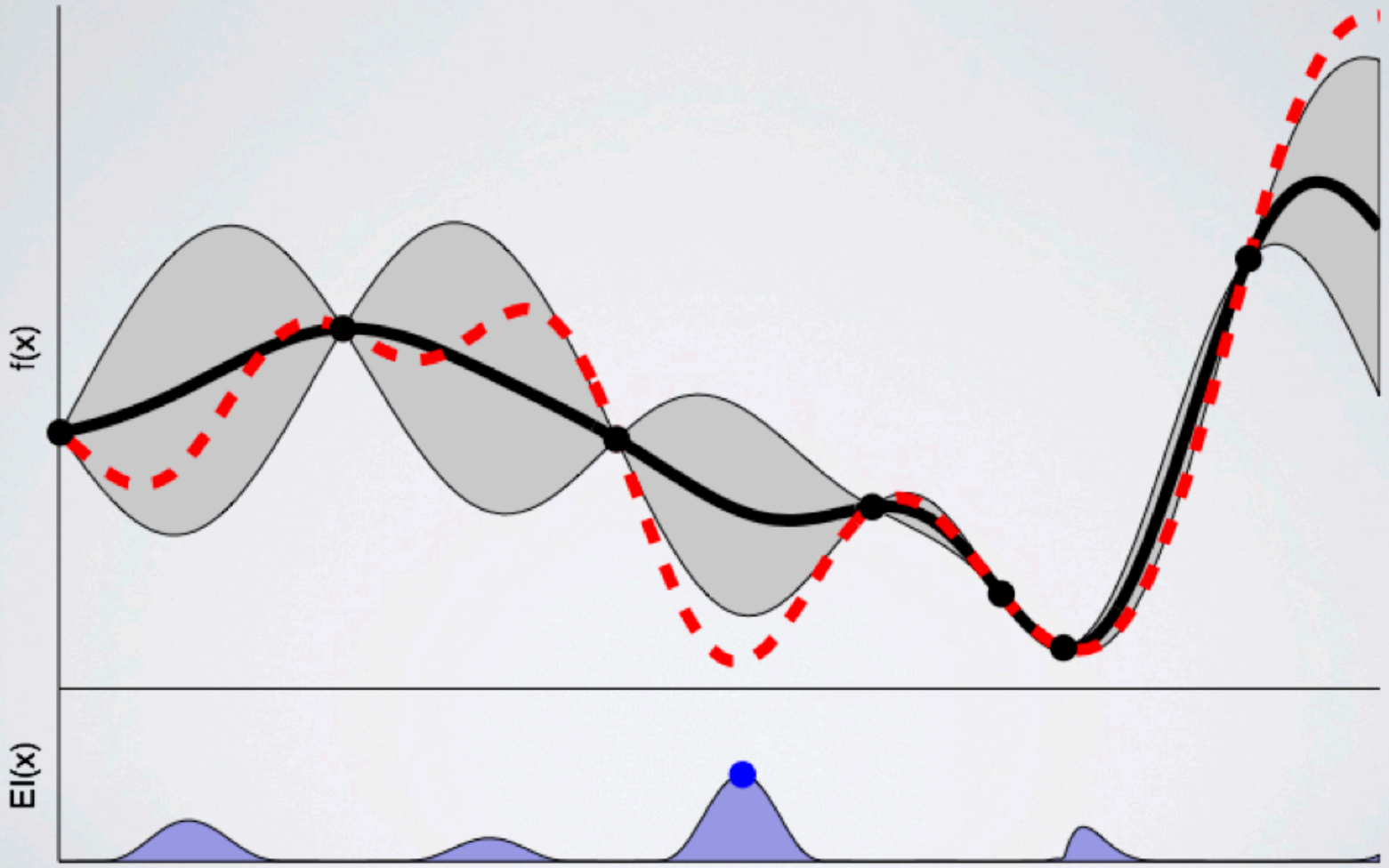
source: R.P. Adams

Illustrating Bayesian Optimization



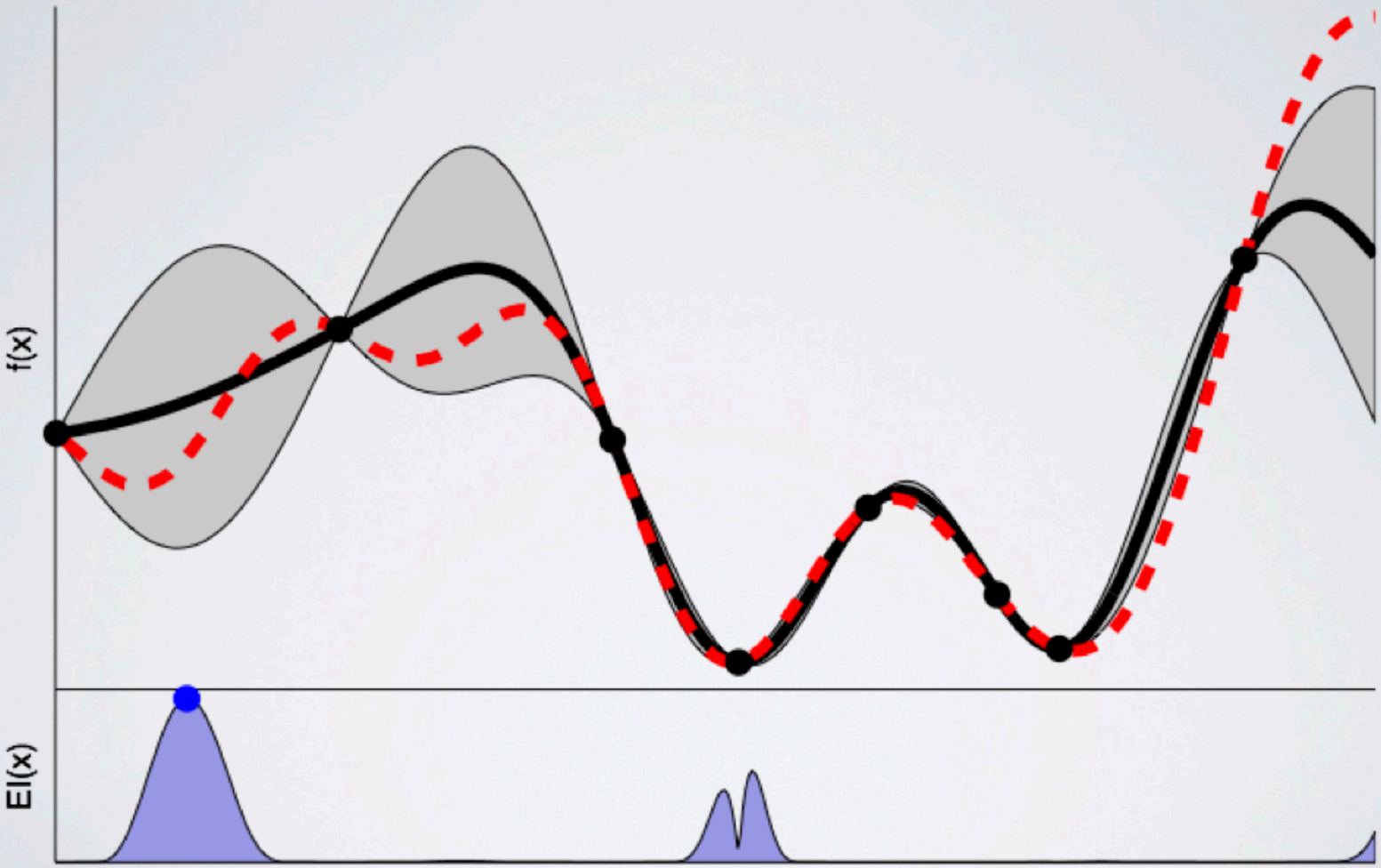
source: R.P. Adams

Illustrating Bayesian Optimization



source: R.P. Adams

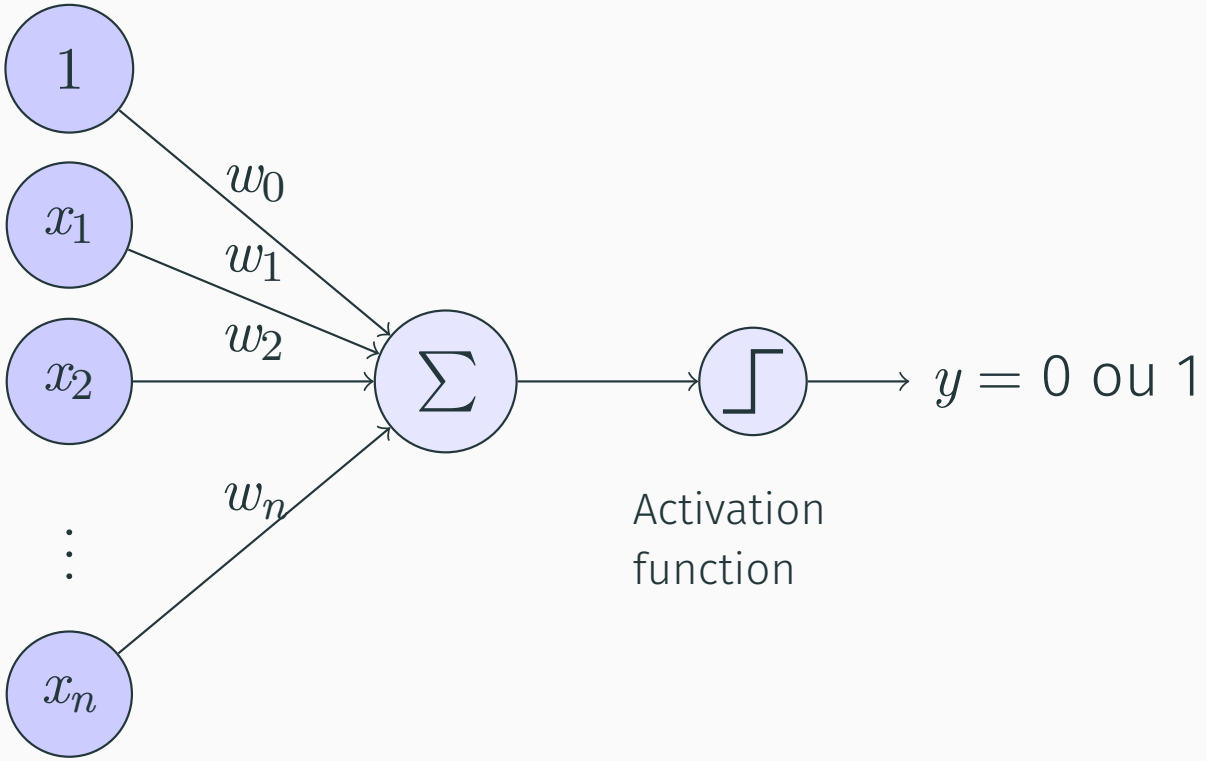
Illustrating Bayesian Optimization



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

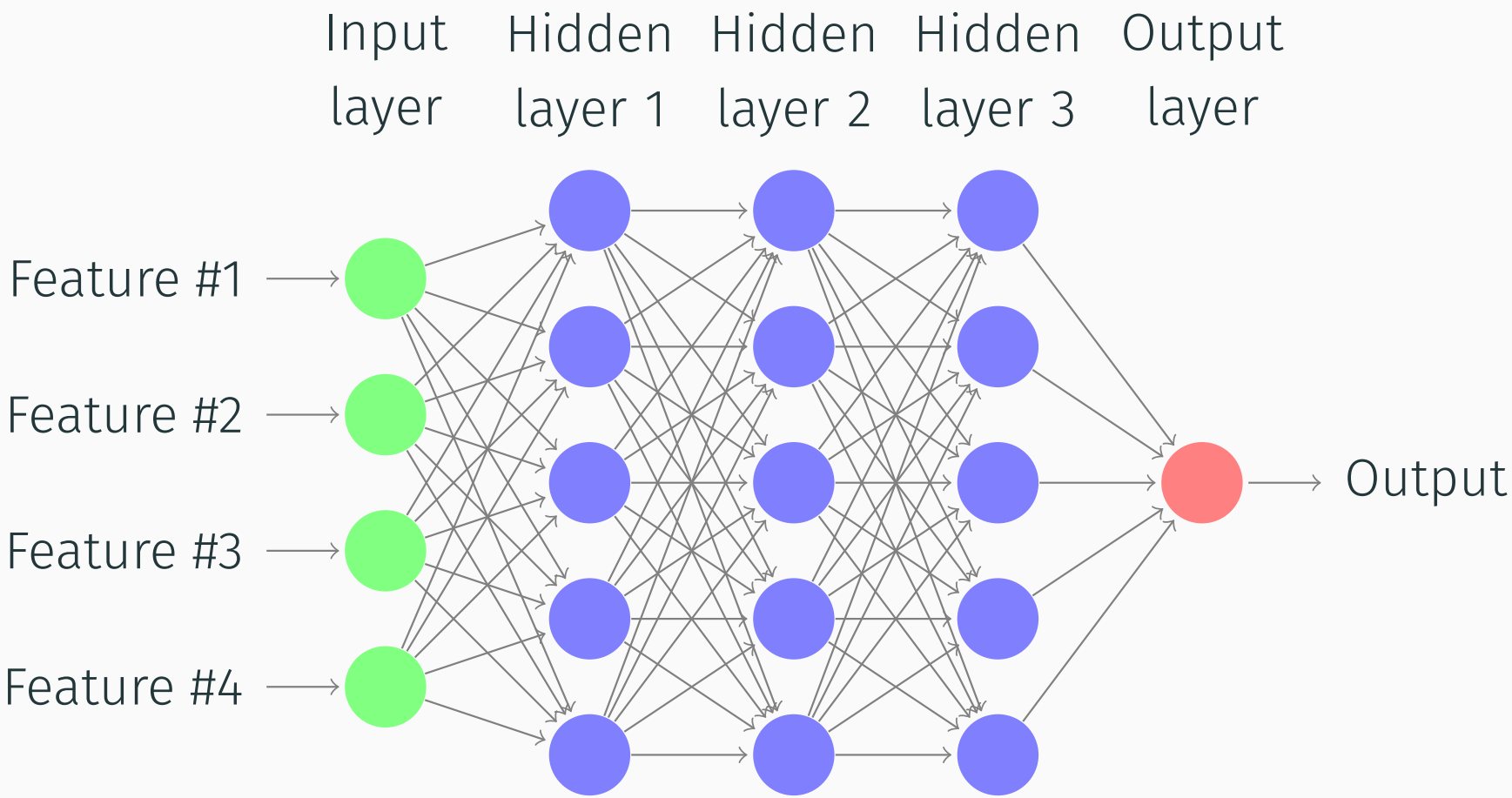
- Inputs x_i are the different features of the data
- Weight w_i are the parameters of the model to optimize
- If the activation function is identity, it is equivalent to a linear regression

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

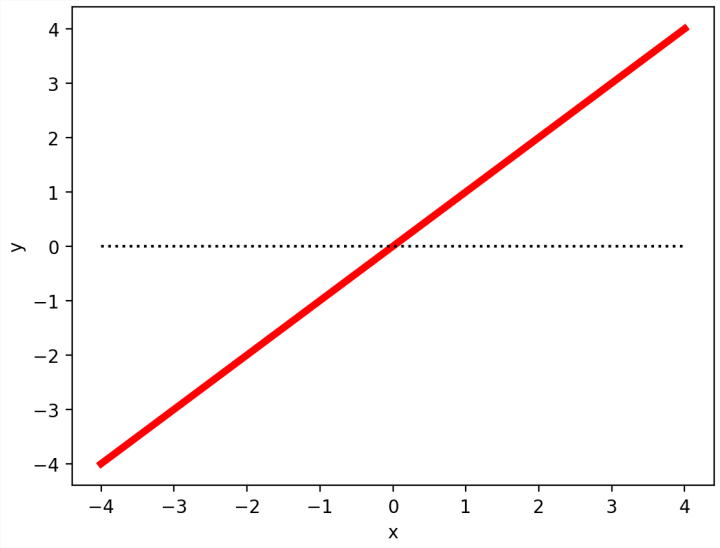
More complexe models are build by combining several perceptrons

Multi-layer perceptron (Densely connected layers)

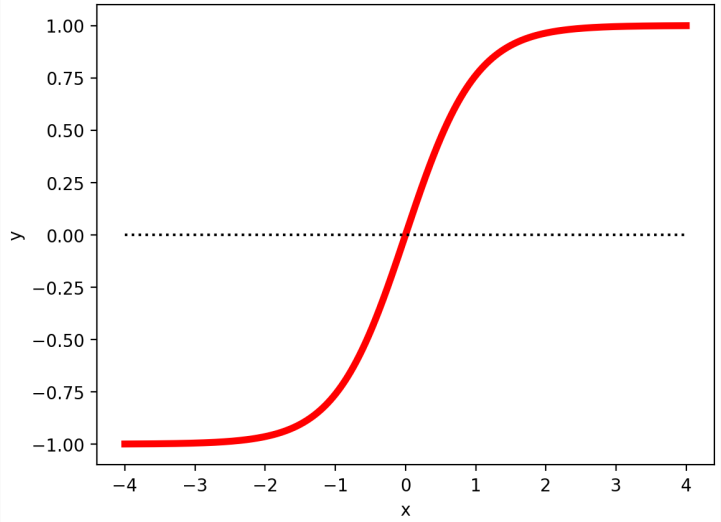


Most usual activation functions

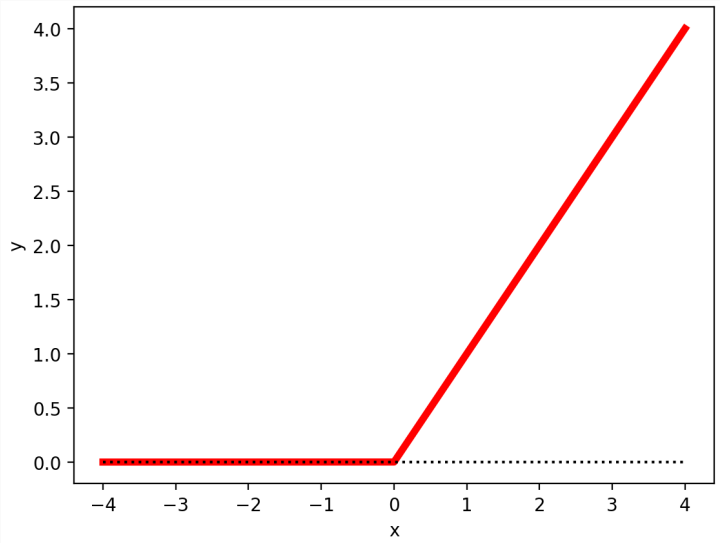
Linear



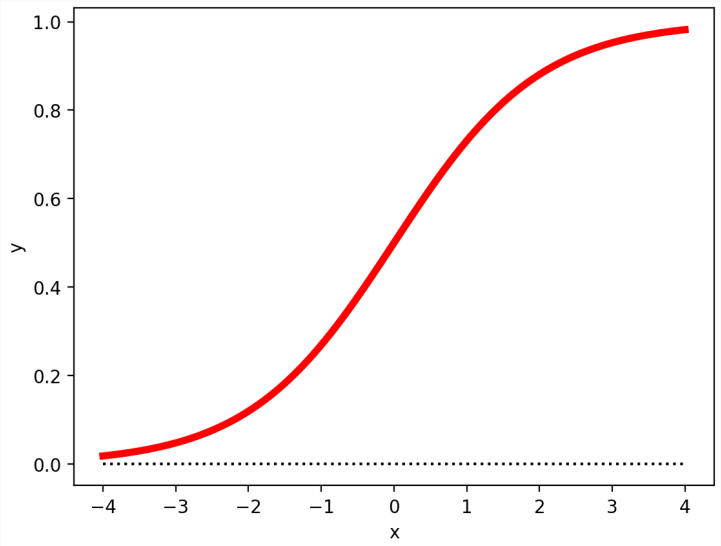
Hyperbolic tangent



ReLU

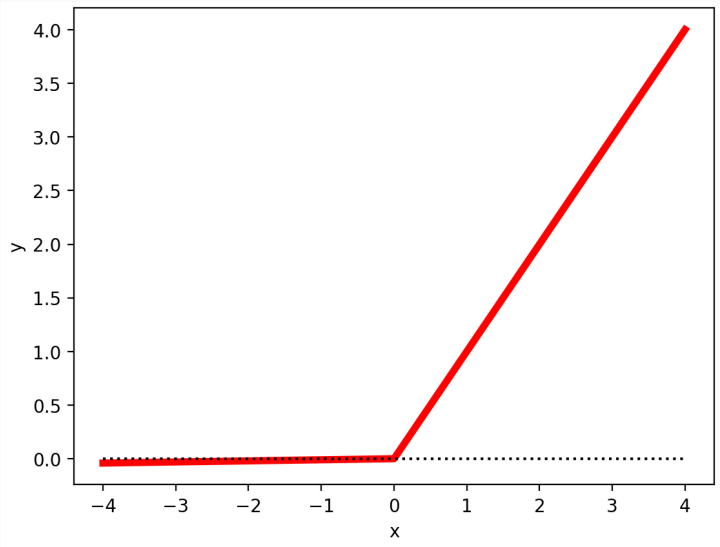


Sigmoid

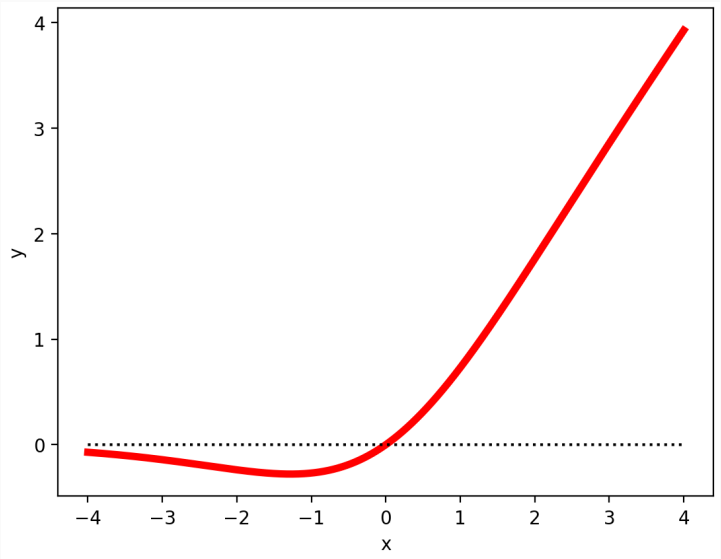


New fancy activation functions

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

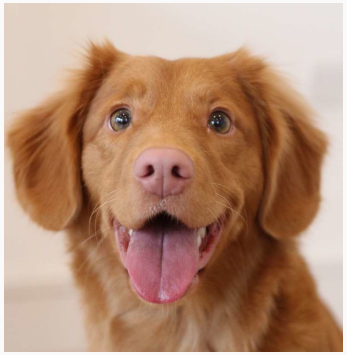
$$L(p, y) = -\sum_i \sum_j y_{i,j}. \log p_{i,j}$$

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:

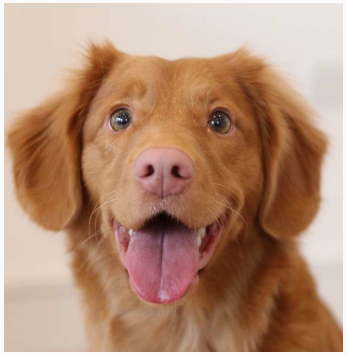
$$L = -\sum y_i \cdot \log(f(x_i)) + (1 - y_i) \cdot \log(1 - f(x_i))$$

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 number 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 number 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

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

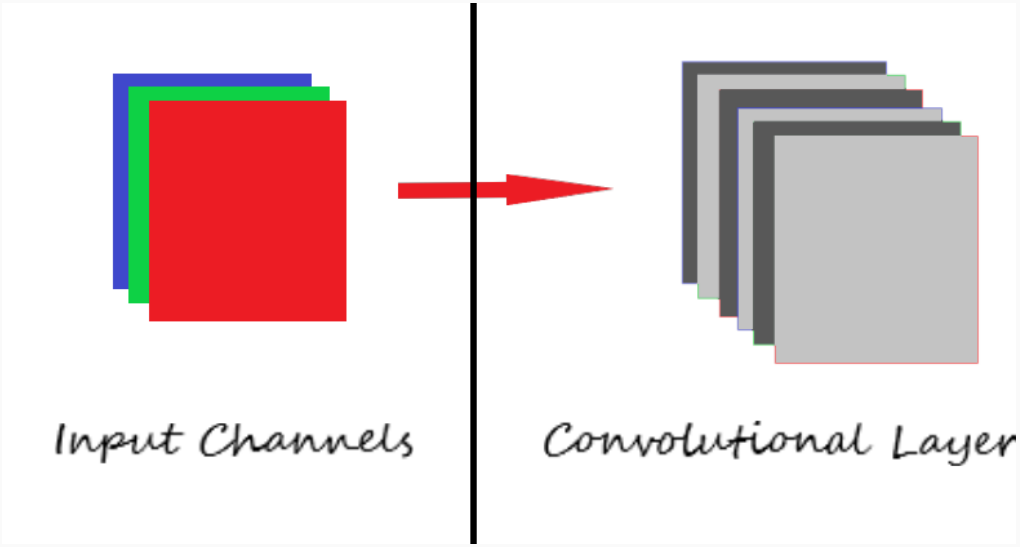
Main parameters of a convolutional layer

- 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.

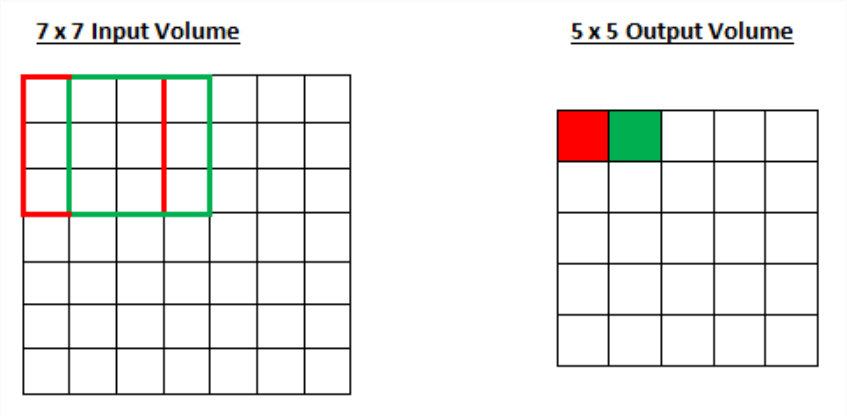


$O = \frac{W-K+2P}{S} + 1$, where O is the output size and W the input size.

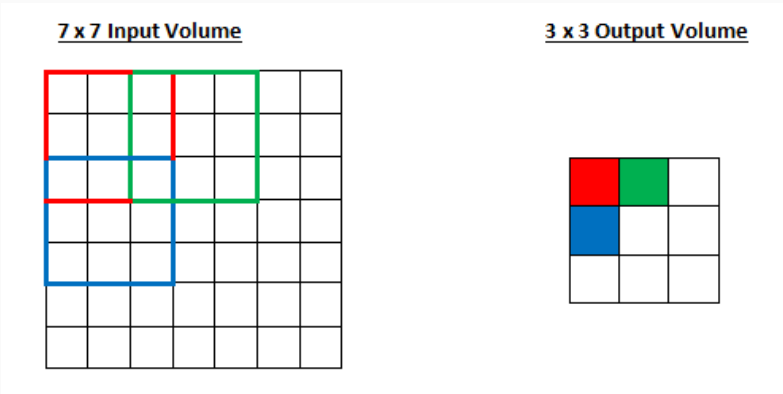
Main parameters of a convolutional layer

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

$S = 1$



$S = 2$

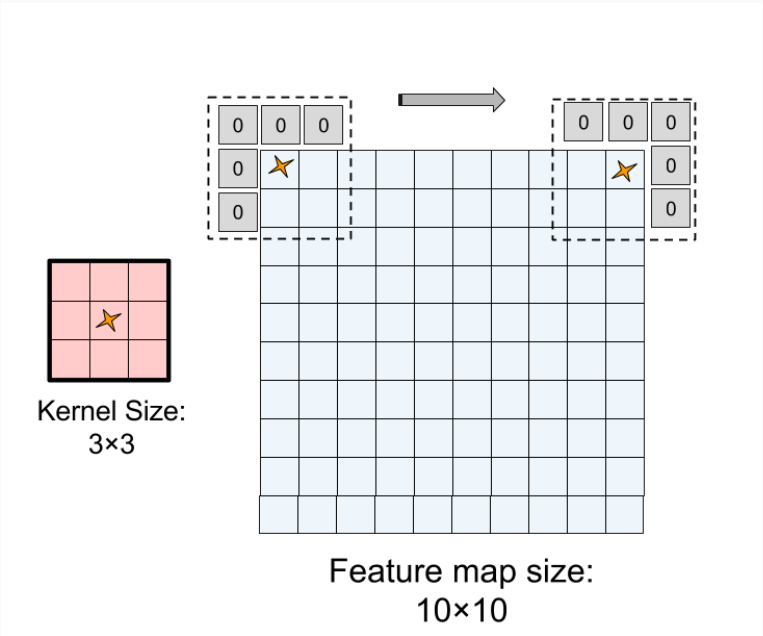


$O = \frac{W-K+2P}{S} + 1$, where O is the output size and W the input size.

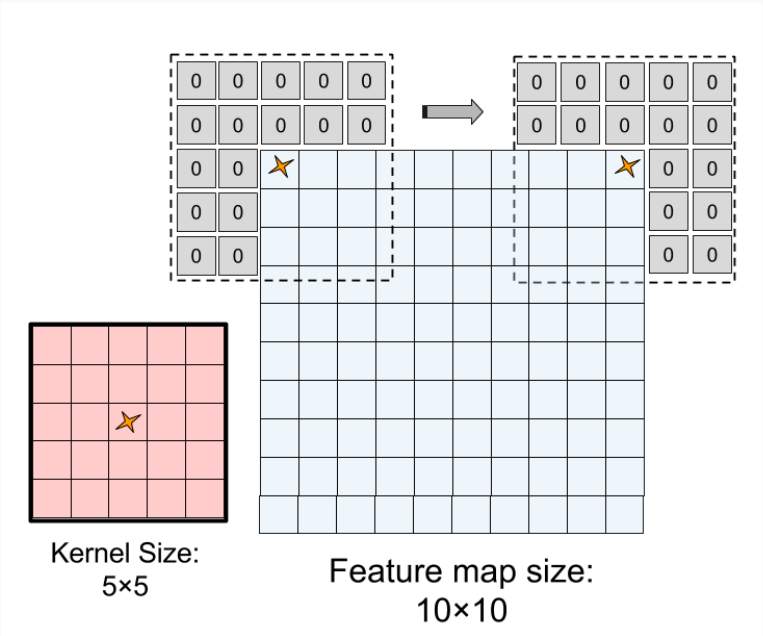
Main parameters of a convolutional layer

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

$P = 1$



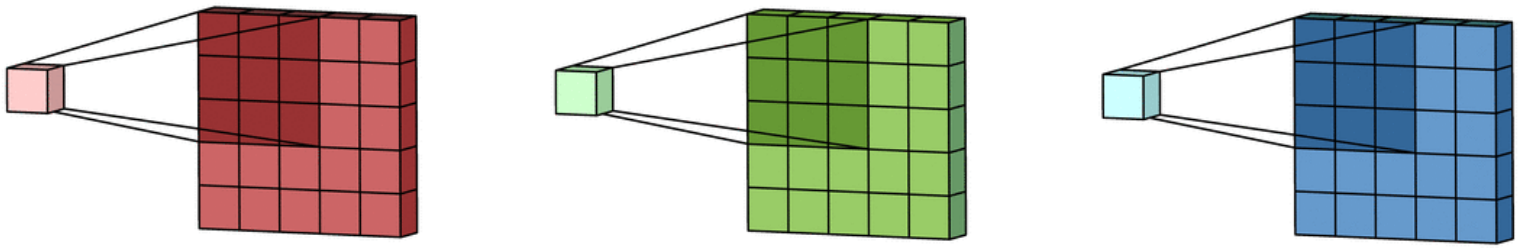
$P = 2$



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

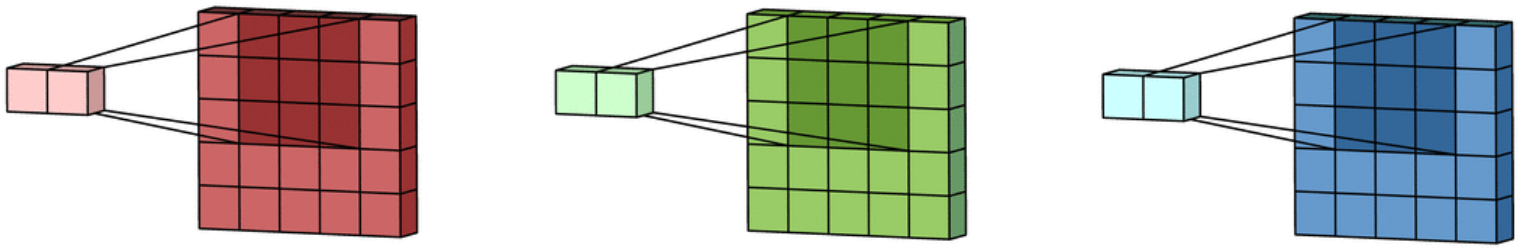
Summary of Convolutional layer steps

1. Convolution



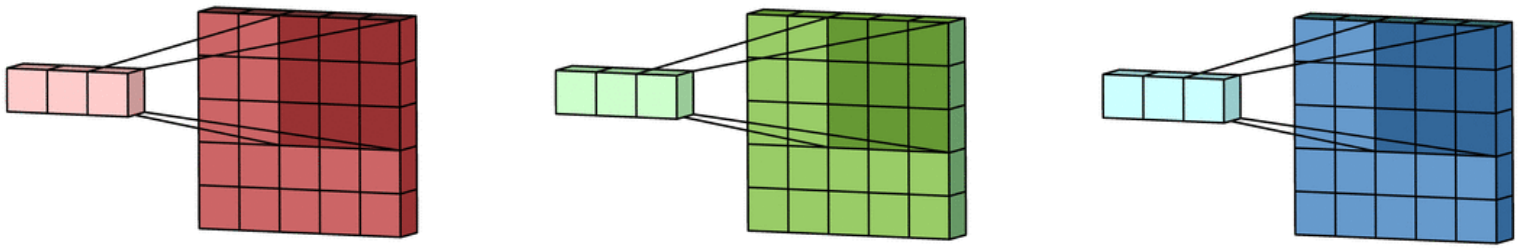
Summary of Convolutional layer steps

1. Convolution



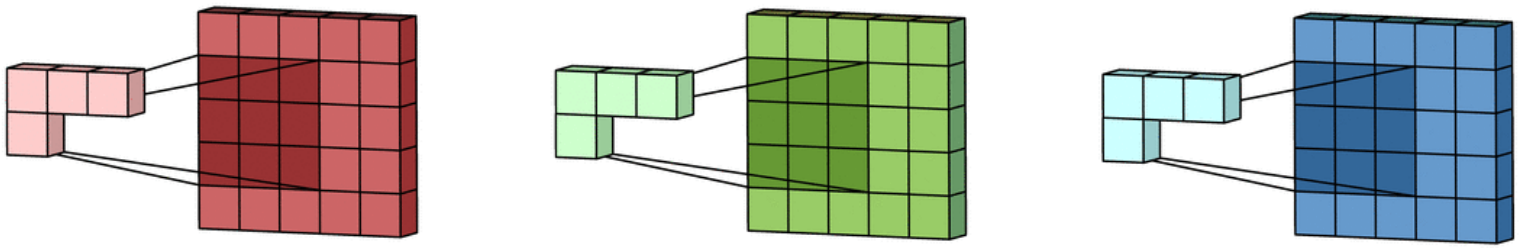
Summary of Convolutional layer steps

1. Convolution



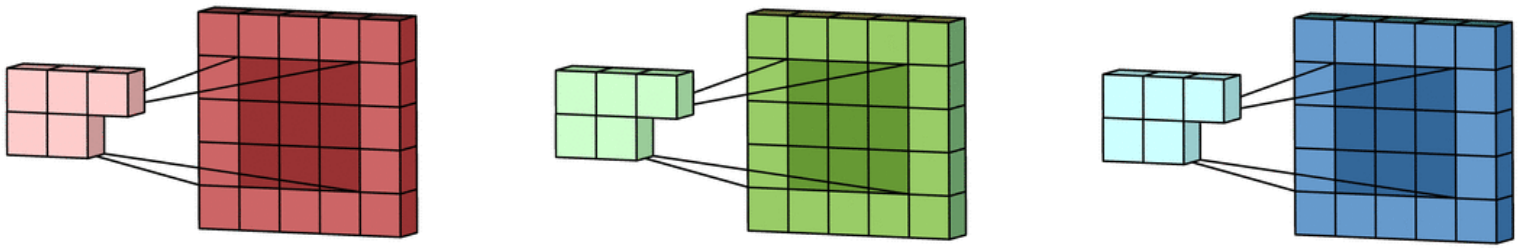
Summary of Convolutional layer steps

1. Convolution



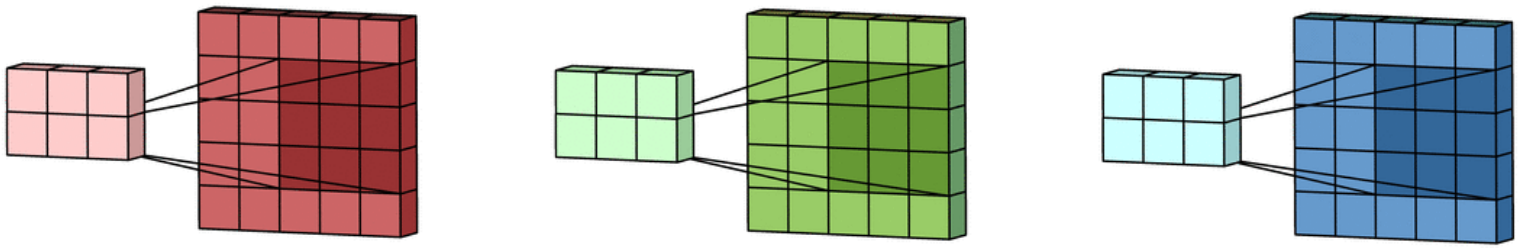
Summary of Convolutional layer steps

1. Convolution



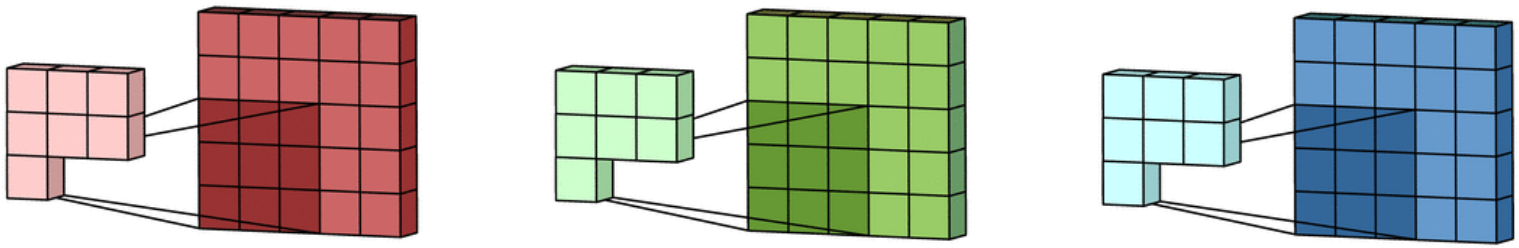
Summary of Convolutional layer steps

1. Convolution



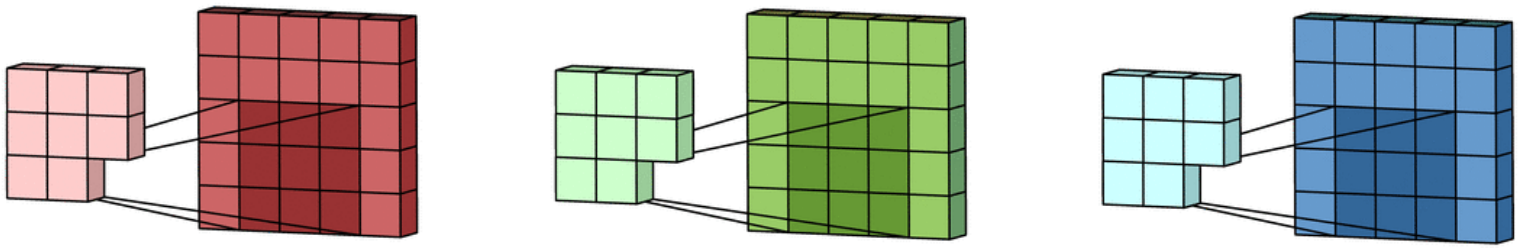
Summary of Convolutional layer steps

1. Convolution



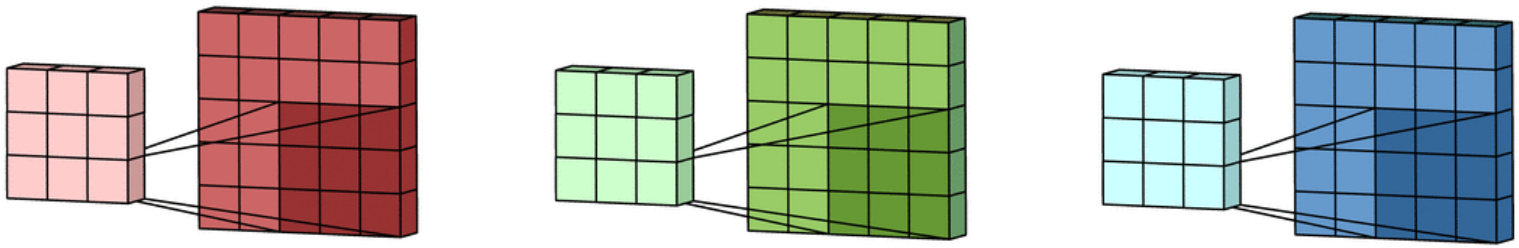
Summary of Convolutional layer steps

1. Convolution



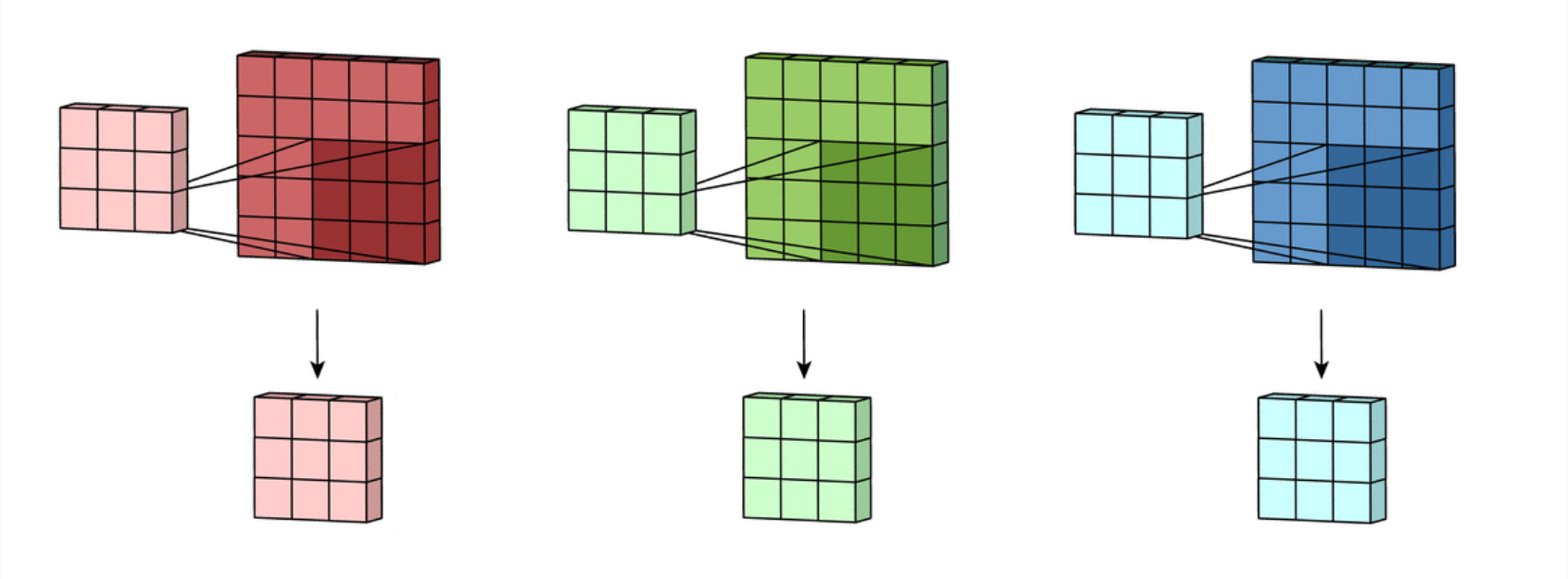
Summary of Convolutional layer steps

1. Convolution



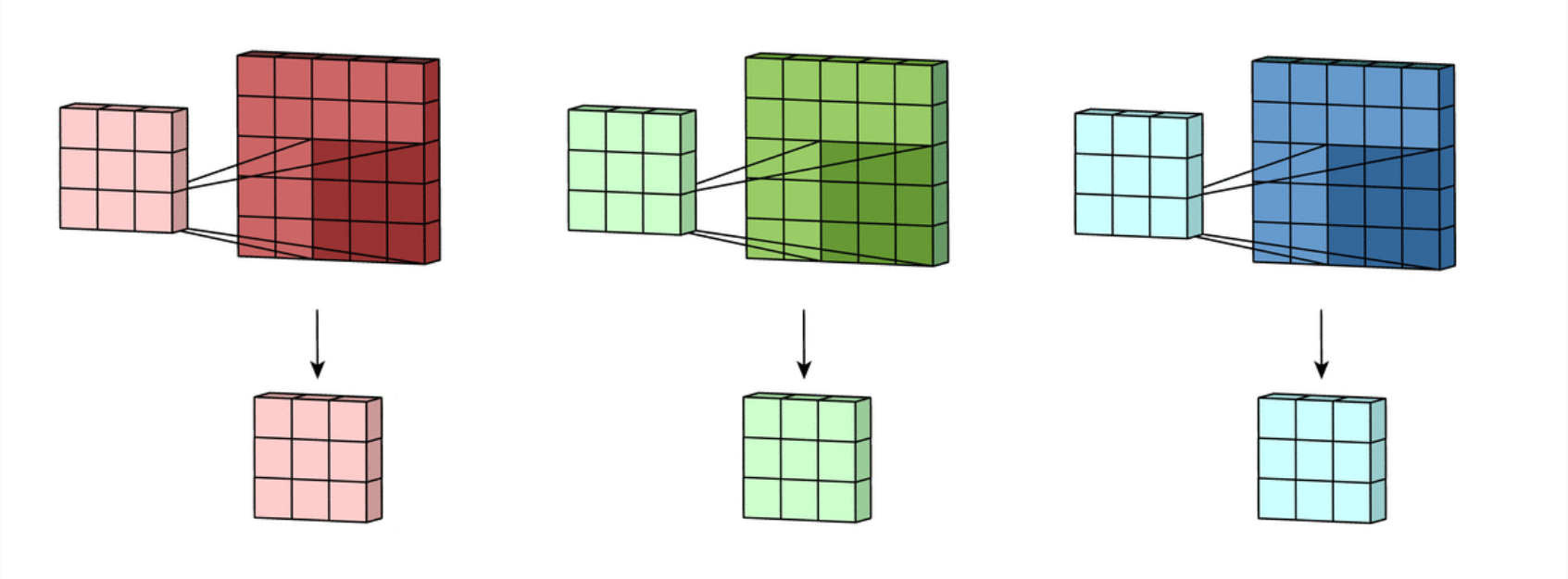
Summary of Convolutional layer steps

1. Convolution



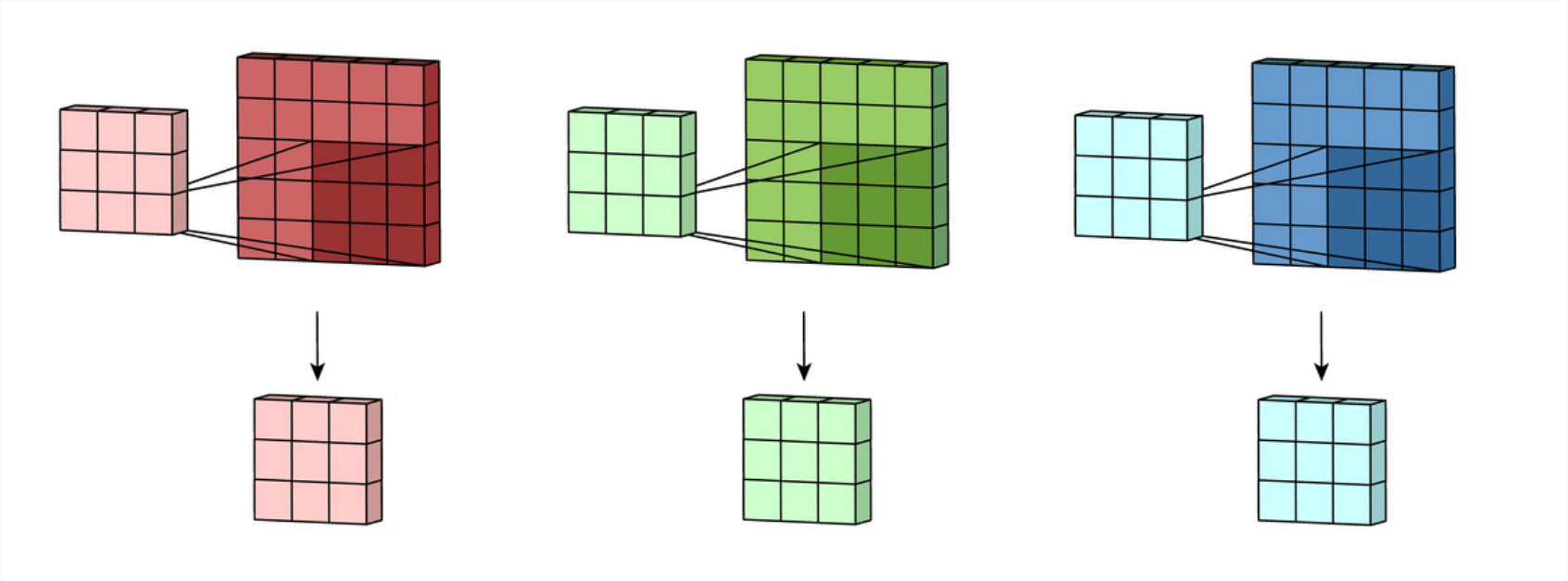
Summary of Convolutional layer steps

1. Convolution

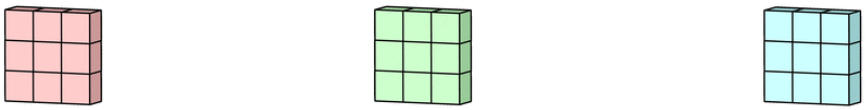


Summary of Convolutional layer steps

1. Convolution

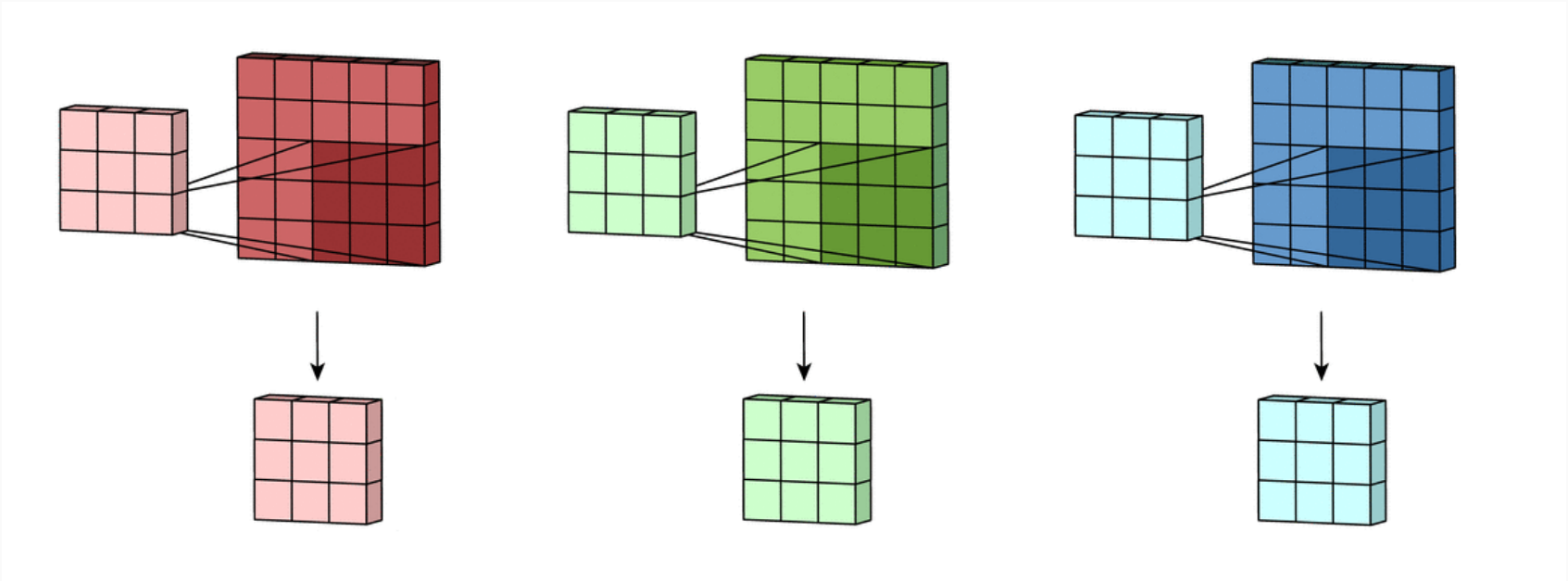


2. Addition

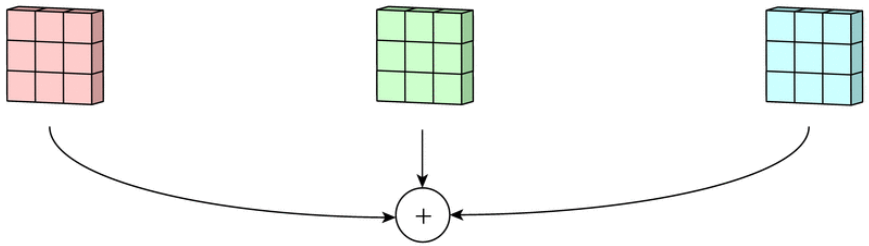


Summary of Convolutional layer steps

1. Convolution

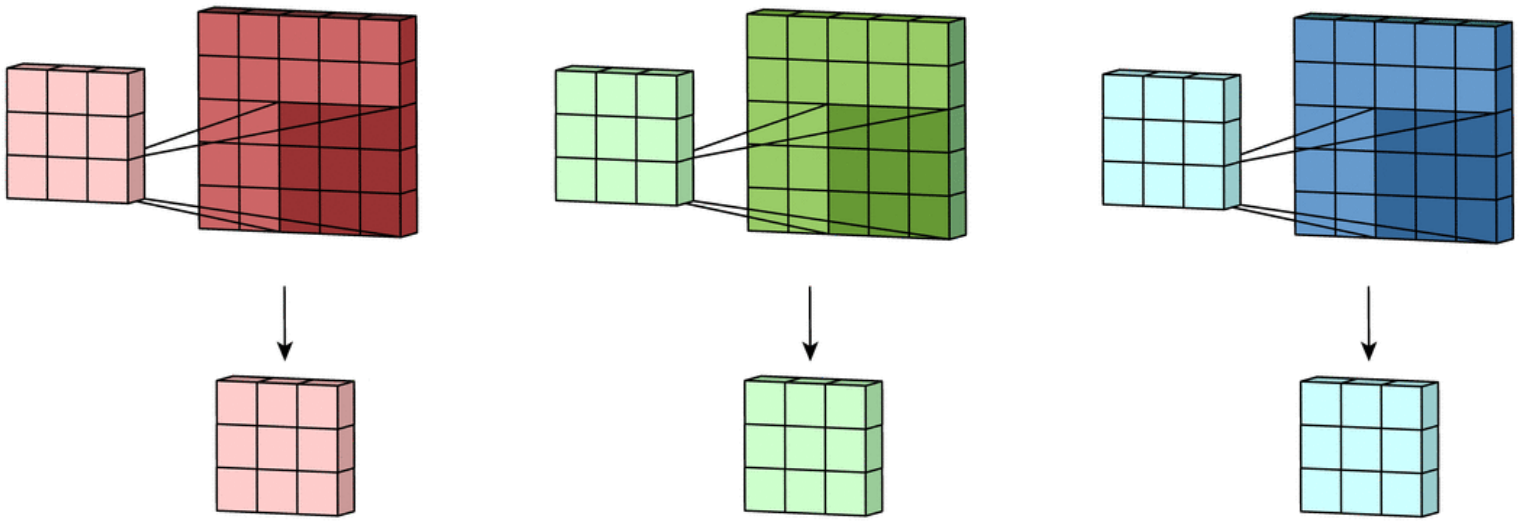


2. Addition

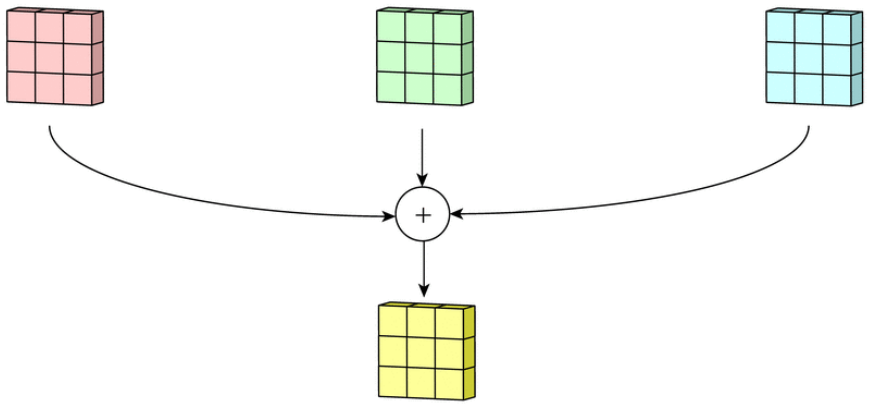


Summary of Convolutional layer steps

1. Convolution

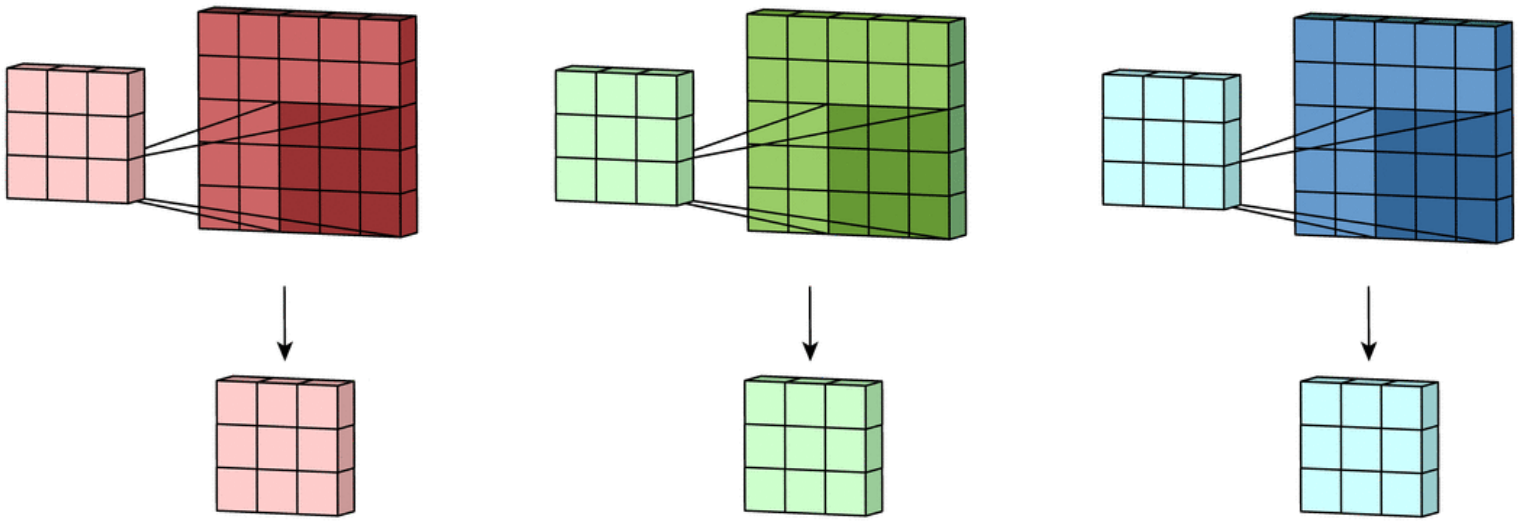


2. Addition

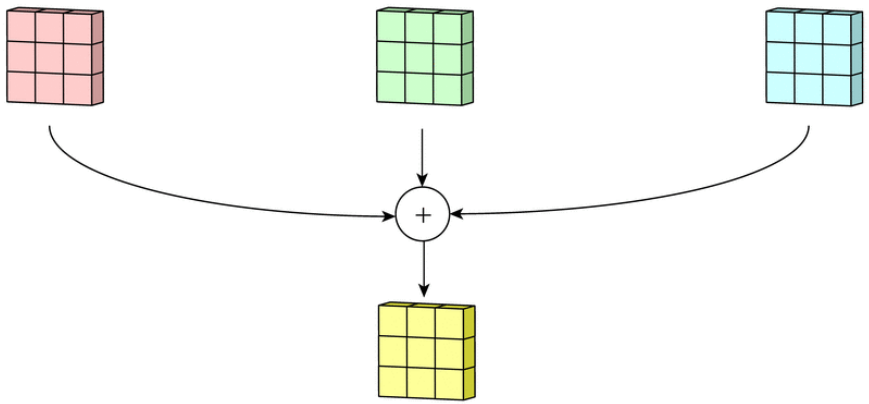


Summary of Convolutional layer steps

1. Convolution

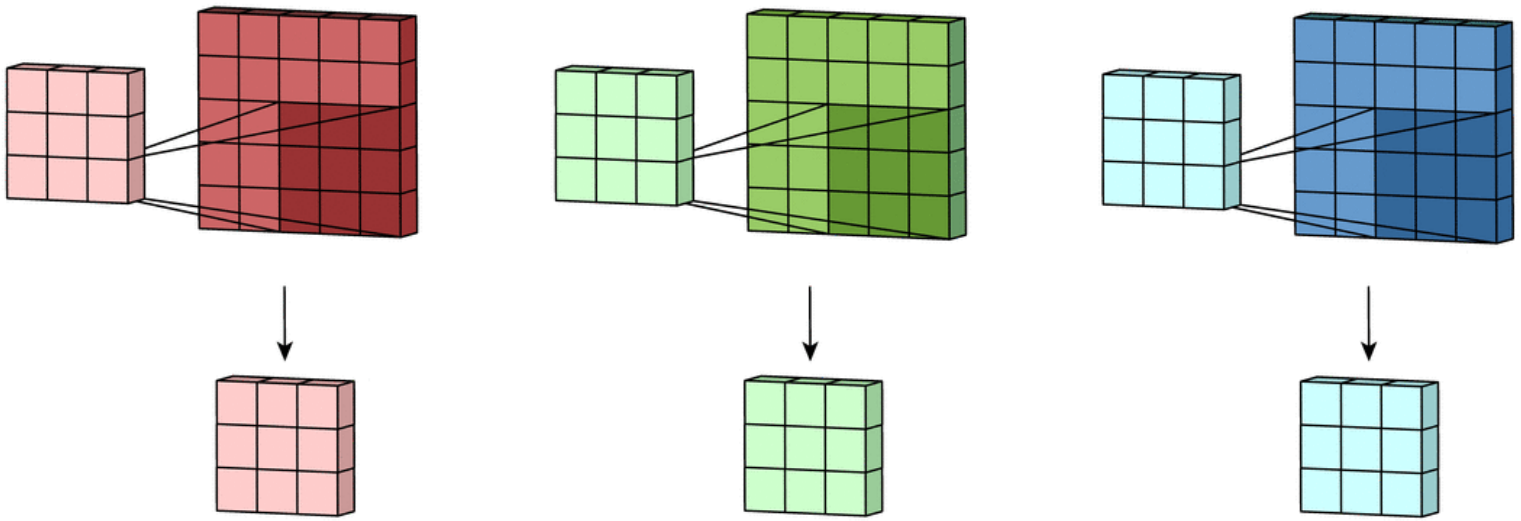


2. Addition

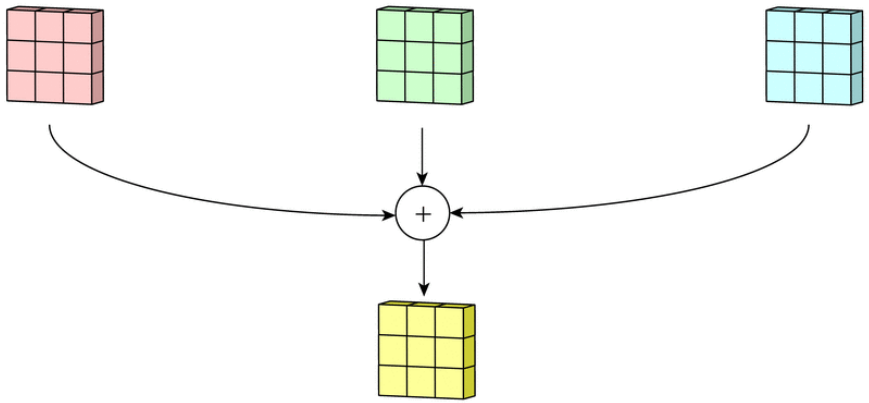


Summary of Convolutional layer steps

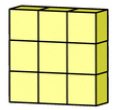
1. Convolution



2. Addition

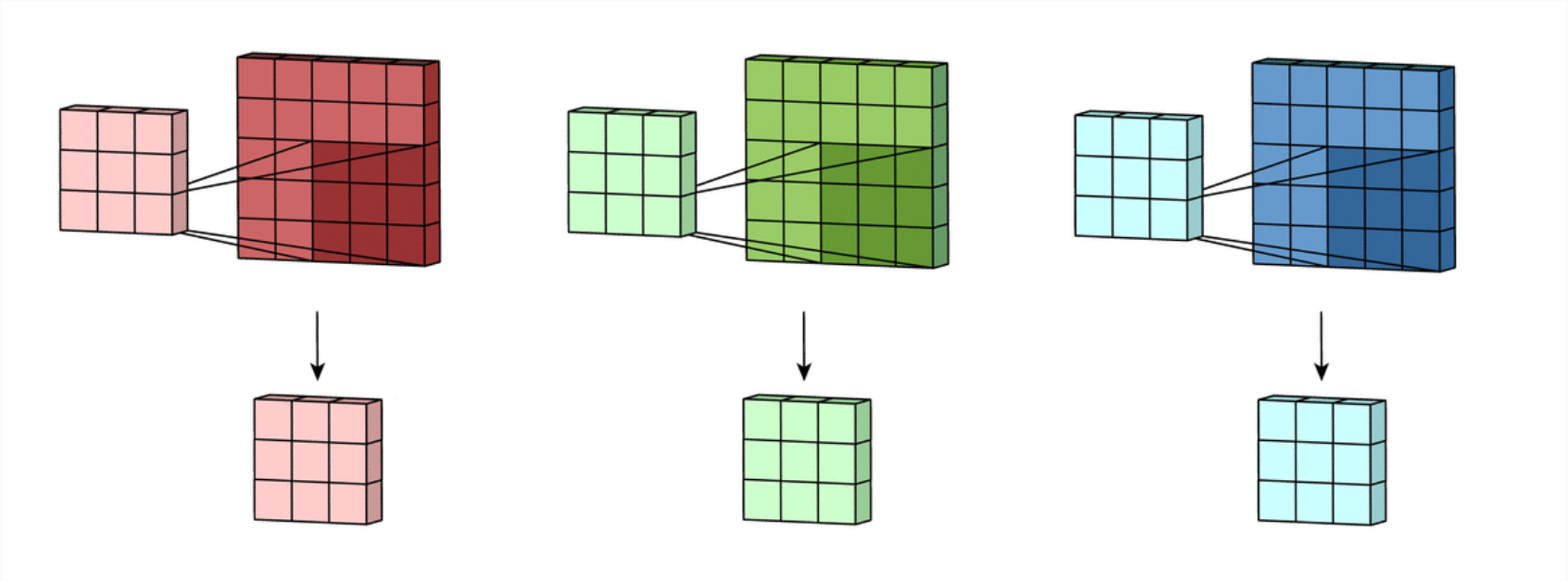


3. Bias

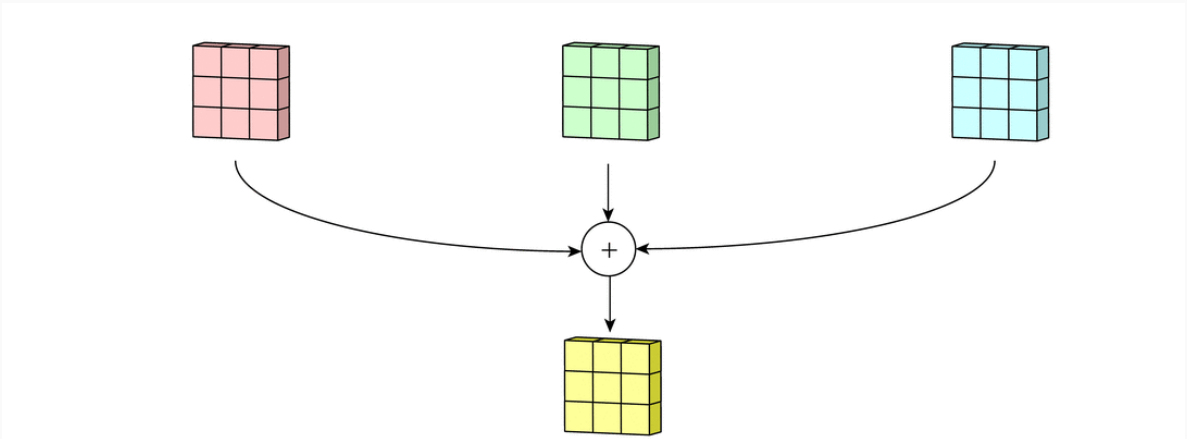


Summary of Convolutional layer steps

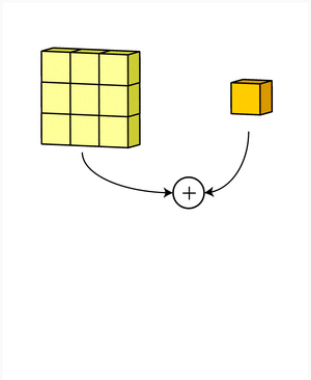
1. Convolution



2. Addition

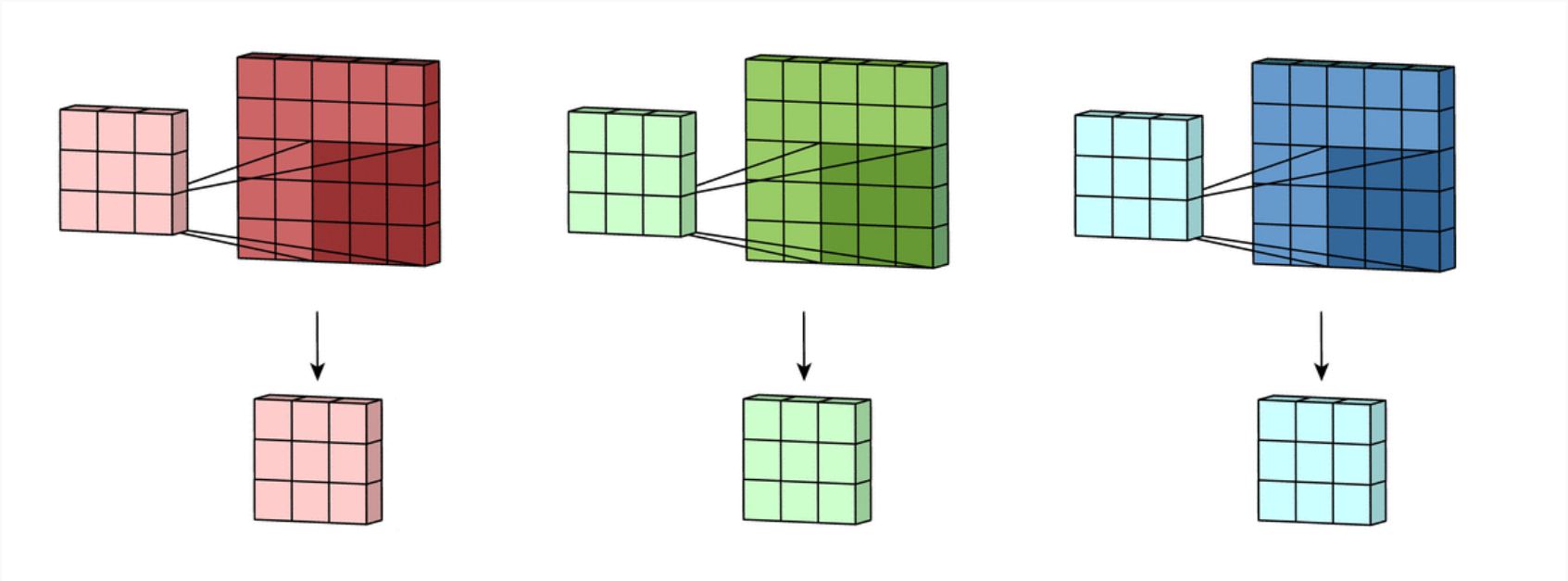


3. Bias

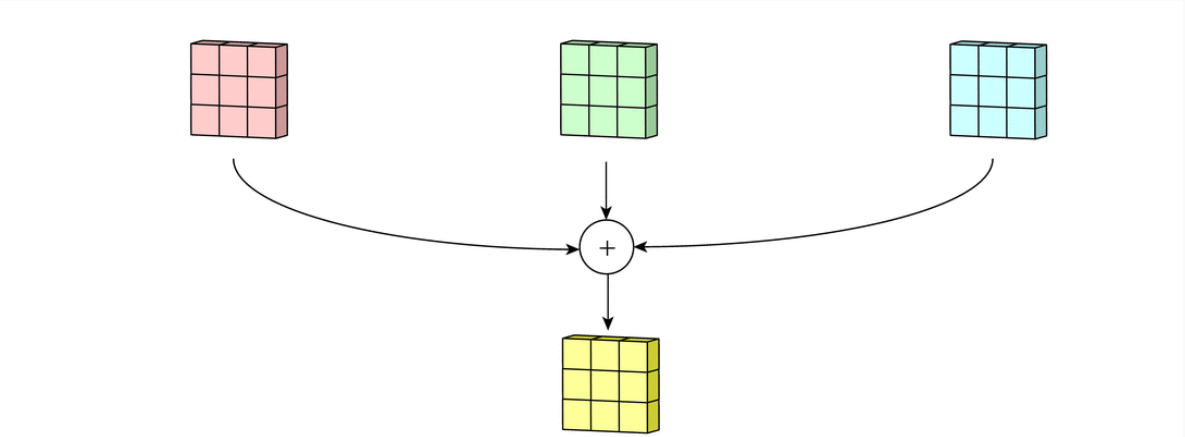


Summary of Convolutional layer steps

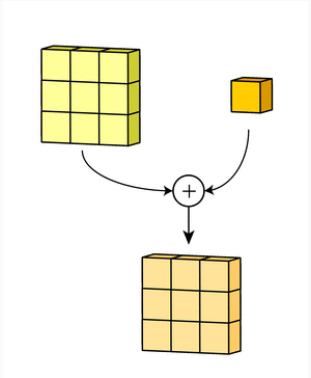
1. Convolution



2. Addition



3. Bias

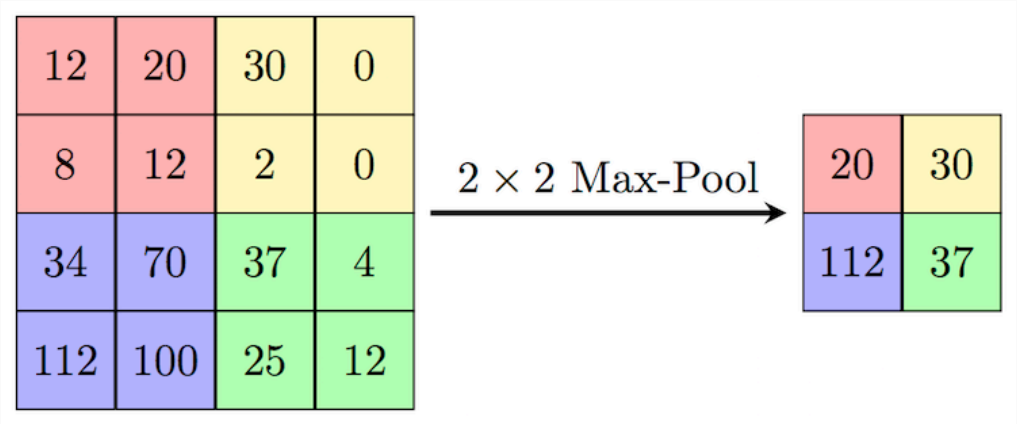


Remarks on Convolutional layers

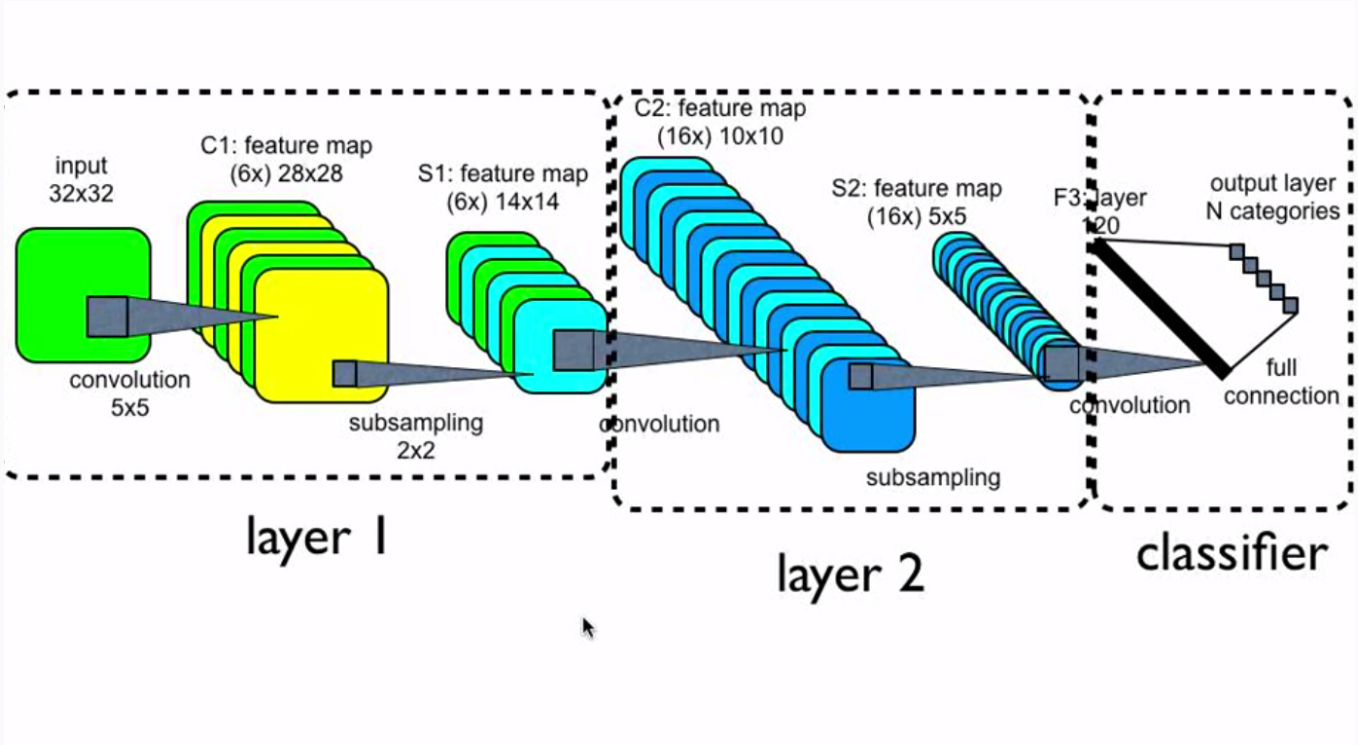
- Convolutional layers are acting locally on the image (But you can still use large scale information by adding more layers)
- Convolutions are invariant by translation (the weights do not depend on the location on the image).
- They can handle images of different sizes.

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

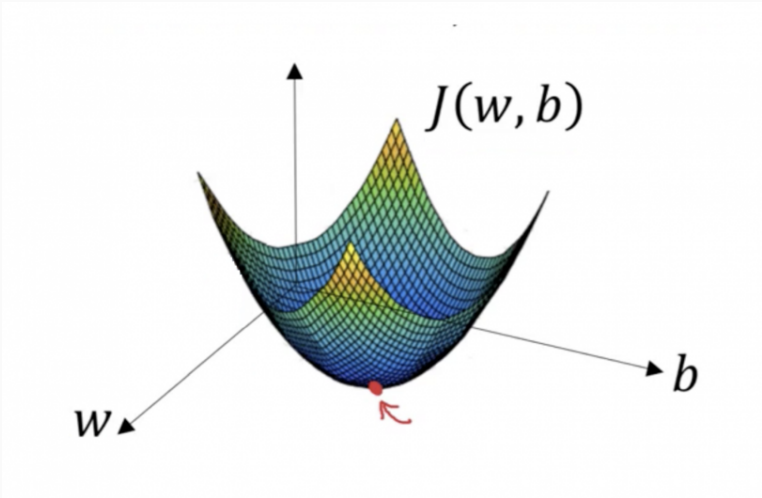
Optimization using gradient descent

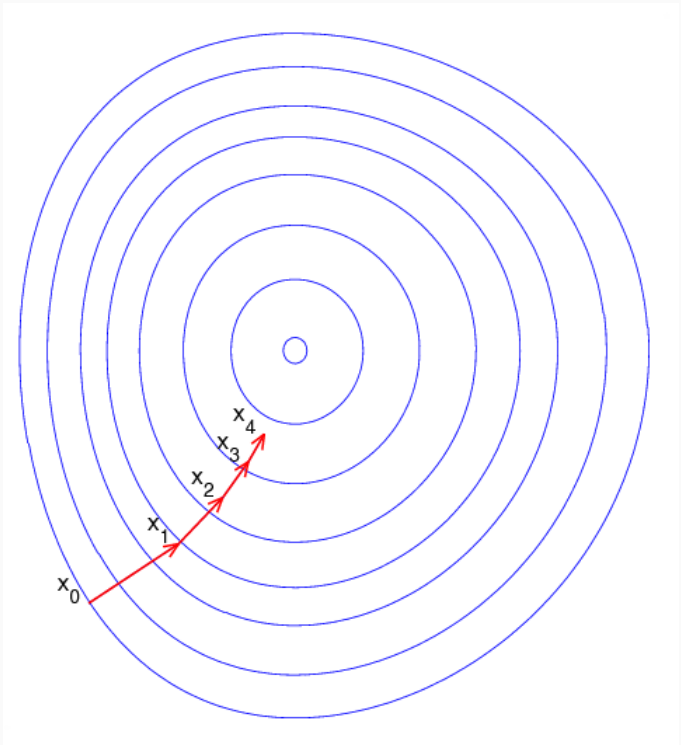
What is gradient descent

Objective

Minimize the function $L(\boldsymbol{\theta})$ where $\boldsymbol{\theta}$ is a vector of parameters (e.g. weights of a neural net).

Example with $\boldsymbol{\theta} = (w, b)$





Iterative algorithm

1. We start with a ”first guess” of the parameter $\boldsymbol{\theta}_0$

Iterative algorithm

1. We start with a "first guess" of the parameter $\boldsymbol{\theta}_0$
2. we iterate over several values of the parameters following the update rule:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \gamma \nabla L(\boldsymbol{\theta}_k),$$

γ is called the learning rate.

Iterative algorithm

1. We start with a "first guess" of the parameter $\boldsymbol{\theta}_0$
2. we iterate over several values of the parameters following the update rule:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \gamma \nabla L(\boldsymbol{\theta}_k),$$

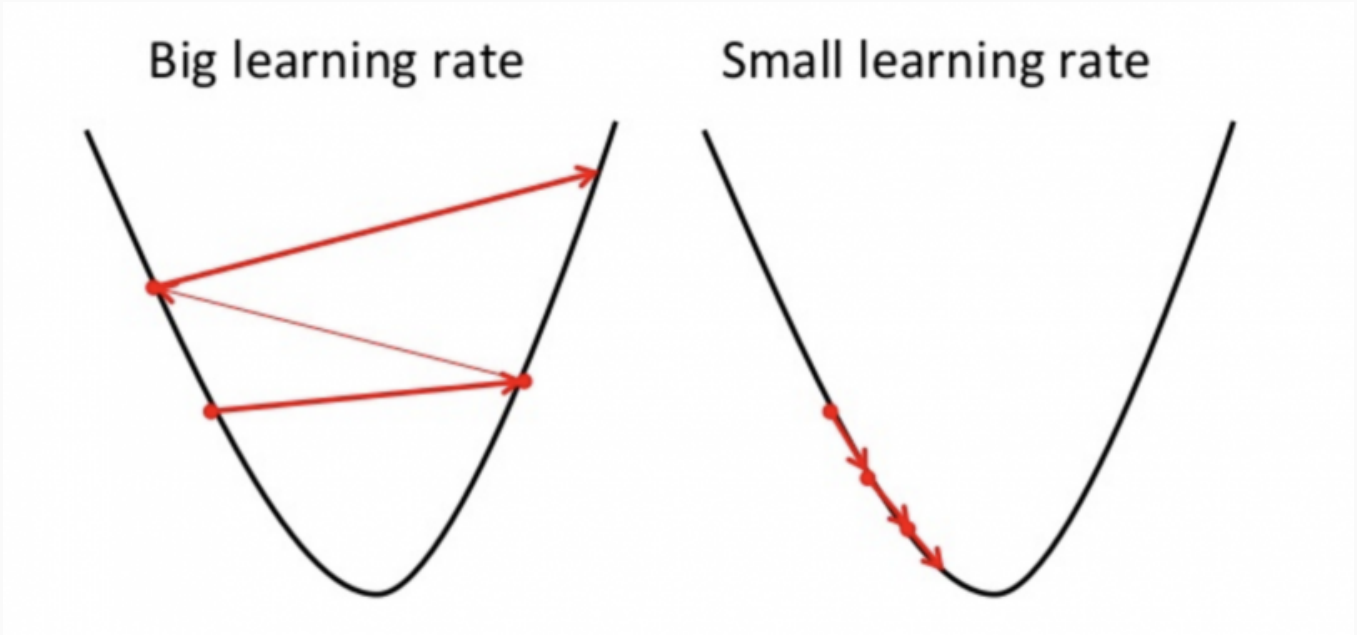
γ is called the learning rate.

Iterative algorithm

1. We start with a ”first guess” of the parameter $\boldsymbol{\theta}_0$
2. we iterate over several values of the parameters following the update rule:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \gamma \nabla L(\boldsymbol{\theta}_k),$$

γ is called the learning rate.



Few comments

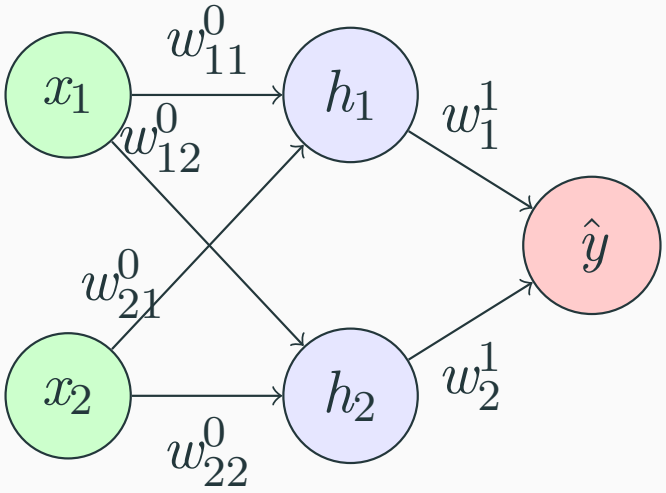
- The learning γ is an important hyperparameter that needs to be tuned using, e.g random search

Few comments

- The learning γ is an important hyperparameter that needs to be tuned using, e.g random search
- The key part of the formula is the computation of the gradient $\nabla L(\boldsymbol{\theta}_k)$

Gradient backpropagation

Training a neural-net: gradient backpropagation



1. Given a couple (x, y)

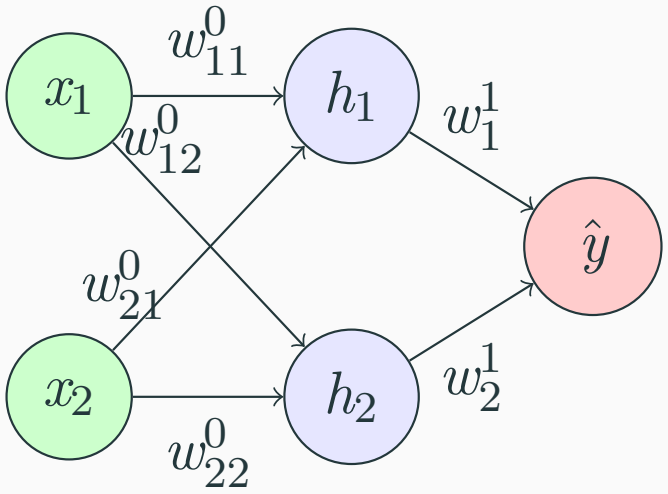
Objective

Determination of the best set of weights $\mathbf{w}$ to minimize the Loss function

$L(\mathbf{w}) = ||\hat{y}(\mathbf{w}) - y||^2$.

Calculation of $\partial L / \partial w$

Training a neural-net: gradient backpropagation

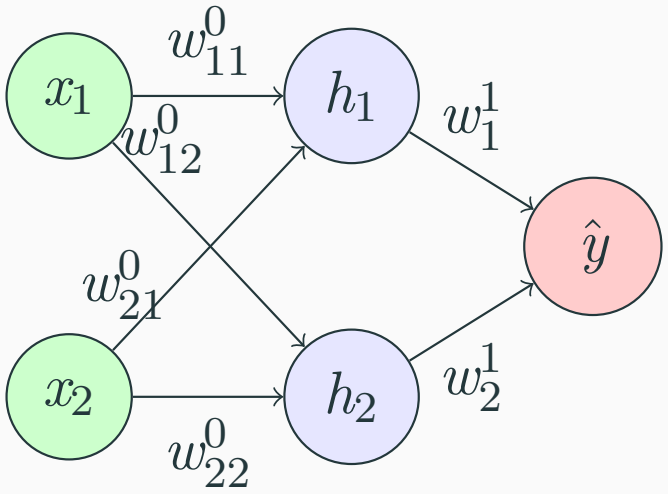


1. Given a couple (x, y)
2. Forward computation:
   $h_j = f_0(\sum_{i=1}^{2} w^0_{ij}.x_i)$
   $\hat{y} = f_1(\sum_{j=1}^{2} w^1_j.h_j)$

Objective

Determination of the best set of weights $\mathbf{w}$ to minimize the Loss function
$L(\mathbf{w}) = ||\hat{y}(\mathbf{w}) - y||^2$.
Calculation of $\partial L/\partial w$

Training a neural-net: gradient backpropagation

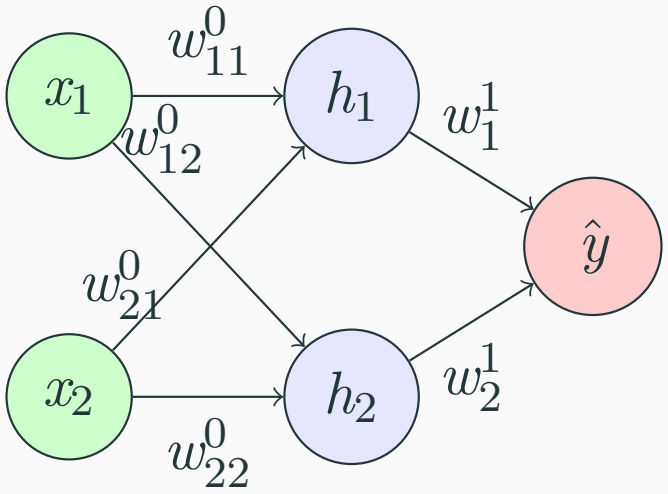


1. Given a couple (x, y)
2. Forward computation:
   $h_j = f_0(\sum_{i=1}^{2} w^0_{ij}.x_i)$
   $\hat{y} = f_1(\sum_{j=1}^{2} w^1_j.h_j)$
3. Compute the gradient of the loss:
   $\boxed{\partial L/\partial \hat{y}}$

Objective

Determination of the best set of weights $\mathbf{w}$ to minimize the Loss function
$L(\mathbf{w}) = ||\hat{y}(\mathbf{w}) - y||^2$.
Calculation of $\partial L/\partial w$

Training a neural-net: gradient backpropagation



Objective

Determination of the best set of weights $\mathbf{w}$ to minimize the Loss function

$L(\mathbf{w}) = ||\hat{y}(\mathbf{w}) - y||^2$.

Calculation of $\partial L / \partial w$

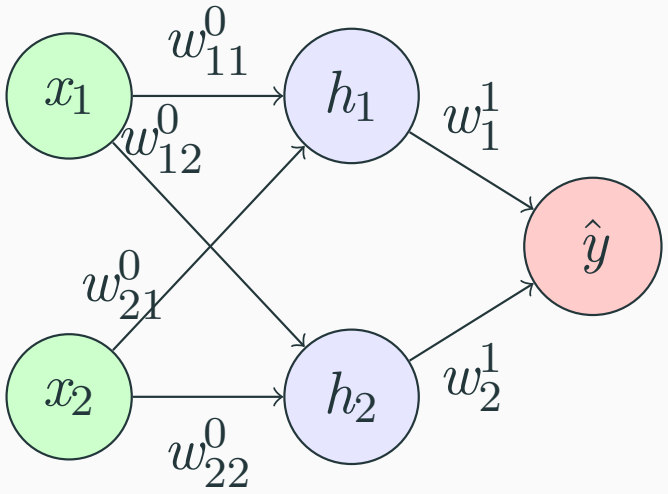
1. Given a couple (x, y)
2. Forward computation:

   $h_j = f_0(\sum_{i=1}^{2} w^0_{ij}.x_i)$

   $\hat{y} = f_1(\sum_{j=1}^{2} w^1_j.h_j)$
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

$L(\mathbf{w}) = ||\hat{y}(\mathbf{w}) - y||^2$.

Calculation of $\partial L/\partial w$

1. Given a couple (x, y)
2. Forward computation:

   $h_j = f_0(\sum_{i=1}^{2} w^0_{ij}.x_i)$

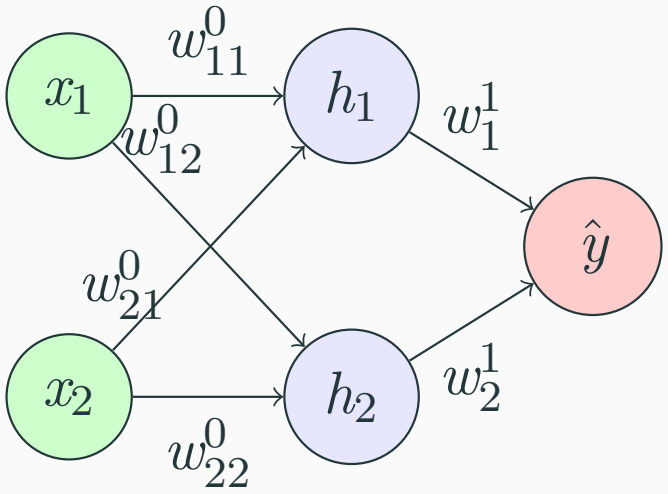
   $\hat{y} = f_1(\sum_{j=1}^{2} w^1_j.h_j)$
3. Compute the gradient of the loss:

   $\boxed{\partial L/\partial \hat{y}}$
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

$L(\mathbf{w}) = ||\hat{y}(\mathbf{w}) - y||^2$.

Calculation of $\partial L/\partial w$

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

     $\partial L/\partial w^0_{ij} = \boxed{\partial L/\partial h_j}.\partial f_1/\partial w^0_{ij}$

Optimizing a machine learning (gradient method)

Optimizing the loss

Several loss function (depending on the problem) can be defined.

For example, Mean Square Error:

Method

Find a minimum of L by adjusting the parameters (weights) $\mathbf{w}$ given the gradient of the loss with respect to the weights $\nabla_{\mathbf{w}} L$.

Batch Vs Stochastic training

Dataset: (X, Y) with N samples denoted $(\mathbf{x_i}, y_i)$

Batch gradient:

Require: Learning rate(s): ν_k
Require: Initial weights: $\mathbf{w}$
  $k \leftarrow 1$
  while stopping criterion not met **do**
    Compute gradient:
  $\mathbf{g} \leftarrow \frac{1}{N} \sum_i^N \nabla_{\mathbf{w}} L(f(\mathbf{x}_i, y_i))$
    Update weights: $\mathbf{w} \leftarrow \mathbf{w} - \nu_k \mathbf{g}$
    $k \leftarrow k + 1$
  end while

1 Update / N forwards

Batch Vs Stochastic training

Dataset: (X, Y) with N samples denoted $(\mathbf{x_i}, y_i)$

Batch gradient:

Require: Learning rate(s): ν_k
Require: Initial weights: $\mathbf{w}$
  $k \leftarrow 1$
  while stopping criterion not met **do**
    Compute gradient:
  $\mathbf{g} \leftarrow \frac{1}{N} \sum_i^N \nabla_{\mathbf{w}} L(f(\mathbf{x}_i, y_i))$
    Update weights: $\mathbf{w} \leftarrow \mathbf{w} - \nu_k \mathbf{g}$
    $k \leftarrow k + 1$
  end while

1 Update / N forwards

Stochastic gradient:

Require: Learning rate(s): ν_k
Require: Initial weights: $\mathbf{w}$
  $k \leftarrow 1$
  while stopping criterion not met **do**
    Sample an example $(\mathbf{x}, y)$ from (X, Y)
    Compute gradient: $\mathbf{g} \leftarrow \nabla_{\mathbf{w}} L(f(\mathbf{x}, y))$
    Update weights: $\mathbf{w} \leftarrow \mathbf{w} - \nu_k \mathbf{g}$
    $k \leftarrow k + 1$
  end while

1 Update / 1 forward

Mini-Batch training

Dataset: (X, y) with N samples

Mini-Batch gradient:

Require: Learning rate(s): ν_k

Require: Initial weights: $\mathbf{w}$

$k \leftarrow 1$

while stopping criterion not met **do**

Sample m examples $(\mathbf{x}_i, y_i)$ from (X, y)

Compute gradient: $\mathbf{g} \leftarrow \frac{1}{m} \sum_i^m \nabla_{\mathbf{w}} L(f(\mathbf{x}_i, y_i)$

Update weights: $\mathbf{w} \leftarrow \mathbf{w} - \nu_k \mathbf{g}$

$k \leftarrow k + 1$

end while

Mini-Batch training

Dataset: (X, y) with N samples

Mini-Batch gradient:

Require: Learning rate(s): ν_k
Require: Initial weights: $\mathbf{w}$
  $k \leftarrow 1$
  while stopping criterion not met **do**
    Sample m examples $(\mathbf{x}_i, y_i)$ from (X, y)
    Compute gradient: $\mathbf{g} \leftarrow \frac{1}{m} \sum_i^m \nabla_{\mathbf{w}} L(f(\mathbf{x}_i, y_i)$
    Update weights: $\mathbf{w} \leftarrow \mathbf{w} - \nu_k \mathbf{g}$
    $k \leftarrow k + 1$
  end while

1 Update / m forward

$m = 1$: Pure stochastic gradient.

$m = N$: Batch gradient

Let’s have a break

https://playground.tensorflow.org

Other regularization techniques

Regularization

Definition:

Regularization refers to the set of techniques that constraints the optimization. It is generally used to avoid overfitting, but can also be used to inject prior knowledge during the training phase (e.g. set known limits to parameters)

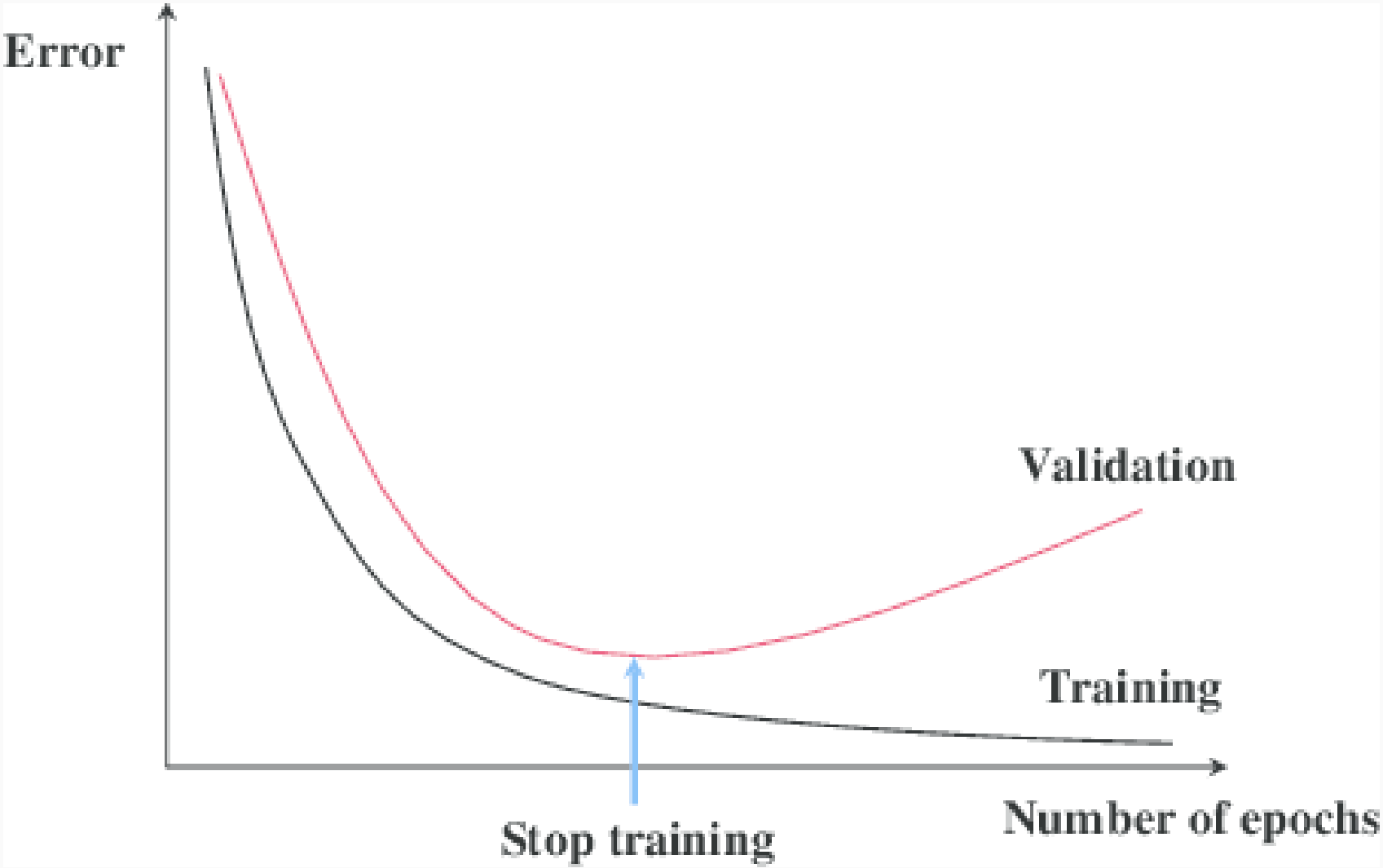
Regularization

Definition:

Regularization refers to the set of techniques that constraints the optimization. It is generally used to avoid overfitting, but can also be used to inject prior knowledge during the training phase (e.g. set known limits to parameters)

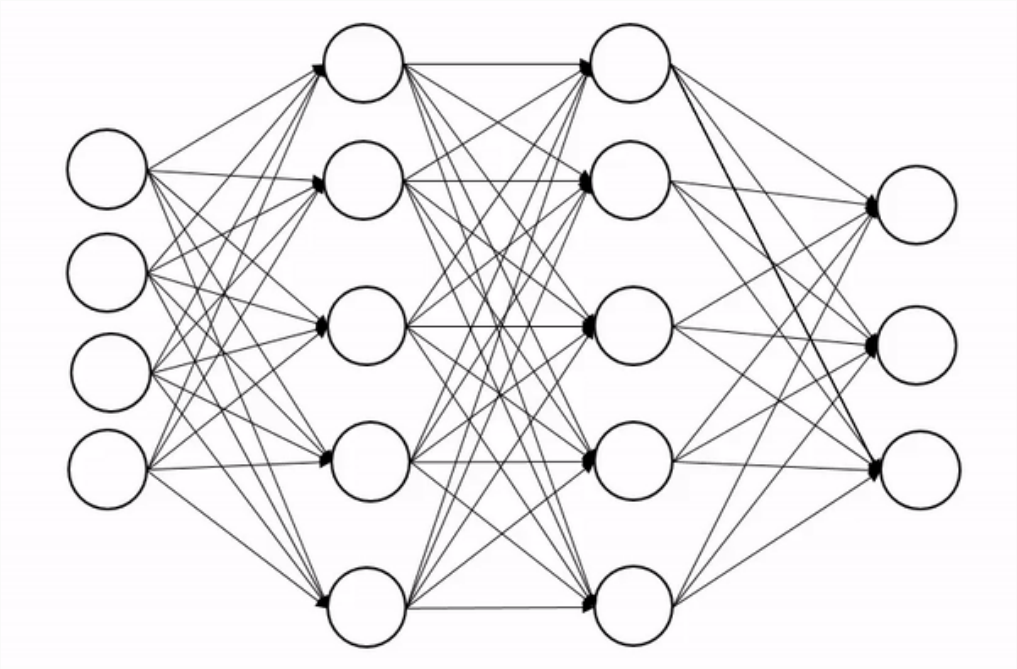
- Stochastic mini-batch gradient is a regularization technique

Early Stopping



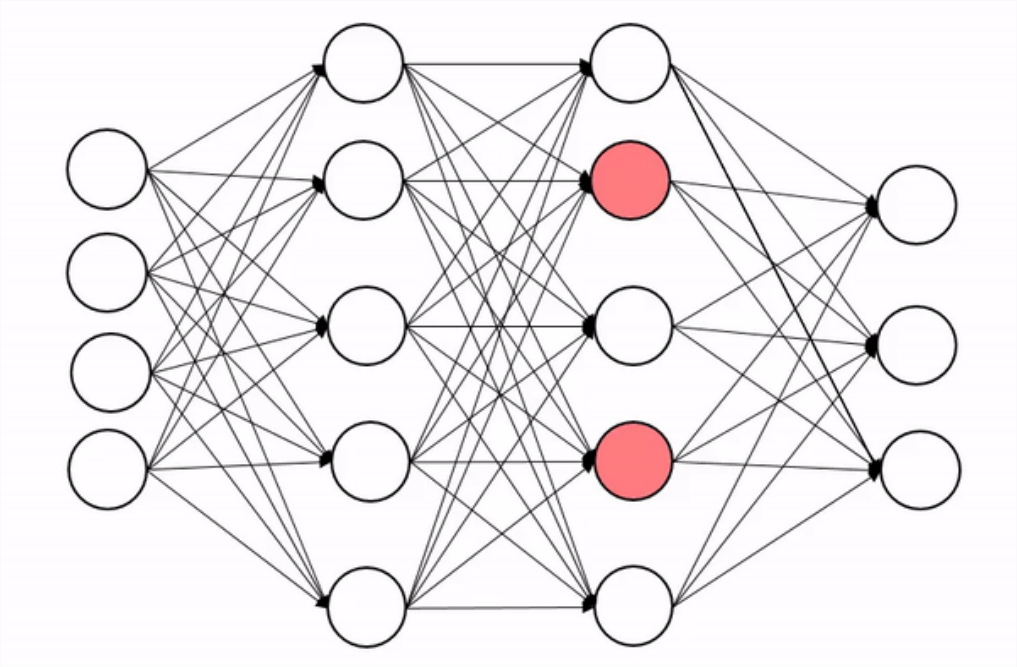
Dropout

During training, randomly remove neurons on a layer with probability p.



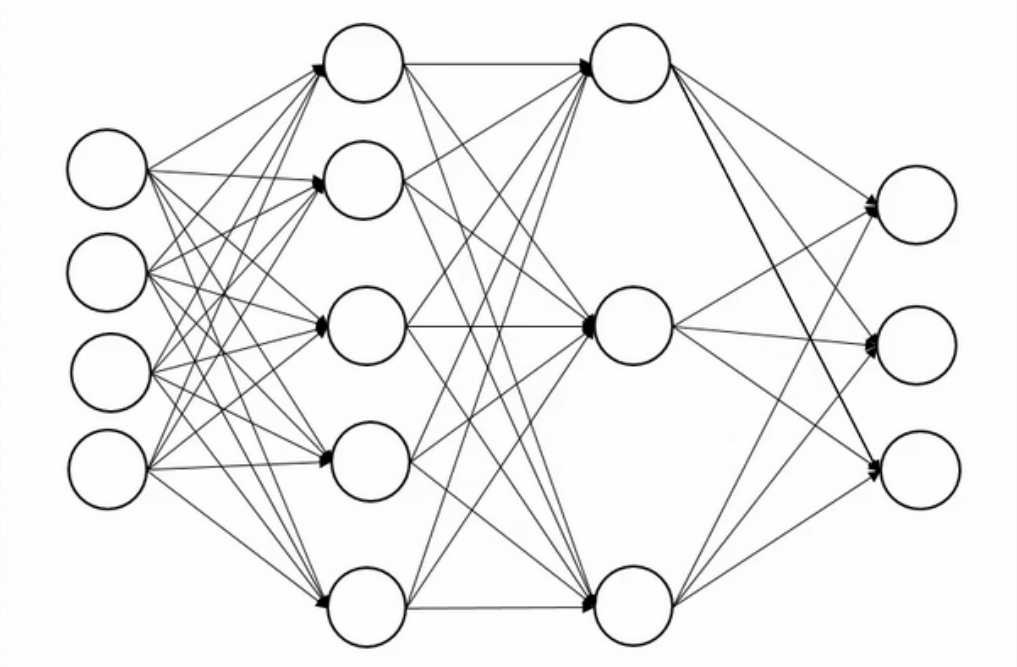
Dropout

During training, randomly remove neurons on a layer with probability p.



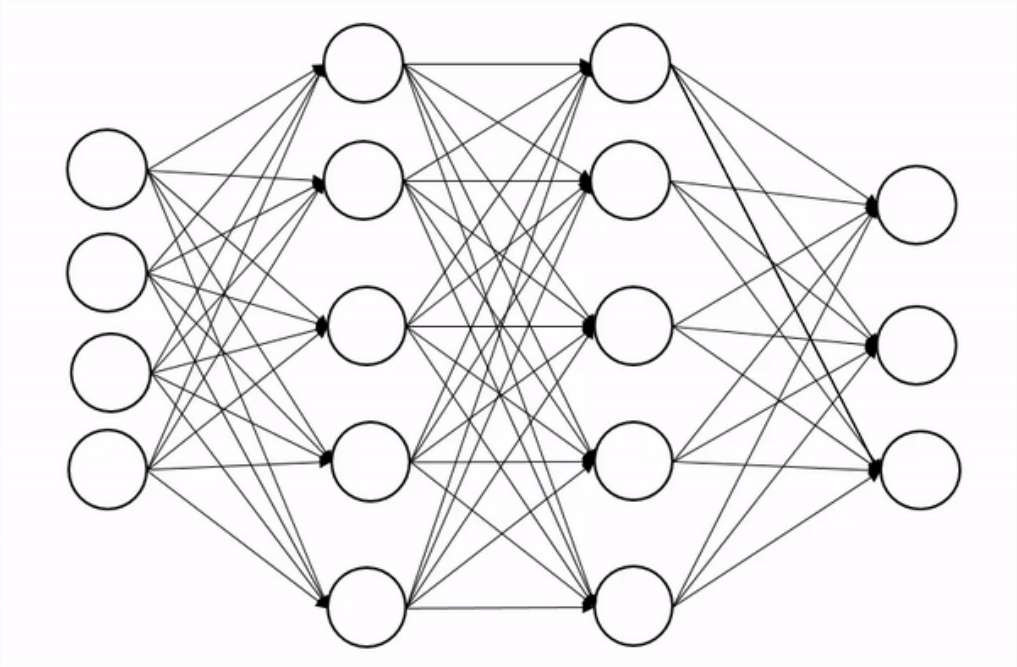
Dropout

During training, randomly remove neurons on a layer with probability p.



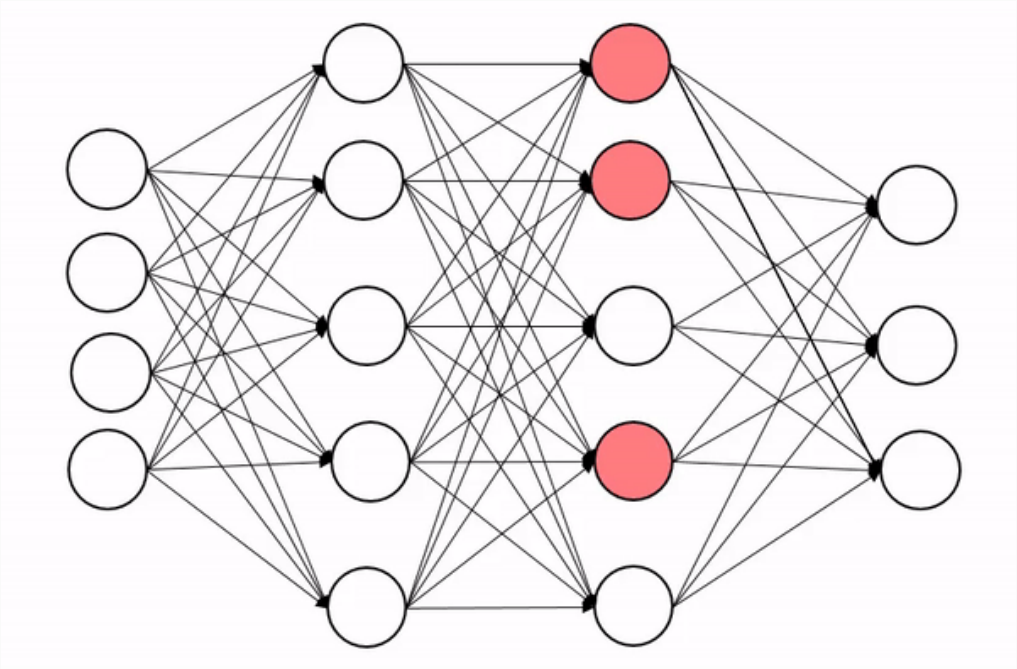
Dropout

During training, randomly remove neurons on a layer with probability p.



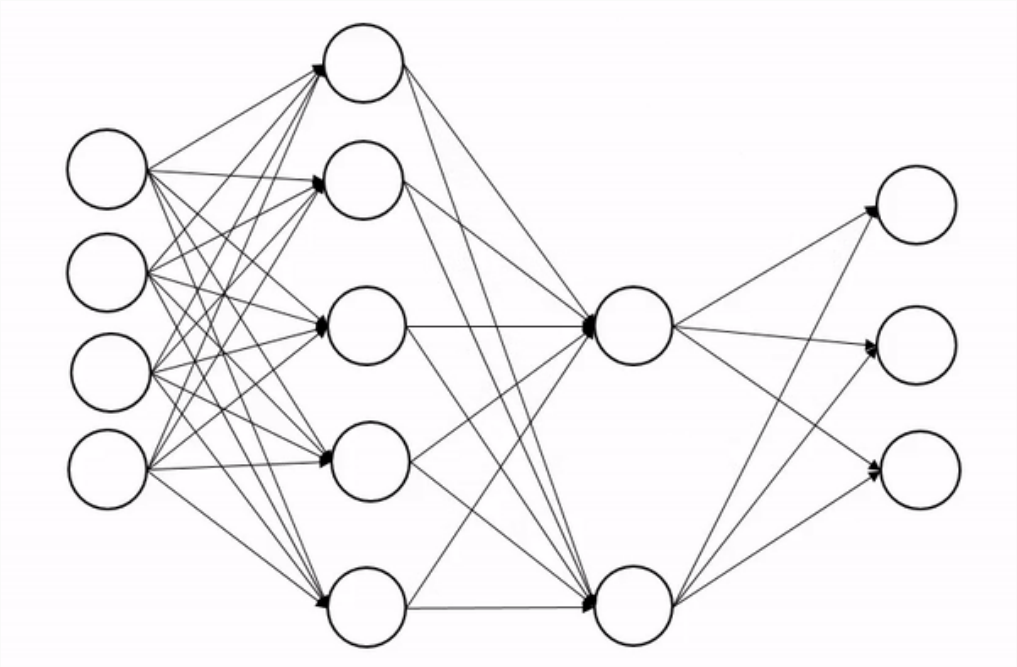
Dropout

During training, randomly remove neurons on a layer with probability p.



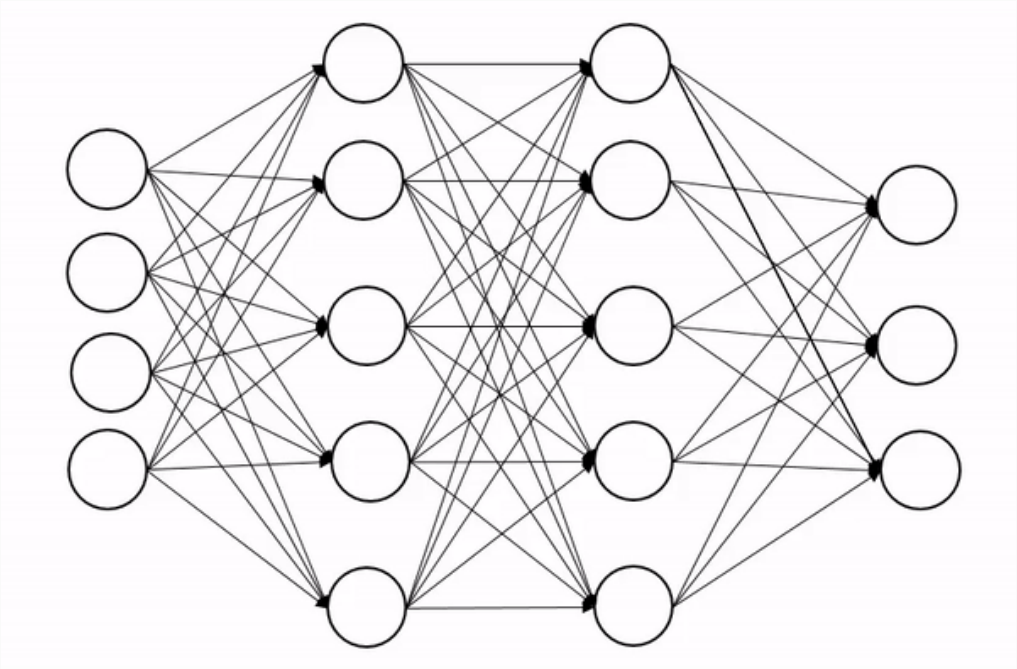
Dropout

During training, randomly remove neurons on a layer with probability p.



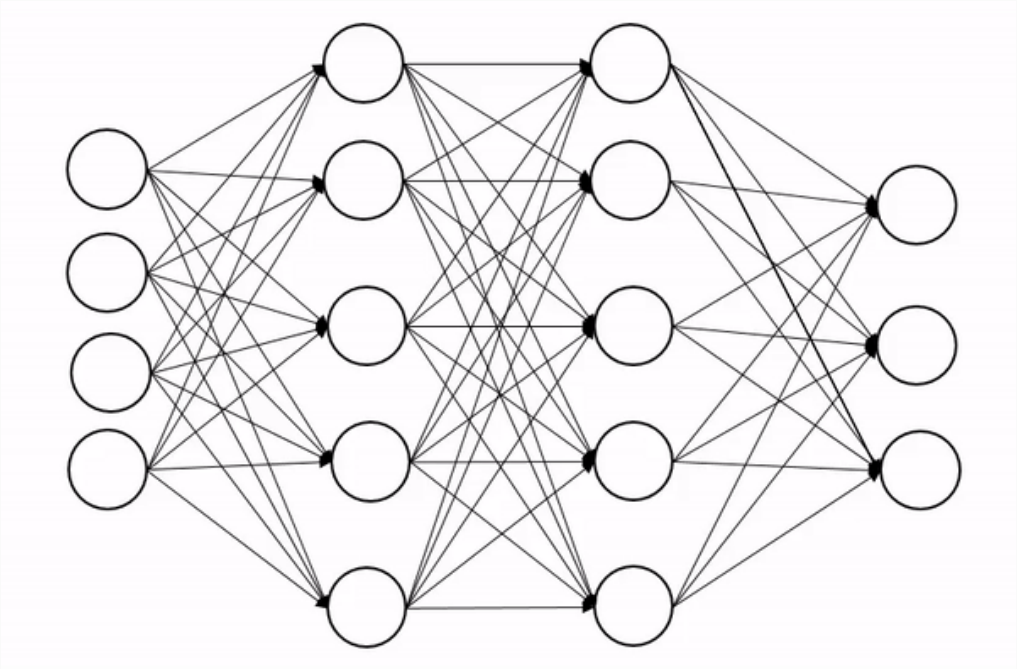
Dropout

During training, randomly remove neurons on a layer with probability p.



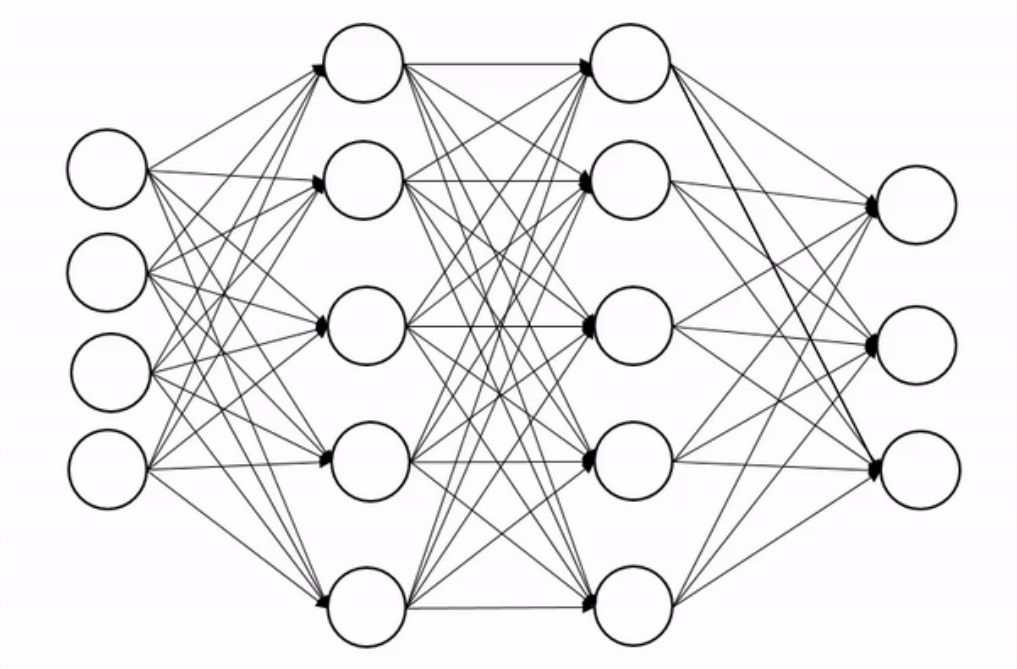
Dropout

During training, randomly remove neurons on a layer with probability p.



Dropout

During training, randomly remove neurons on a layer with probability p.



Practical remarks:

- Avoid Dropout on convolutive layer
- Avoid Dropout on the last layer.

Batch normalization

From *Ioffe et al. 2015, Batch normalizaion...*
Batch Normalization is a new type of layer.

If we use mini-batch training with a minibatch of size m:

Mini Batch Layer:

Input: Values of $\mathbf{x_{1...m}}$
Input: Initial parameters to be optimized: γ, β
Output: $\mathbf{z_i} = \mathrm{BN}_{\gamma,\beta}(\mathbf{x}_i)$

$\mu \leftarrow \frac{1}{m}\sum_{i=1}^{m} \mathbf{x}_i$ ▷ mini-batch mean
$\sigma^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (\mathbf{x}_i - \mu)^2$ ▷ (mini-batch variance)
$\hat{\mathbf{x}}_i \leftarrow (\mathbf{x}_i - \mu)/\sqrt{\sigma^2 + \epsilon}$ ▷ normalize
$\mathbf{z_i} = \gamma\hat{\mathbf{x}}_i + \beta$ ▷ Scale and shift
return $\mathbf{z_i}$

μ and σ^2 are non-trainable parameters. They are fixed for inferring new result (in test/validation).

Link with data assimilation

Data Assimilation

Example of BLUE: Best Linear Unbiased Estimator

Given a state vector $\mathbf{x} \in \mathbb{R}^n$ and a data vector $\mathbf{d} \in \mathbb{R}^m$:

$$\mathbf{x}^{\mathrm{f}} = \mathbf{x}^{\mathrm{t}} + \mathbf{p}, \qquad \overline{\mathbf{p}} = 0, \qquad \overline{\mathbf{p}\mathbf{p}^T} = \mathbf{C}_{xx}.$$

$$\mathbf{d} = \mathbf{H}\mathbf{x}^{\mathrm{t}} + \boldsymbol{\epsilon}, \qquad \overline{\boldsymbol{\epsilon}} = 0, \qquad \overline{\boldsymbol{\epsilon}\boldsymbol{\epsilon}^T} = \mathbf{C}_{\epsilon\epsilon}.$$

Data Assimilation

Example of BLUE: Best Linear Unbiased Estimator

Given a state vector $\mathbf{x} \in \mathbb{R}^n$ and a data vector $\mathbf{d} \in \mathbb{R}^m$:

$$\mathbf{x}^{\mathrm{f}} = \mathbf{x}^{\mathrm{t}} + \mathbf{p}, \qquad \overline{\mathbf{p}} = 0, \qquad \overline{\mathbf{p}\mathbf{p}^T} = \mathbf{C}_{xx}.$$

$$\mathbf{d} = \mathbf{H}\mathbf{x}^{\mathrm{t}} + \boldsymbol{\epsilon}, \qquad \overline{\boldsymbol{\epsilon}} = 0, \qquad \overline{\boldsymbol{\epsilon}\boldsymbol{\epsilon}^T} = \mathbf{C}_{\epsilon\epsilon}.$$

Estimating $\mathbf{x}^{\mathrm{t}}$ by minimizing the estimation error leads to minizing the following function:

$$\mathcal{J}(\mathbf{x}) = (\mathbf{d} - \mathbf{H}\mathbf{x})^T \mathbf{C}_{\epsilon\epsilon}^{-1} (\mathbf{d} - \mathbf{H}\mathbf{x}) + (\mathbf{x} - \mathbf{x}^{\mathrm{f}})^T \mathbf{C}_{xx}^{-1} (\mathbf{x} - \mathbf{x}^{\mathrm{f}})$$

Data Assimilation

Example of BLUE: Best Linear Unbiased Estimator

Given a state vector $\mathbf{x} \in \mathbb{R}^n$ and a data vector $\mathbf{d} \in \mathbb{R}^m$:

$$\mathbf{x}^{\mathrm{f}} = \mathbf{x}^{\mathrm{t}} + \mathbf{p}, \qquad \overline{\mathbf{p}} = 0, \qquad \overline{\mathbf{p}\mathbf{p}^T} = \mathbf{C}_{xx}.$$

$$\mathbf{d} = \mathbf{H}\mathbf{x}^{\mathrm{t}} + \boldsymbol{\epsilon}, \qquad \overline{\boldsymbol{\epsilon}} = 0, \qquad \overline{\boldsymbol{\epsilon}\boldsymbol{\epsilon}^T} = \mathbf{C}_{\epsilon\epsilon}.$$

Estimating $\mathbf{x}^{\mathrm{t}}$ by minimizing the estimation error leads to minizing the following function:

$$\mathcal{J}(\mathbf{x}) = (\mathbf{d} - \mathbf{H}\mathbf{x})^T \mathbf{C}_{\epsilon\epsilon}^{-1} (\mathbf{d} - \mathbf{H}\mathbf{x}) + (\mathbf{x} - \mathbf{x}^{\mathrm{f}})^T \mathbf{C}_{xx}^{-1} (\mathbf{x} - \mathbf{x}^{\mathrm{f}})$$

This is data assimilation!

We correct a forecast $\mathbf{x}^{\mathrm{f}}$ given some observational data $\mathbf{d}$

Is it machine learning?

- Ridge regression: $J(\boldsymbol{\theta}) = (\mathbf{y} - h_{\boldsymbol{\theta}}(\mathbf{x}))^T(\mathbf{y} - h_{\boldsymbol{\theta}}(\mathbf{x})) + \alpha\theta^T\theta$
- BLUE: $\mathcal{J}(\mathbf{x}) = (\mathbf{d} - \mathbf{H}\mathbf{x})^T\mathbf{C}_{\epsilon\epsilon}^{-1}(\mathbf{d} - \mathbf{H}\mathbf{x}) + (\mathbf{x} - \mathbf{x}^{\mathrm{f}})^T\mathbf{C}_{xx}^{-1}(\mathbf{x} - \mathbf{x}^{\mathrm{f}})$

Is it machine learning?

- Ridge regression: $J(\boldsymbol{\theta}) = (\mathbf{y} - h_{\boldsymbol{\theta}}(\mathbf{x}))^T(\mathbf{y} - h_{\boldsymbol{\theta}}(\mathbf{x})) + \alpha\theta^T\theta$
- BLUE: $\mathcal{J}(\mathbf{x}) = (\mathbf{d} - \mathbf{H}\mathbf{x})^T\mathbf{C}_{\epsilon\epsilon}^{-1}(\mathbf{d} - \mathbf{H}\mathbf{x}) + (\mathbf{x} - \mathbf{x}^{\mathrm{f}})^T\mathbf{C}_{xx}^{-1}(\mathbf{x} - \mathbf{x}^{\mathrm{f}})$

BLUE	Ridge
data $\mathbf{d}$	target $\mathbf{y}$
Observation operator $\mathbf{H}$	feature $\mathbf{x}$
State $\mathbf{x}$	parameters $\boldsymbol{\theta}$
$\mathbf{C}_{\epsilon\epsilon}\mathbf{C}_{xx}^{-1}$	α

Is it machine learning?

- Ridge regression: $J(\boldsymbol{\theta}) = (\mathbf{y} - h_{\boldsymbol{\theta}}(\mathbf{x}))^T(\mathbf{y} - h_{\boldsymbol{\theta}}(\mathbf{x})) + \alpha\theta^T\theta$
- BLUE: $\mathcal{J}(\mathbf{x}) = (\mathbf{d} - \mathbf{H}\mathbf{x})^T\mathbf{C}_{\epsilon\epsilon}^{-1}(\mathbf{d} - \mathbf{H}\mathbf{x}) + (\mathbf{x} - \mathbf{x}^{\mathrm{f}})^T\mathbf{C}_{xx}^{-1}(\mathbf{x} - \mathbf{x}^{\mathrm{f}})$

BLUE	Ridge
data $\mathbf{d}$	target $\mathbf{y}$
Observation operator $\mathbf{H}$	feature $\mathbf{x}$
State $\mathbf{x}$	parameters $\boldsymbol{\theta}$
$\mathbf{C}_{\epsilon\epsilon}\mathbf{C}_{xx}^{-1}$	α

Further links...

If a numerical model is integrated over several time steps, it can be related to successive layers of a Neural Network.

Probabilistic interpretation

Maximum likelihood estimator and loss

We can assume that the observation y follows a Gaussian law:

$$p(y/x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp -\frac{(y-\mu(x))^2}{2\sigma^2},$$

where x is observed and $\mu(x)$ is a function of x.
Given a set of samples $(x_k, y_k)_{1:K}$, the negative log-likelihood is defined by

$$L = \sum_{k=1}^{K} \left(\frac{\log 2\pi\sigma^2}{2} + \frac{(y_k - \mu(x_k))^2}{2\sigma^2} \right)$$

Minimizing L is maximizing the probability of having the observations y_k given x_k.

Loss function of a neural net

First case: σ is constant

$\mu(x)$ is parametrized by a neural net $G_\mu(x, \boldsymbol{\theta}_\mu)$
The Maximum likelihood estimator is found by minizming:

$$L(\boldsymbol{\theta}_\mu) = \sum_{k=1}^{K} (y_k - G_\mu(x, \boldsymbol{\theta}_\mu))^2$$

which is exactly the regression loss already introduced.

Loss function of a neural net

First case: σ is constant

$\mu(x)$ is parametrized by a neural net $G_\mu(x, \boldsymbol{\theta}_\mu)$
The Maximum likelihood estimator is found by minizming:

$$L(\boldsymbol{\theta}_\mu) = \sum_{k=1}^{K} (y_k - G_\mu(x, \boldsymbol{\theta}_\mu))^2$$

which is exactly the regression loss already introduced.

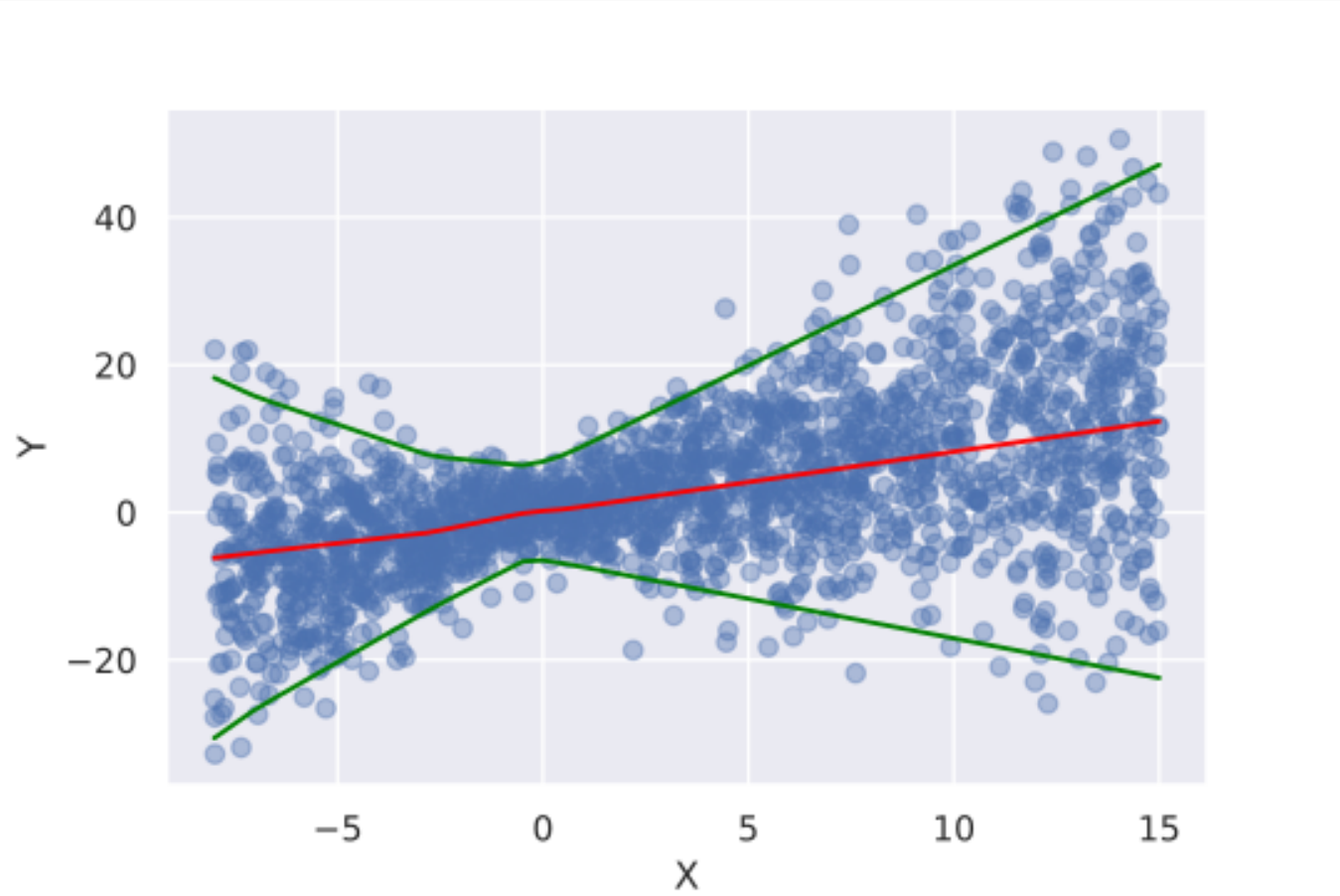
Second case: $\sigma(x)$ is a function of x.

In addition to $G_\mu(x, \boldsymbol{\theta}_\mu)$, $\sigma(x)$ is parametrized by $G_\sigma(x, \boldsymbol{\theta}_\sigma)$. The loss to minimize is then:

$$L(\boldsymbol{\theta}_\mu, \boldsymbol{\theta}_\sigma) = \sum_{k=1}^{K} \left(\frac{\log 2\pi\, G_\sigma(x_k, \boldsymbol{\theta}_\sigma)^2}{2} + \frac{(y_k - G_\mu(x_k, \boldsymbol{\theta}_\mu))^2}{2 G_\sigma(x_k, \boldsymbol{\theta}_\sigma)^2} \right)$$

The neural net $G = (G_\mu, G_\sigma)$ gives also the uncertainty of its estimation in the form of the standard deviation.

Illustration



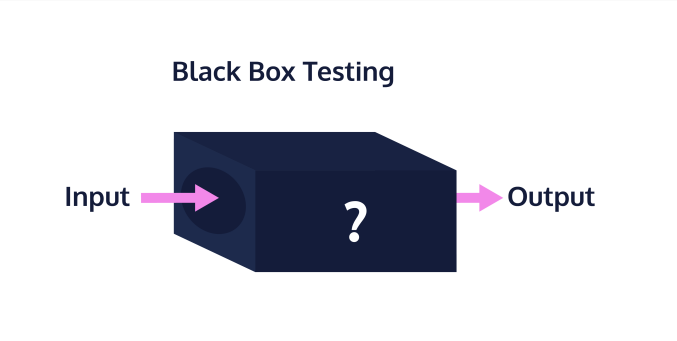
In red the estimation of the mean $G_\mu(x, \boldsymbol{\theta}_\mu)$

In green the confidence interval $G_\mu(x, \boldsymbol{\theta}_\mu) \pm \sigma(x_k, \boldsymbol{\theta}_\sigma)$

A black box?

The black box paradigm

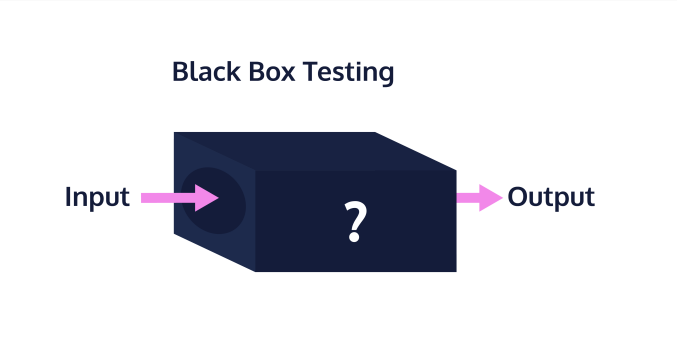
The machine-learning based model is a black box. It gives some results but we don't understand how.



Do we need to understand the model?

The black box paradigm

The machine-learning based model is a black box. It gives some results but we don't understand how.



Do we need to understand the model?

"Every time I fire a linguist, the performance of our speech recognition system goes up."

F. Jelinek, 1988

Frederick Jelinek 1932-2010

Beyond the black box

Motivation

- Build models that can be trusted
- Use other source of knowledge (e.g. physical properties) when data are not sufficient.

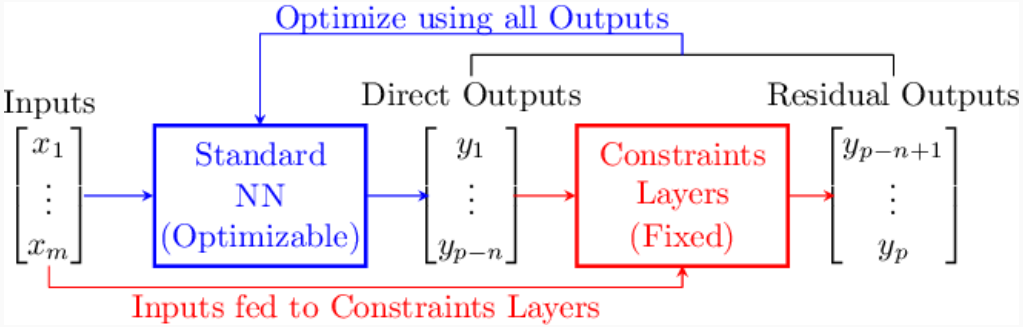
Two directions:

- Add physical constraints to ML models
- Explainable/Transparent ML.

Add physical constraints to ML models

- Simple example: enforce the positivity of some quantities (e.g. concentration)
- More complex: enforce conservation laws

  Beucler, T., Pritchard, M., Rasp, S., Ott, J., Baldi, P. and Gentine, P., 2021. Enforcing analytic constraints in neural networks emulating physical systems. *Physical Review Letters*, 126(9), p.098302.



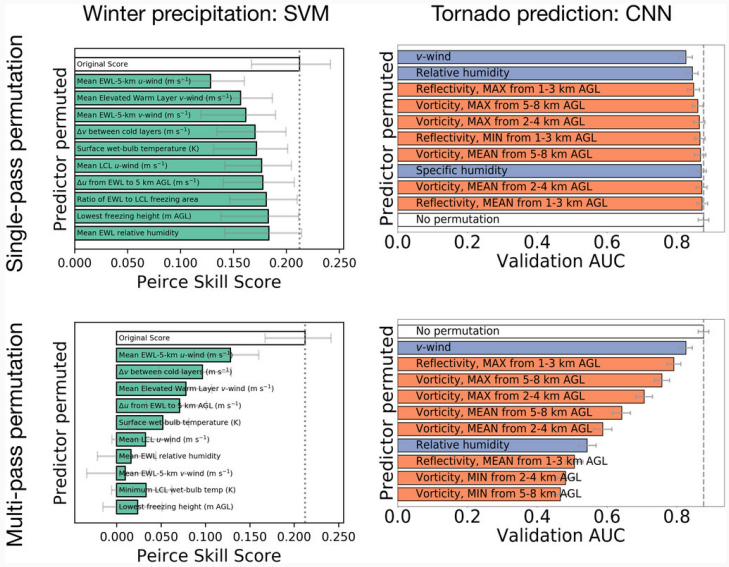
Beucler et al.

Explainable/Transparent ML.

The objective is to understand how the machine learning makes a prediction (e.g. which feature is important for the prediction).

McGovern et al., 2019, Making the black box more transparent: Understanding the physical implications of machine learning. *Bulletin of the American Meteorological Society*

Sonnewald et al.,2021. Bridging observation, theory and numerical simulation of the ocean using Machine Learning. *Environ. Res. Lett.*



McGovern et al.