Part 4: Neural networks

julien.brajard@nersc.no

October 2021

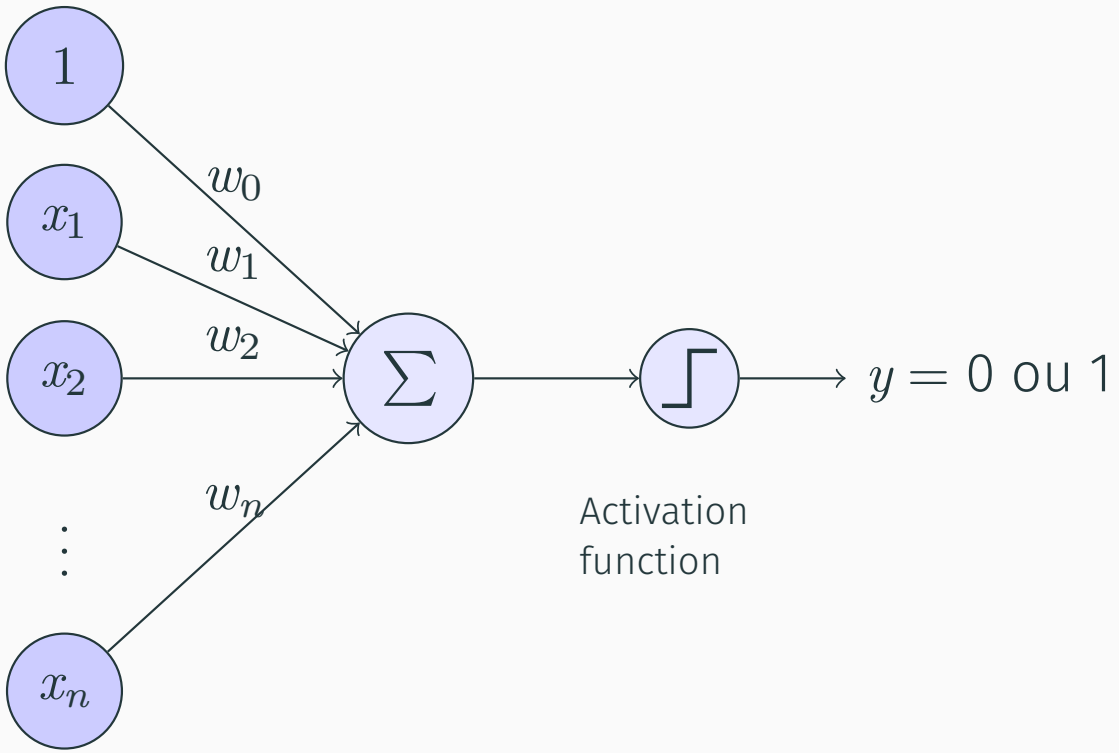
NERSC

slides+notebook: *https://github.com/nansencenter/nersc_ml_course*

Table of contents i

Neural Networks

The perceptron : an artificial neuron



Computation

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

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

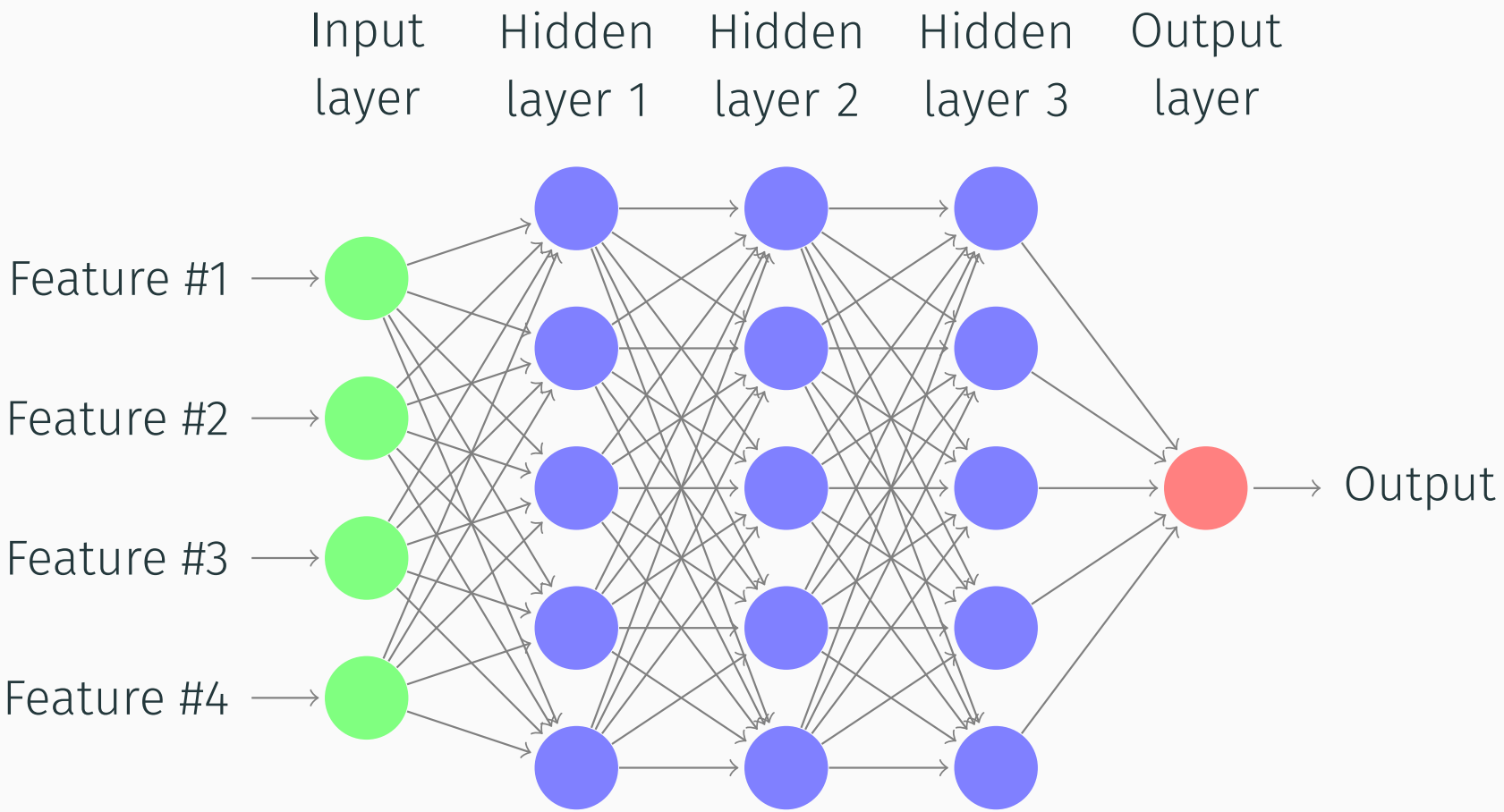
- Inputs x_i are the different features of the data
- Weight w_i are the parameters of the model to optimize
- If the activation function is identity, it is equivalent to a linear regression

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

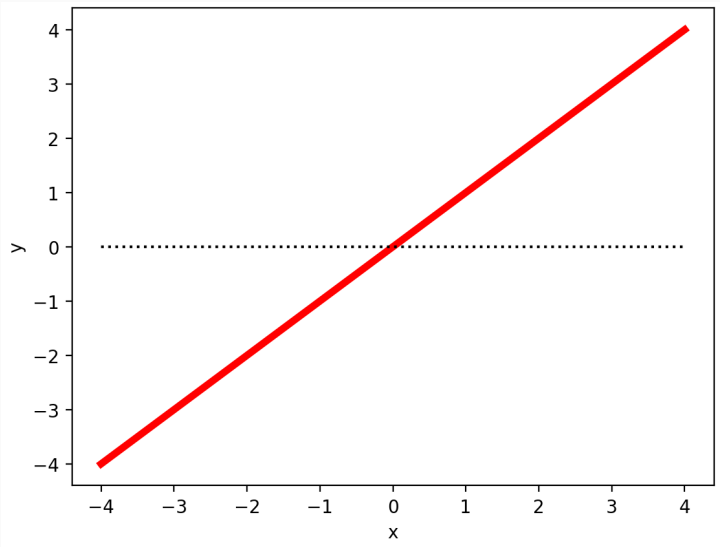
More complexe models are build by combining several perceptrons

Multi-layer perceptron (Densely connected layers)

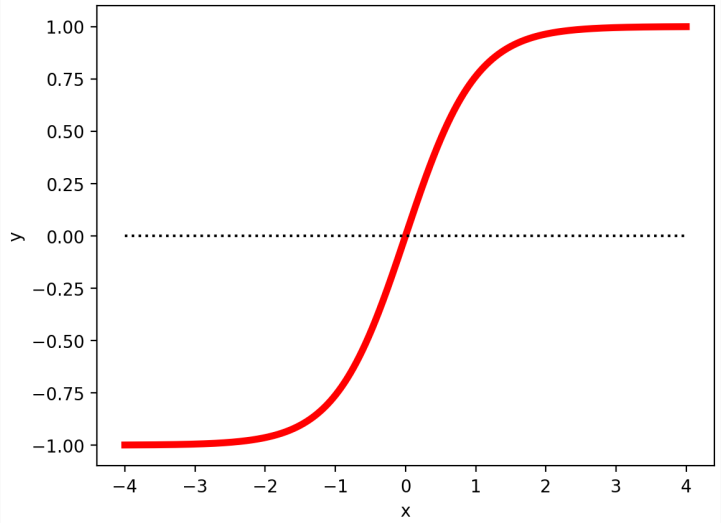


Most usual activation functions

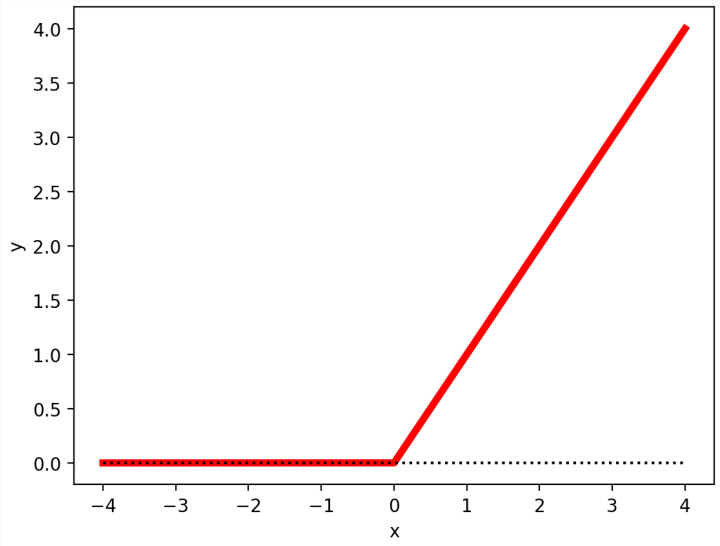
Linear



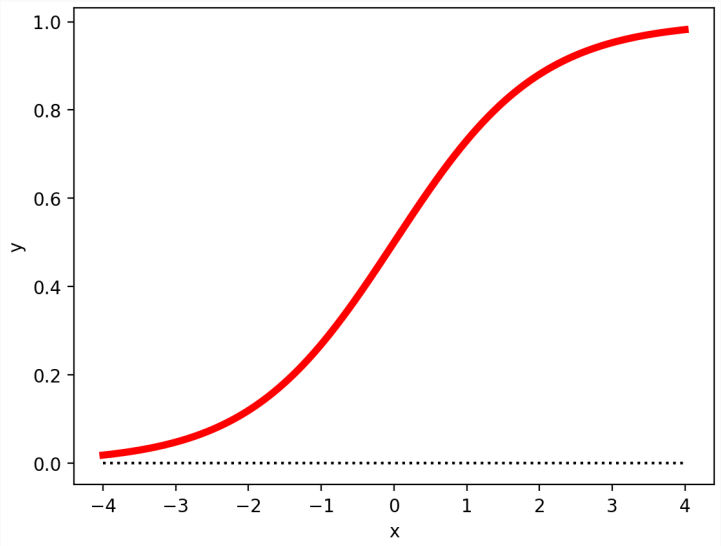
Hyperbolic tangent



ReLU

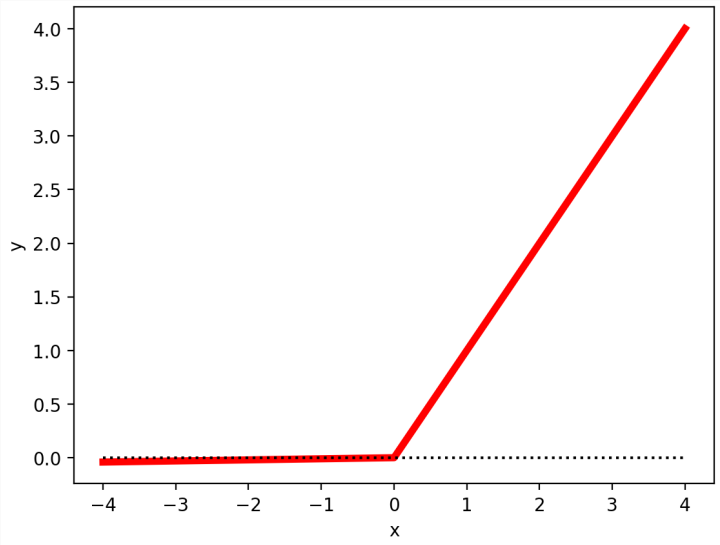


Sigmoid

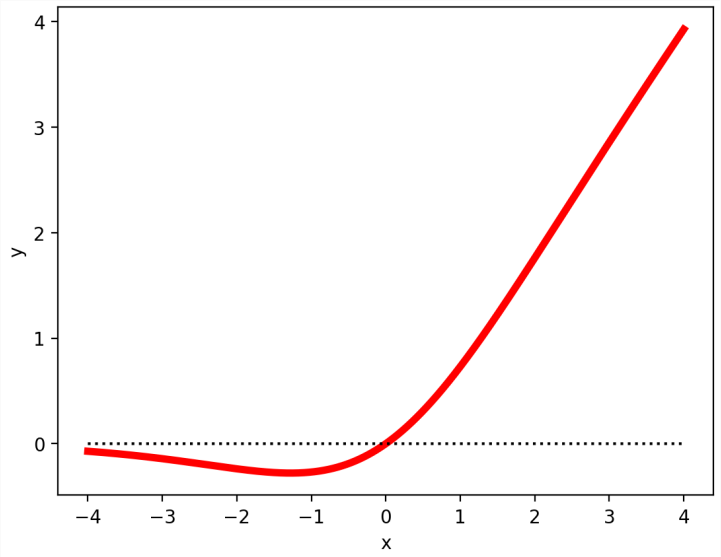


New fancy activation functions

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

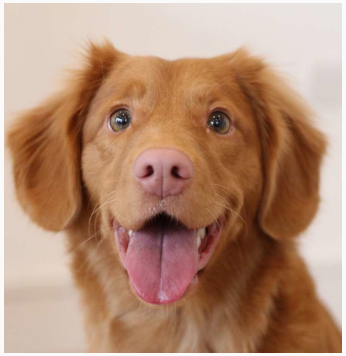
$$L(p, y) = -\sum_i \sum_j y_{i,j}. \log p_{i,j}$$

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:

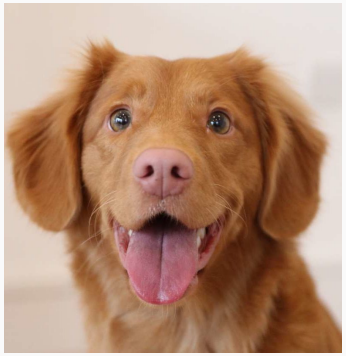
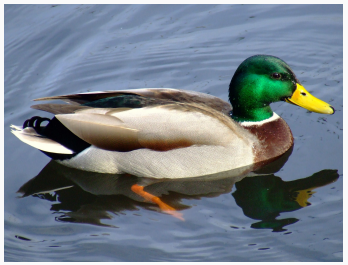
$$L = -\sum y_i \cdot \log(f(x_i)) + (1 - y_i) \cdot \log(1 - f(x_i))$$

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

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

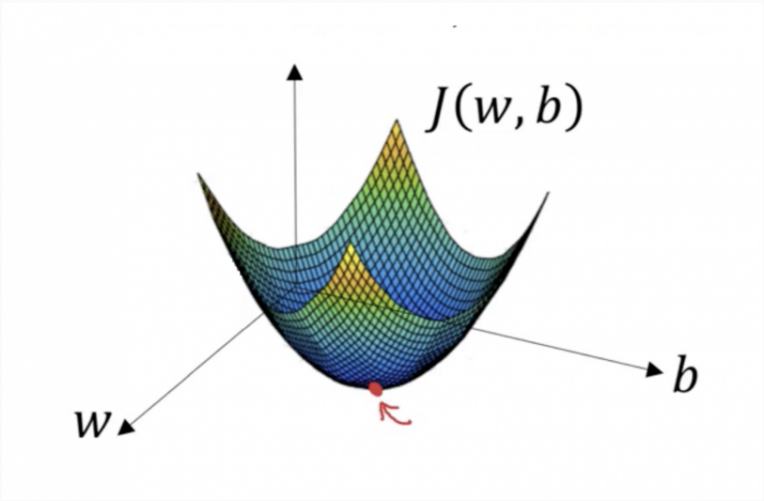
Optimization using gradient descent

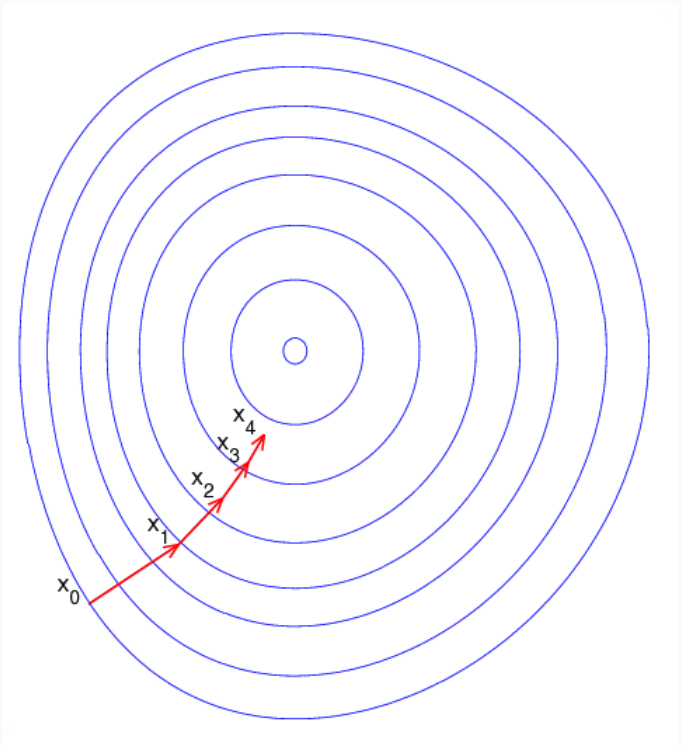
What is gradient descent

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:

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

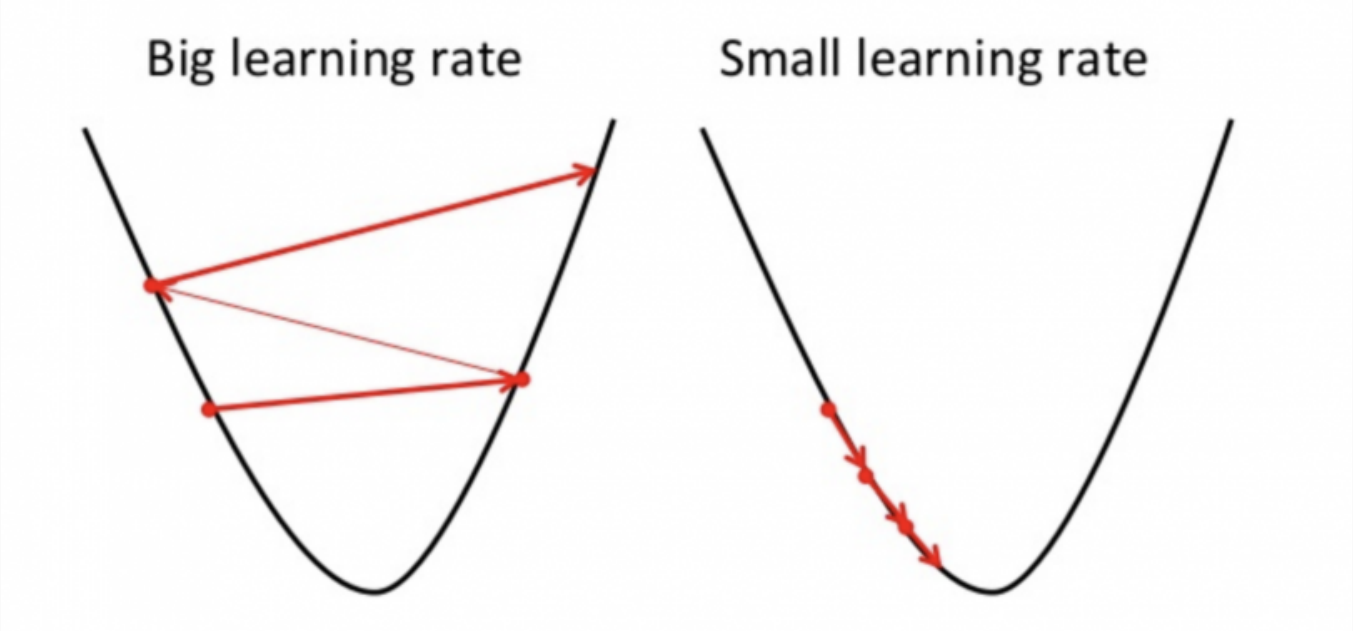
γ is called the learning rate.

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

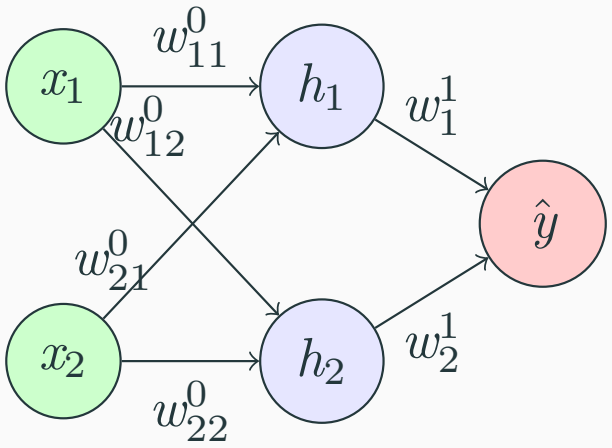
- The learning γ is an important hyperparameter that needs to be tuned using, e.g random search

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

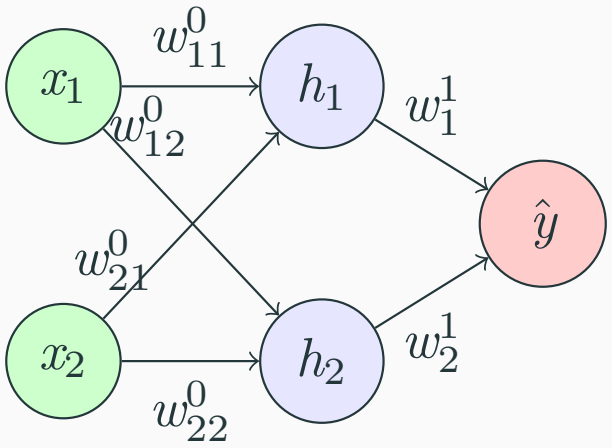


1. Given a couple (x, y)

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

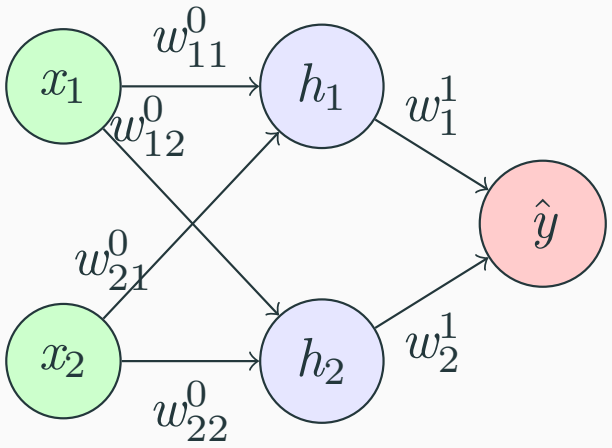


1. Given a couple (x, y)
2. Forward computation:
   $h_j = f_0(\sum_{i=1}^{2} w^0_{ij}.x_i)$
   $\hat{y} = f_1(\sum_{j=1}^{2} w^1_j.h_j)$

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

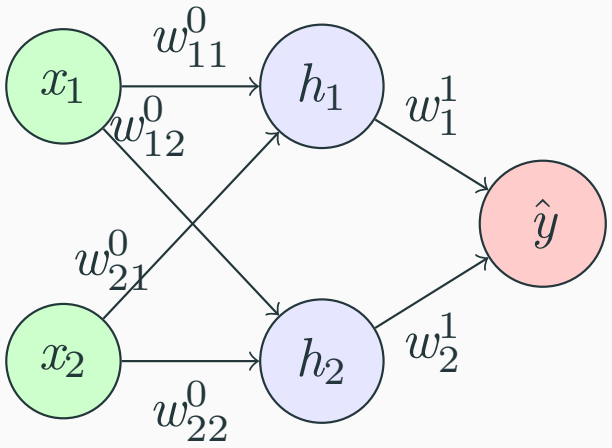


1. Given a couple (x, y)
2. Forward computation:
   $h_j = f_0(\sum_{i=1}^{2} w^0_{ij}.x_i)$
   $\hat{y} = f_1(\sum_{j=1}^{2} w^1_j.h_j)$
3. Compute the gradient of the loss:
   $\boxed{\partial L/\partial \hat{y}}$

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

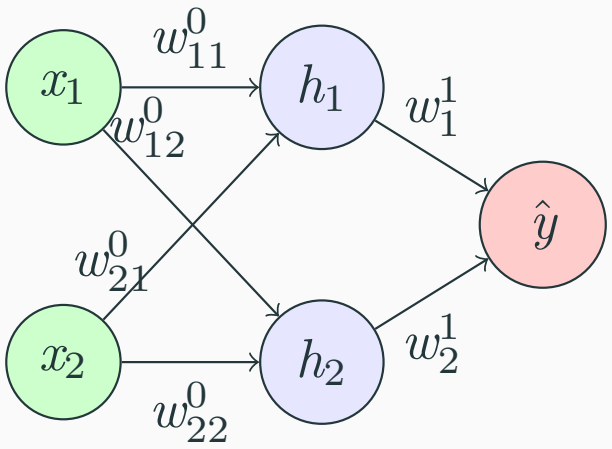


Objective

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

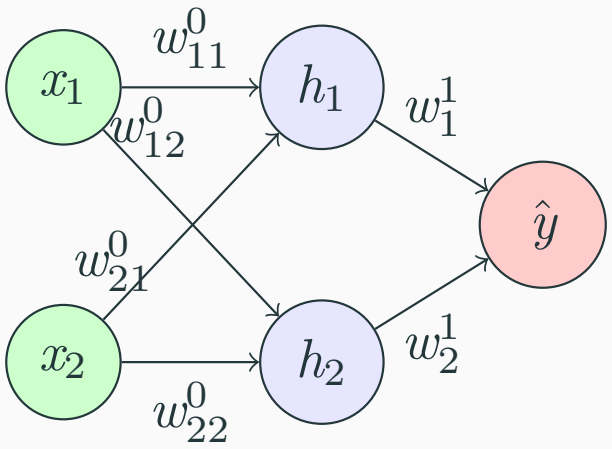


Objective

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 adjustig 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