

# Fundaments of Machine learning for and with engineering applications

Enrico Riccardi<sup>1</sup>

Department of Mathematics and Physics, University of Stavanger (UiS).<sup>1</sup>

Sep 17, 2025



1 Loss function

2 Regularization

3 Neural Network -NN-

4 Optimization

# Loss function

One of the main core concepts behind machine learning is the **loss function**.

In mathematical optimization and decision theory, a loss function or cost function (sometimes also called an error function) is a function that maps an event or values of one or more variables onto a real number intuitively representing some "cost" associated with the event. [Wiki]

## Loss function in Machine Learning

It quantifies the difference between the predicted outputs of a machine learning algorithm and the actual target values.

# Loss function

One of the main core concepts behind machine learning is the **loss function**.

In mathematical optimization and decision theory, a loss function or cost function (sometimes also called an error function) is a function that maps an event or values of one or more variables onto a real number intuitively representing some "cost" associated with the event. [Wiki]

## Loss function in Machine Learning

It quantifies the difference between the predicted outputs of a machine learning algorithm and the actual target values.

# Loss function aims

It provides a set of core quantifications/possibilities:

- ① Performance measurement: it provides the metric of the prediction performances
- ② Direction for improvement: it allows the identification of convergent solutions
- ③ Balancing bias and variance: it allows to account for artefact in the sampling phase
- ④ Influencing model behaviour: it allows to bridge data driven methods with mathematics/physics

# Loss function aims

It provides a set of core quantifications/possibilities:

- ① Performance measurement: it provides the metric of the prediction performances
- ② Direction for improvement: it allows the identification of convergent solutions
- ③ Balancing bias and variance: it allows to account for artefact in the sampling phase
- ④ Influencing model behaviour: it allows to bridge data driven methods with mathematics/physics

# Loss function aims

It provides a set of core quantifications/possibilities:

- ① Performance measurement: it provides the metric of the prediction performances
- ② Direction for improvement: it allows the identification of convergent solutions
- ③ Balancing bias and variance: it allows to account for artefact in the sampling phase
- ④ Influencing model behaviour: it allows to bridge data driven methods with mathematics/physics

# Loss function aims

It provides a set of core quantifications/possibilities:

- ① Performance measurement: it provides the metric of the prediction performances
- ② Direction for improvement: it allows the identification of convergent solutions
- ③ Balancing bias and variance: it allows to account for artefact in the sampling phase
- ④ Influencing model behaviour: it allows to bridge data driven methods with mathematics/physics

# Loss function aims

It provides a set of core quantifications/possibilities:

- ① Performance measurement: it provides the metric of the prediction performances
- ② Direction for improvement: it allows the identification of convergent solutions
- ③ Balancing bias and variance: it allows to account for artefact in the sampling phase
- ④ Influencing model behaviour: it allows to bridge data driven methods with mathematics/physics

# Most common loss function in machine learning

## Most popular entries:

- ① Mean Square Error (regression): Fast computations, good convergence.
- ② Mean Absolute Error (regression): No focus on the outliers, poor convergence.
- ③ Entropy Loss (classification): measures the difference between the model probability distribution outcomes and the predicted values

# Most common loss function in machine learning

Most popular entries:

- ① Mean Square Error (regression): Fast computations, good convergence.
- ② Mean Absolute Error (regression): No focus on the outliers, poor convergence.
- ③ Entropy Loss (classification): measures the difference between the model probability distribution outcomes and the predicted values

# Most common loss function in machine learning

Most popular entries:

- ① Mean Square Error (regression): Fast computations, good convergence.
- ② Mean Absolute Error (regression): No focus on the outliers, poor convergence.
- ③ Entropy Loss (classification): measures the difference between the model probability distribution outcomes and the predicted values

# Most common loss function in machine learning

Most popular entries:

- ① Mean Square Error (regression): Fast computations, good convergence.
- ② Mean Absolute Error (regression): No focus on the outliers, poor convergence.
- ③ Entropy Loss (classification): measures the difference between the model probability distribution outcomes and the predicted values

# Losses

Mean absolute error, L1 loss

$$MAE = \frac{1}{n} \sum_{i=1}^n (|y_i - \bar{y}|)$$

Mean square error, L2 loss

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \bar{y})^2$$

Log Loss for binary classification

$$\text{Log Loss} = -[y \log(f(x)) + (1 - y) \log(1 - f(x))]$$

# MSE in Vanilla Python

```
def mean_squared_error(observed, predicted):
    """ Compute the mean squared error.

    Params:
    -----
    observed : list,
        The observed values.
    predicted : list,
        The predicted values.

    Output:
    -----
    sys[0] : float,
        Mean squared error.

    """
    if len(observed) != len(predicted):
        raise ValueError(
            "The lengths of input lists are" +
            f"not equal {len(observed)} {len(predicted)}.")  

    # Initialize the sum of squared errors
    sum_squared_error = 0
```

# MSE in Vanilla Python

```
sum_squared_error = 0

# Loop through all observations
for obs, pred in zip(observed, predicted):
    # Calculate the square difference, and add it to the sum
    sum_squared_error += (obs - pred) ** 2

# Calculate the mean squared error
mse = sum_squared_error / len(observed)

return mse
```

# MSE in Numpy Python

```
import numpy as np

def mean_squared_error(observed, predicted):
    observed_np = np.array(observed)
    predicted_np = np.array(predicted)
    mse = np.mean((observed_np - predicted_np) ** 2)
    return mse
```

# MSE in Python

```
from mse_vanilla import mean_squared_error as vanilla_mse
from mse_numpy import mean_squared_error as numpy_mse
import sklearn.metrics as sk

observed = [2, 4, 6, 8]
predicted = [2.5, 3.5, 5.5, 7.5]

mse_vanilla = vanilla_mse(observed, predicted)
print("Mean Squared Error, vanilla : ", mse_vanilla)

mse_numpy = numpy_mse(observed, predicted)
print("Mean Squared Error, numpy : ", mse_numpy)

sk_mse = sk.mean_squared_error(observed, predicted)
print("Mean Squared Error, sklearn : ", sk_mse)

assert(mse_vanilla == mse_numpy == sk_mse)
```

## MSE in benchmark

```
from mse_vanilla import mean_squared_error as vanilla_mse
from mse_numpy import mean_squared_error as numpy_mse
import sklearn.metrics as sk
import timeit as it

observed = [2, 4, 6, 8]
predicted = [2.5, 3.5, 5.5, 7.5]

mse_vanilla = vanilla_mse(observed, predicted)
time_v = it.timeit('vanilla_mse(observed, predicted)',
                   globals=globals(), number=10) / 100
mse_numpy = numpy_mse(observed, predicted)
time_np = it.timeit('numpy_mse(observed, predicted)',
                     globals=globals(), number=10) / 100
sk_mse = sk.mean_squared_error(observed, predicted)
time_sk = it.timeit('sk.mean_squared_error(observed, predicted)',
                     globals=globals(), number=10) / 100

for mse, mse_type, time in zip([mse_vanilla, mse_numpy, sk_mse],
                               ['vanilla', 'numpy', 'sklearn'],
                               [time_v, time_np, time_sk]):
    print(f"Mean Squared Error, {mse_type}: {mse},"
          f"Average execution time: {time} seconds")

assert(mse_vanilla == mse_numpy == sk_mse)
```

# Coding MSE

```
from mse_vanilla import mean_squared_error as vanilla_mse
from mse_numpy import mean_squared_error as numpy_mse
from sklearn.metrics import mean_squared_error as sk_mse
import timeit as it

observed = [2, 4, 6, 8]
predicted = [2.5, 3.5, 5.5, 7.5]

karg = {'observed': observed, 'predicted': predicted}

factory = {'mse_vanilla' : vanilla_mse,
           'mse_numpy' : numpy_mse,
           # 'mse_sk' : sk_mse
           }

for talker, worker in factory.items():
    exec_time = it.timeit('{worker(**karg)}',
                          globals=globals(), number=100) / 100
    mse = worker(**karg)
    print(f"Mean Squared Error, {talker} :", mse,
          f"Average execution time: {exec_time} seconds")
```

1 Loss function

2 Regularization

3 Neural Network -NN-

4 Optimization

# Regularization

Conventionally, MSE is used as loss function:

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \bar{y})^2$$

We might want to force our model not fit so well the data, i.e. we *penalise* the model.

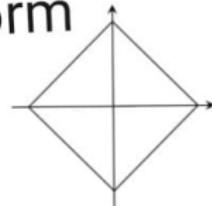
$$L1\_loss\_regularized = MSE + \lambda \sum_{i=1}^m |b_i|$$

$$L2\_loss\_regularized = MSE + \lambda \sum_{i=1}^m b_i^2$$

$$Elastic\ Net = MSE + \lambda_1 \sum_{i=1}^m |b_i| + \lambda_2 \sum_{i=1}^m b_i^2$$

# Norms

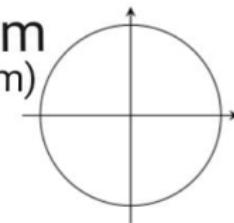
1-norm



$$p = 1$$



2-norm  
(Euclidean norm)



$$p = 2$$



# Outcomes



# Regularization

$\lambda$  is the regularisation strength. The larger value it has, the more the target function is "wrong". For  $\lambda = 0$ , no regularisation is imposed.

Forcing a model to be "wrong" might sound rather counter-intuitive. Furthermore, we have introduced one more parameter that has not much meaning?!

Bias - variance trade off

$$\text{Error} = \text{bias}^2 + \text{variance} + \text{statistical error}$$

# Regularization

$\lambda$  is the regularisation strength. The larger value it has, the more the target function is "wrong". For  $\lambda = 0$ , no regularisation is imposed.

Forcing a model to be "wrong" might sound rather counter-intuitive. Furthermore, we have introduced one more parameter that has not much meaning?!

## Bias - variance trade off

$$\text{Error} = \text{bias}^2 + \text{variance} + \text{statistical error}$$

# Regularization

$\lambda$  is the regularisation strength. The larger value it has, the more the target function is "wrong". For  $\lambda = 0$ , no regularisation is imposed.

Forcing a model to be "wrong" might sound rather counter-intuitive. Furthermore, we have introduced one more parameter that has not much meaning?!

## Bias - variance trade off

$$\text{Error} = \text{bias}^2 + \text{variance} + \text{statistical error}$$

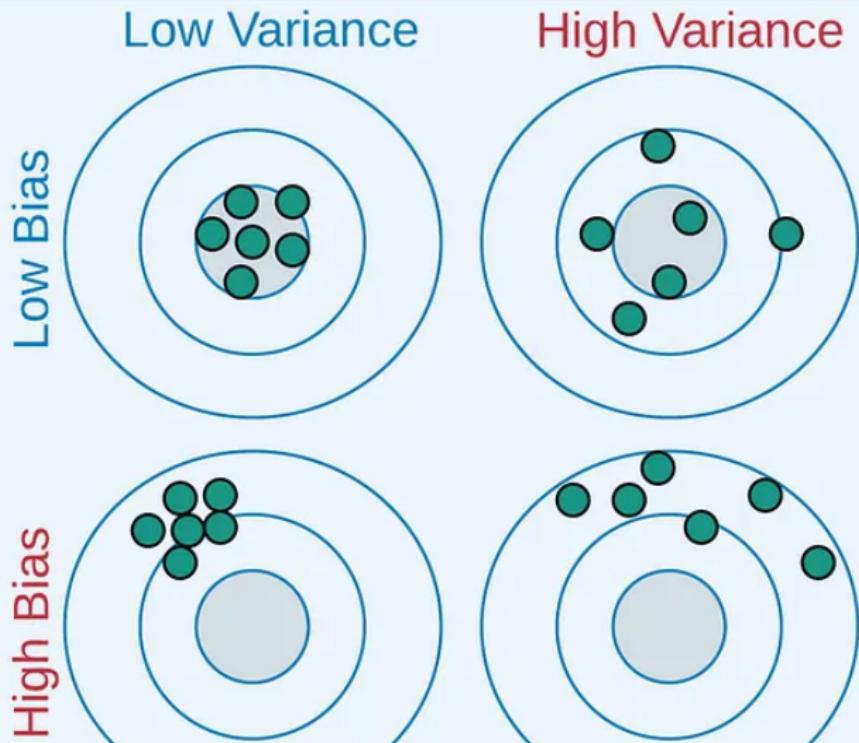
# Bias

Bias is the difference of the average value of predictions ( $q$ ) from the true relationship function ( $f$ ):

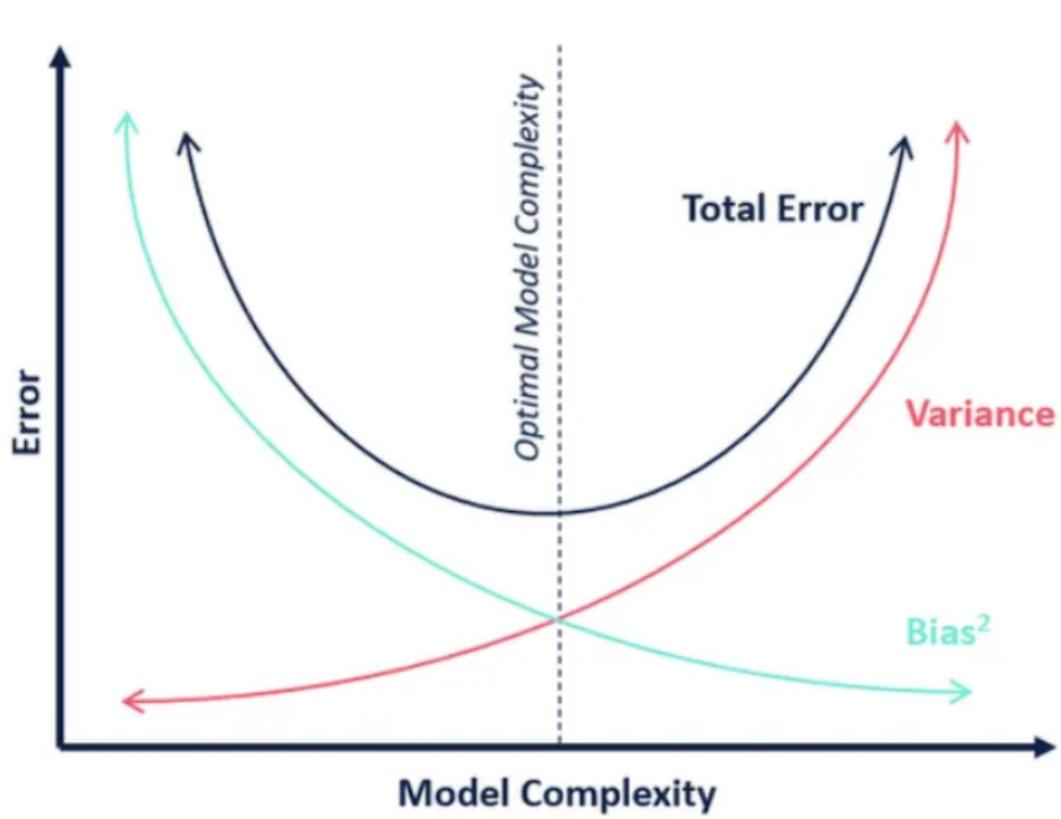
$$bias[q(x)] = \mathbb{E}[q(x)] - f(x)$$

Variance is the expectation of the squared deviation of  $q(x)$  from its expected value  $\mathbb{E}[q(x)]$ .

$$var[q(x)] = \mathbb{E}[(q(x) - \mathbb{E}[q(x)])^2]$$



# Bias-Variance



1 Loss function

2 Regularization

3 Neural Network -NN-

4 Optimization

# Neural Network -NN-

One of the most famous models in Machine Learning is **Neural Network**.

The name comes from how the brain functions: a set of connected neurons that are either off or active.

In its essence,

NN is a large set of (linear) regressions executed both in parallel and in series.

Conventional NN are composed by:

- ① Input layer
- ② Hidden layers
- ③ Output layers

which, together, can approximately approximate any function in any dimension.

# Neural Network -NN-

One of the most famous models in Machine Learning is **Neural Network**.

The name comes from how the brain functions: a set of connected neurons that are either off or active.

In its essence,

NN is a large set of (linear) regressions executed both in parallel and in series.

Conventional NN are composed by:

- ① Input layer
- ② Hidden layers
- ③ Output layers

which, together, can approximately approximate any function in any dimension.

# Neural Network -NN-

One of the most famous models in Machine Learning is **Neural Network**.

The name comes from how the brain functions: a set of connected neurons that are either off or active.

In its essence,

NN is a large set of (linear) regressions executed both in parallel and in series.

Conventional NN are composed by:

- ① Input layer
- ② Hidden layers
- ③ Output layers

which, together, can approximately approximate any function in any dimension.

# Neural Network -NN-

One of the most famous models in Machine Learning is **Neural Network**.

The name comes from how the brain functions: a set of connected neurons that are either off or active.

In its essence,

NN is a large set of (linear) regressions executed both in parallel and in series.

Conventional NN are composed by:

- ① Input layer
- ② Hidden layers
- ③ Output layers

which, together, can approximately approximate any function in any dimension.

# Neural Network -NN-

One of the most famous models in Machine Learning is **Neural Network**.

The name comes from how the brain functions: a set of connected neurons that are either off or active.

In its essence,

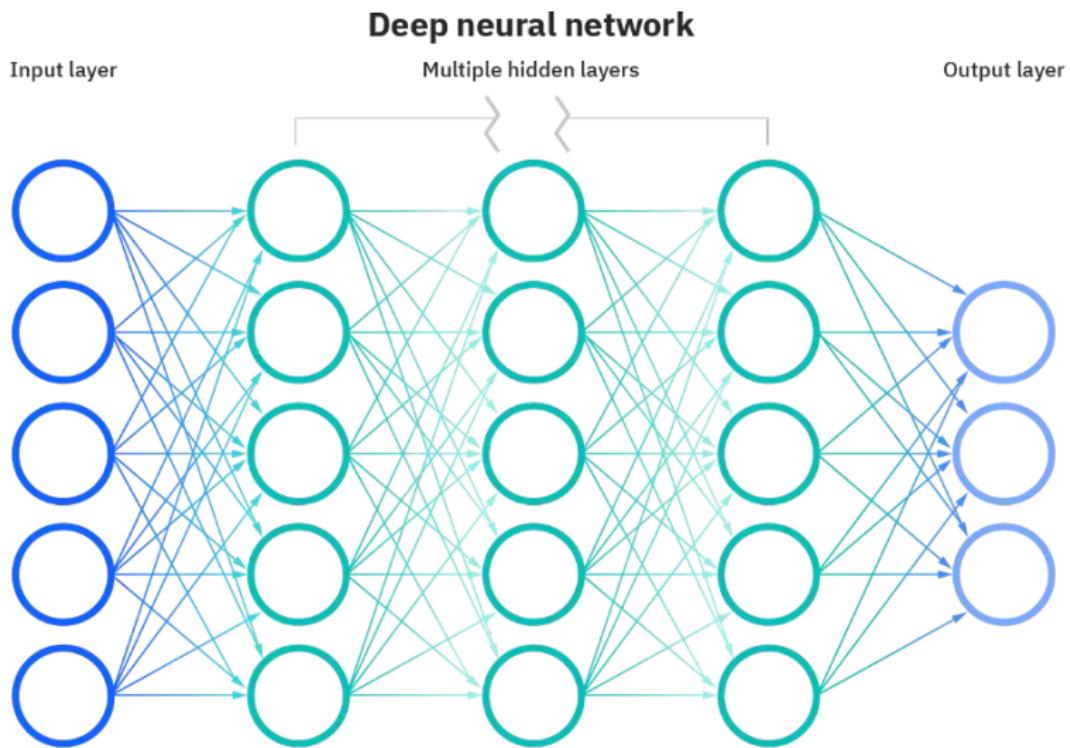
NN is a large set of (linear) regressions executed both in parallel and in series.

Conventional NN are composed by:

- ① Input layer
- ② Hidden layers
- ③ Output layers

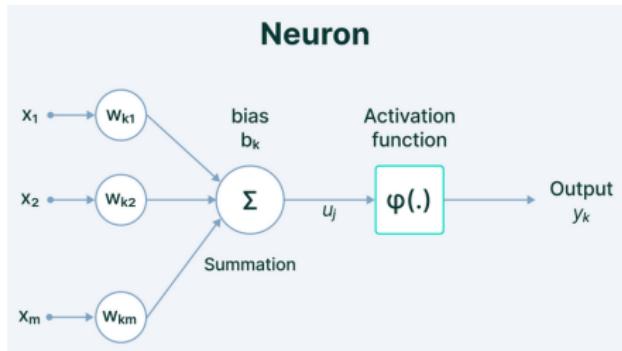
which, together, can approximately approximate any function in any dimension.

# deep Neural Network



# How do they work?

Each node is called **neuron**

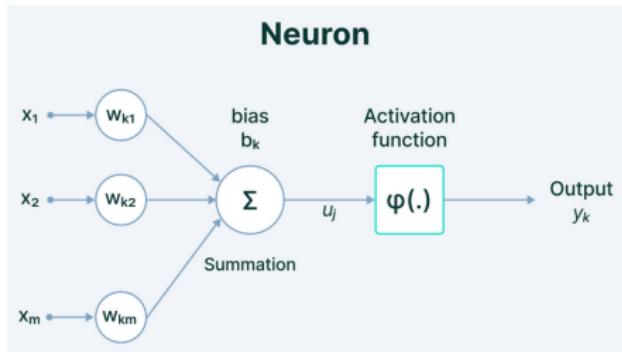


Use many of these, many times, and you have builded a NN!

- It is really just a  $Y = f(X)$ , where  $w_i$  and  $b_i$  are the unknowns to solve for.
- As in linear regression, this becomes mainly a number of numerical recipes.
- As the problem's dimensionality is significant, several strategies have been developed.

# How do they work?

Each node is called **neuron**



Use many of these, many times, and you have builded a NN!

- It is really just a  $Y = f(X)$ , where  $w_i$  and  $b_i$  are the unknowns to solve for.
- As in linear regression, this becomes mainly a number of numerical recipes.
- As the problem's dimensionality is significant, several strategies have been developed.

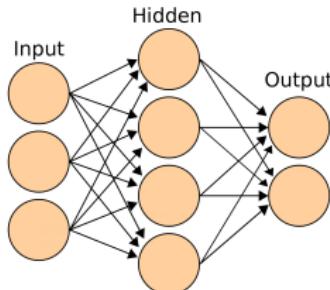
# How do they work?

- ① We thus need also an **activation function**.
- ② The **NN structure**
- ③ Also, we might need to specify the number of **epochs**.

Using NN is essentially running a **simulation campaign**: a set of numerical experiments

Please say yes

Have you already heard of experimental design?



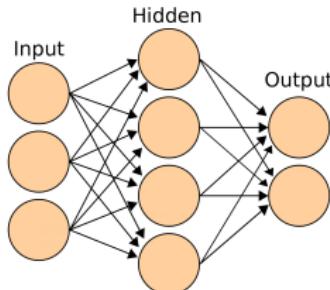
# How do they work?

- ① We thus need also an **activation function**.
- ② The **NN structure**
- ③ Also, we might need to specify the number of **epochs**.

Using NN is essentially running a simulation campaign: a set of numerical experiments

Please say yes

Have you already heard of experimental design?

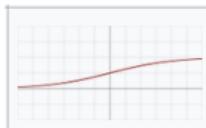


# Activation function

Each node applies an activation function on the weighted sum of the previous nodes.

Such functions are called activation functions. There are MANY!

The most popular has been the logistic



$$\sigma(x) \doteq \frac{1}{1 + e^{-x}}$$

$$g(x)(1 - g(x))$$

Now the most common is ReLU (Rectified linear unit)



$$(x)^+ \doteq \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases} = \max(0, x) = x\mathbf{1}_{x>0}$$

$$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \end{cases}$$

# Activation function

Each node applies an activation function on the weighted sum of the previous nodes.

Such functions are called activation functions. There are MANY!

The most popular has been the logistic



$$\sigma(x) \doteq \frac{1}{1 + e^{-x}}$$

$$g(x)(1 - g(x))$$

Now the most common is ReLU (Rectified linear unit)



$$(x)^+ \doteq \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases} = \max(0, x) = x\mathbf{1}_{x>0}$$

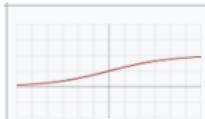
$$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \end{cases}$$

# Activation function

Each node applies an activation function on the weighted sum of the previous nodes.

Such functions are called activation functions. There are MANY!

The most popular has been the logistic



$$\sigma(x) \doteq \frac{1}{1 + e^{-x}}$$

$$g(x)(1 - g(x))$$

Now the most common is ReLU (Rectified linear unit)



$$(x)^+ \doteq \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases} = \max(0, x) = x\mathbf{1}_{x>0}$$

$$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \end{cases}$$

# NN Architecture

A mostly complete chart of

# Neural Networks

©2016 Fjodor van Veen - asimovinstitute.org

- (○) Backfed Input Cell
- (●) Input Cell
- (△) Noisy Input Cell
- (●) Hidden Cell
- (○) Probabilistic Hidden Cell
- (△) Spiking Hidden Cell
- (●) Output Cell
- (○) Match Input Output Cell
- (●) Recurrent Cell
- (○) Memory Cell
- (△) Different Memory Cell
- (●) Kernel
- (○) Convolution or Pool

Perceptron (P)



Feed Forward (FF)



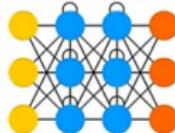
Radial Basis Network (RBF)



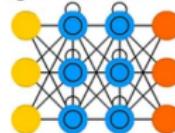
Deep Feed Forward (DFF)



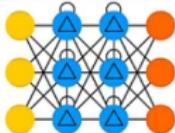
Recurrent Neural Network (RNN)



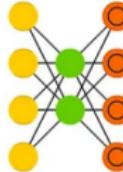
Long / Short Term Memory (LSTM)



Gated Recurrent Unit (GRU)



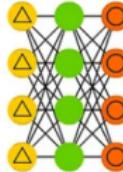
Auto Encoder (AE)



Variational AE (VAE)



Denoising AE (DAE)



Sparse AE (SAE)



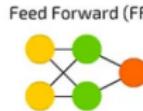
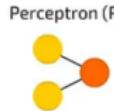
# NN Architecture

A mostly complete chart of

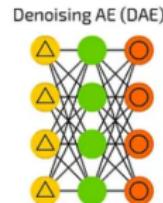
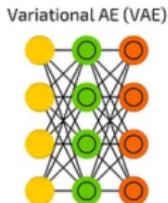
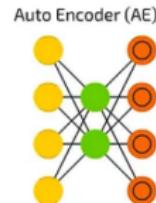
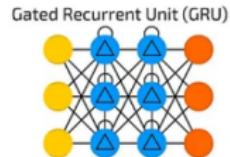
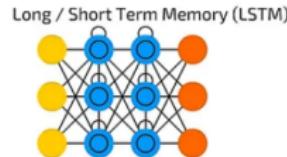
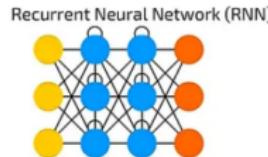
# Neural Networks

©2016 Fjodor van Veen - asimovinstitute.org

- (○) Backfed Input Cell
- (●) Input Cell
- (△) Noisy Input Cell
- (●) Hidden Cell
- (○) Probabilistic Hidden Cell
- (△) Spiking Hidden Cell
- (●) Output Cell
- (○) Match Input Output Cell
- (●) Recurrent Cell
- (○) Memory Cell
- (△) Different Memory Cell
- (●) Kernel
- (○) Convolution or Pool



Deep Feed Forward (DFF)



# NN Architecture

Markov Chain (MC)



Hopfield Network (HN)



Boltzmann Machine (BM)



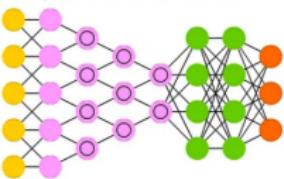
Restricted BM (RBM)



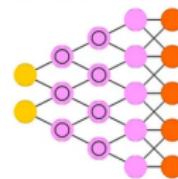
Deep Belief Network (DBN)



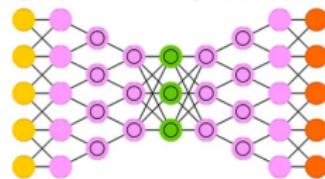
Deep Convolutional Network (DCN)



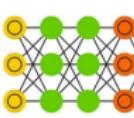
Deconvolutional Network (DN)



Deep Convolutional Inverse Graphics Network (DCIGN)



Generative Adversarial Network (GAN)



Liquid State Machine (LSM)



Extreme Learning Machine (ELM)



Echo State Network (ESN)



Deep Residual Network (DRN)



Kohonen Network (KN)



Support Vector Machine (SVM)



Neural Turing Machine (NTM)



# NN Architecture

**Neural net suffers from overfitting problems.**

**Even more parameters**

The number of nodes, the architecture, the number of hidden layer etc are NN **hyperparameters**

Unfortunately, at the current status of knowledge, the best architecture can be found only by trial and error.

The best fitting NN usually has a poor validation (generalizability).

NN is a very expensive approach and it should be used only if really needed.

# NN Architecture

Neural net suffers from overfitting problems.

## Even more parameters

The number of nodes, the architecture, the number of hidden layer etc are NN **hyperparameters**

Unfortunately, at the current status of knowledge, the best architecture can be found only by trial and error.

The best fitting NN usually has a poor validation (generalizability).

NN is a very expensive approach and it should be used only if really needed.

# NN Architecture

Neural net suffers from overfitting problems.

## Even more parameters

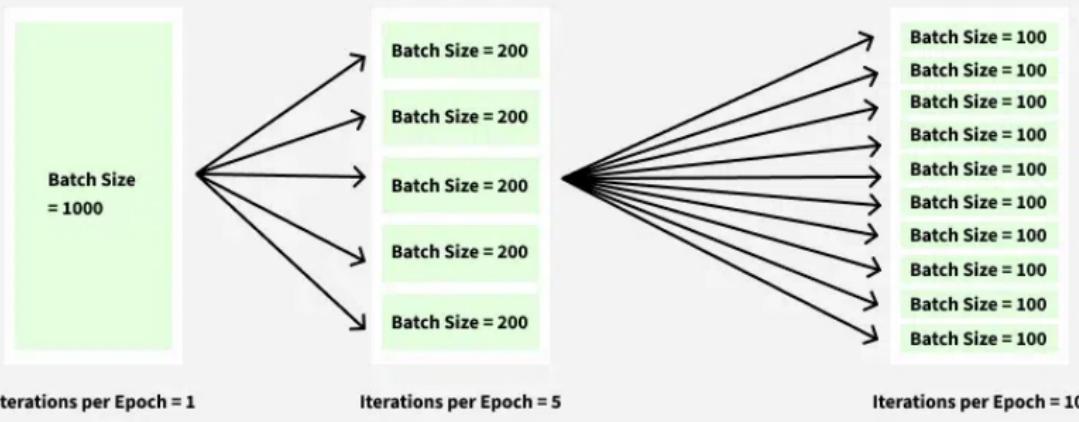
The number of nodes, the architecture, the number of hidden layer etc are NN **hyperparameters**

Unfortunately, at the current status of knowledge, the best architecture can be found only by trial and error.

The best fitting NN usually has a poor validation (generalizability).

NN is a very expensive approach and it should be used only if really needed.

## Epoch in Machine Learning



## Pro

- ① Better performance
- ② Progress tracking
- ③ Memory efficiency
- ④ Improved stopping criteria
- ⑤ More effective training

## Cons

- ① Overfitting risk
- ② Computational cost
- ③ One more hyperparameter

## Pro

- ① Better performance
- ② Progress tracking
- ③ Memory efficiency
- ④ Improved stopping criteria
- ⑤ More effective training

## Cons

- ① Overfitting risk
- ② Computational cost
- ③ One more hyperparameter

# NN types

There are many types of NN:

- ANN (artificial NN), just another name for NN
- DNN (deep) deep neural network
- RNN (recurrent NN) for audio
- CNN (convolutional NN) for images
- Autoencoder (for PCA) to compress to a latent space and decompress data
- Deep autoencoder (for interpretability)
- Physics informed NN (to merge NN to differential equations)
- ... and more ...

tensorflow, pytorch and keras are the most popular and popular libraries for NN

# NN types

There are many types of NN:

- ANN (artificial NN), just another name for NN
- DNN (deep) deep neural network
- RNN (recurrent NN) for audio
- CNN (convolutional NN) for images
- Autoencoder (for PCA) to compress to a latent space and decompress data
- Deep autoencoder (for interpretability)
- Physics informed NN (to merge NN to differential equations)
- ... and more ...

tensorflow, pytorch and keras are the most popular and popular libraries for NN

# NN types

There are many types of NN:

- ANN (artificial NN), just another name for NN
- DNN (deep) deep neural network
- RNN (recurrent NN) for audio
- CNN (convolutional NN) for images
- Autoencoder (for PCA) to compress to a latent space and decompress data
- Deep autoencoder (for interpretability)
- Physics informed NN (to merge NN to differential equations)
- ... and more ...

tensorflow, pytorch and keras are the most popular and popular libraries for NN

# NN types

There are many types of NN:

- ANN (artificial NN), just another name for NN
- DNN (deep) deep neural network
- RNN (recurrent NN) for audio
- CNN (convolutional NN) for images
- Autoencoder (for PCA) to compress to a latent space and decompress data
- Deep autoencoder (for interpretability)
- Physics informed NN (to merge NN to differential equations)
- ... and more ...

tensorflow, pytorch and keras are the most popular and popular libraries for NN

# NN types

There are many types of NN:

- ANN (artificial NN), just another name for NN
- DNN (deep) deep neural network
- RNN (recurrent NN) for audio
- CNN (convolutional NN) for images
- Autoencoder (for PCA) to compress to a latent space and decompress data
- Deep autoencoder (for interpretability)
- Physics informed NN (to merge NN to differential equations)
- ... and more ...

tensorflow, pytorch and keras are the most popular and popular libraries for NN

# NN types

There are many types of NN:

- ANN (artificial NN), just another name for NN
- DNN (deep) deep neural network
- RNN (recurrent NN) for audio
- CNN (convolutional NN) for images
- Autoencoder (for PCA) to compress to a latent space and decompress data
- Deep autoencoder (for interpretability)
- Physics informed NN (to merge NN to differential equations)
- ... and more ...

tensorflow, pytorch and keras are the most popular and popular libraries for NN.

# NN types

There are many types of NN:

- ANN (artificial NN), just another name for NN
- DNN (deep) deep neural network
- RNN (recurrent NN) for audio
- CNN (convolutional NN) for images
- Autoencoder (for PCA) to compress to a latent space and decompress data
- Deep autoencoder (for interpretability)
- Physics informed NN (to merge NN to differential equations)
- ... and more ...

tensorflow, pytorch and keras are the most popular and popular libraries for NN.

# NN types

There are many types of NN:

- ANN (artificial NN), just another name for NN
- DNN (deep) deep neural network
- RNN (recurrent NN) for audio
- CNN (convolutional NN) for images
- Autoencoder (for PCA) to compress to a latent space and decompress data
- Deep autoencoder (for interpretability)
- Physics informed NN (to merge NN to differential equations)
- ... and more ...

tensorflow, pytorch and keras are the most popular and popular libraries for NN.

# NN types

There are many types of NN:

- ANN (artificial NN), just another name for NN
- DNN (deep) deep neural network
- RNN (recurrent NN) for audio
- CNN (convolutional NN) for images
- Autoencoder (for PCA) to compress to a latent space and decompress data
- Deep autoencoder (for interpretability)
- Physics informed NN (to merge NN to differential equations)
- ... and more ...

tensorflow, pytorch and keras are the most popular and popular libraries for NN.

1 Loss function

2 Regularization

3 Neural Network -NN-

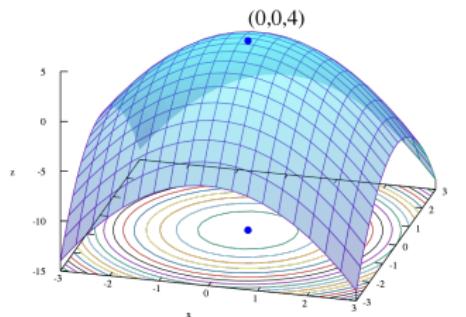
4 Optimization

# Optimization

Optimization is a field on its own.

## Definition

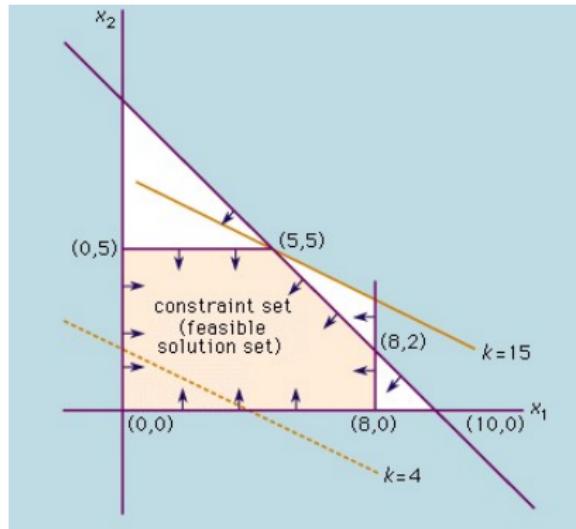
- Finding the "optimus" (latin), the best.
- Collection of mathematical principles and methods used for solving quantitative problems.



# Constraints

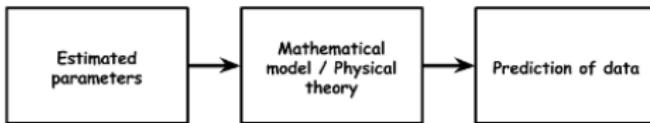
Not only we want to find the best solution, we also need to respect the constraints.

Finding the minimum and maxima of functions subjected to constraints.

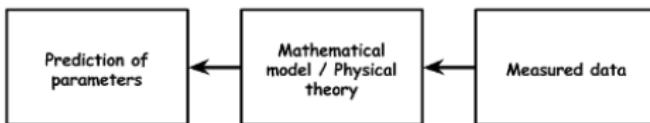


# Forward and backward methods

The forward problem



The inverse problem



Requirements:

- ① Existence
- ② Uniqueness
- ③ Stability
- ④ Efficiency

# Gradient descend

Let's reconsider a linear model:

$$\hat{y}_i = \sum_{j=0}^m x_{ij} b_j$$

Let's get back the residuals

$$R = e^T e$$

What we do is to calculate the derivative of the residuals in respect to the coefficients  $b_j$ .

and the ideal solution is when

$$\frac{\partial R}{\partial b_j} = 0 \quad \forall j \in [0, n]$$

# Gradient descend

Let's reconsider a linear model:

$$\hat{y}_i = \sum_{j=0}^m x_{ij} b_j$$

Let's get back the residuals

$$R = e^T e$$

What we do is to calculate the derivative of the residuals in respect to the coefficients  $b_j$ .

and the ideal solution is when

$$\frac{\partial R}{\partial b_j} = 0 \quad \forall j \in [0, n]$$

# Gradient descend

Let's reconsider a linear model:

$$\hat{y}_i = \sum_{j=0}^m x_{ij} b_j$$

Let's get back the residuals

$$R = e^T e$$

What we do is to calculate the derivative of the residuals in respect to the coefficients  $b_j$ .

and the ideal solution is when

$$\frac{\partial R}{\partial b_j} = 0 \quad \forall j \in [0, n]$$

# Gradient descent

ok, but what if we do not know

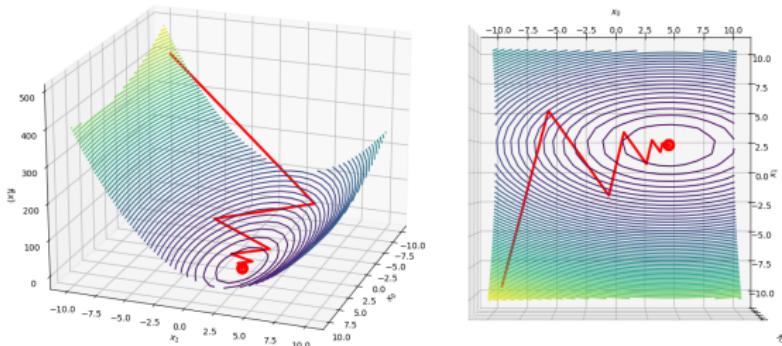
$$\frac{\partial R}{\partial b_j} = 0$$

We **hopefully** be able to numerically calculate  $\frac{\partial R}{\partial b_j}$

When  $f$  is convex:

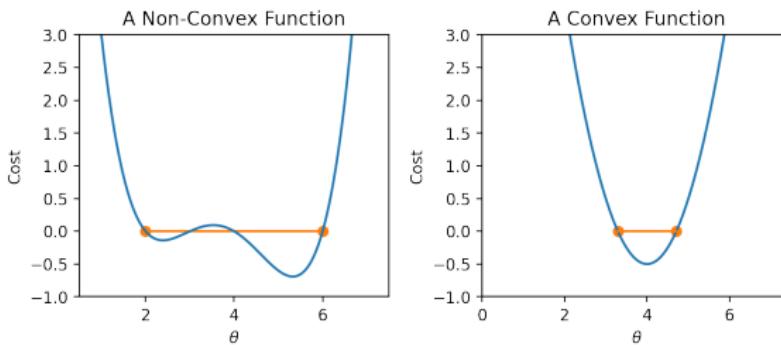
The learning rate  $\eta$  is used to move towards the global minimum:

$$a_{n+1} = a_n + \eta \nabla f(a_n)$$



# Gradient descent

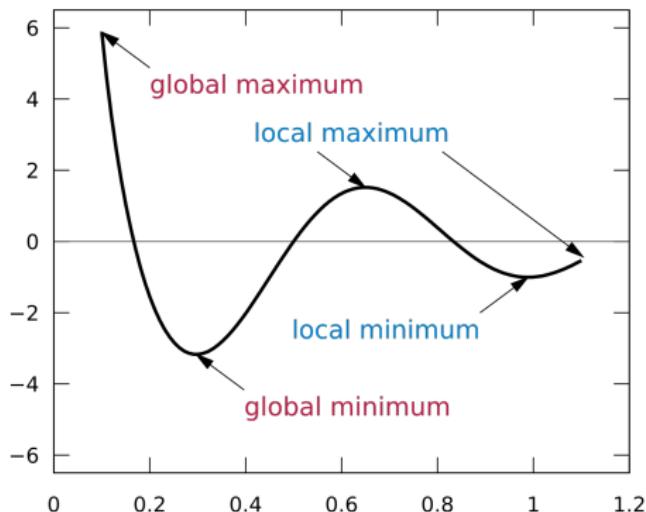
When  $f$  is convex:



$$a_{n+1} = a_n + \eta \nabla f(a_n)$$

# Gradient descent

Multiple minima are commonly present in large dataset

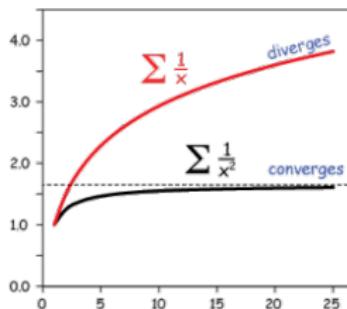


# Learning rate

The learning rate is a tuning parameter in an optimization algorithm that determines the step size at each iteration while moving toward a minimum of a loss function.

This is not that easy... Learning rate can be:

- Constant
- Variable ( a given function)
- Block-set up (sparse data)
- Adaptive (as a function of the loss function)



# Learning rate

The learning rate is a tuning parameter in an optimization algorithm that determines the step size at each iteration while moving toward a minimum of a loss function.

This is not that easy... Learning rate can be:

- Constant
- Variable ( a given function)
- Block-set up (sparse data)
- Adaptive (as a function of the loss function)

