ET-287 – Signal Processing using Neural Networks

3. Feedforward Neural Networks

Professor Sarah Negreiros de Carvalho Leite
Aeronautics Institute of Technology
Electronic Engineering Division
Department of Telecommunications

sarahnc@ita.br, sarah.leite@gp.ita.br
room: 221

Course Syllabus

Unit 1 – Introduction and brief review of linear algebra using Python.

Unit 2 - Introduction to Machine Learning and Neural Networks.

Unit 3 - Feedforward Neural Networks.

a) Multilayer perceptron.

b) Neural networks with radial basis activation function.

c) Extreme learning machines.

d) Convolutional neural networks.

Unit 4 - Recurrent Neural Networks.

Unit 5 – Considerations about Deep Learning.

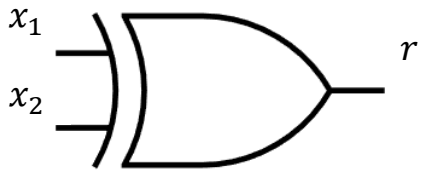
Exercise 1 – Gate XOR

Considering the Gate XOR.

a) Is it a linear problem? Draw a graph.

b) Do a simulation and observe what happens in the solution found by the perceptron. Is it possible to determine the perceptron network parameters capable of solving this problem? Why?

c) Is it possible to separate the 2 classes using the perceptron? Present a solution.



	Inputs			Output	Class
	x_0	x_1	x_2	r	
1	1	0	0	0	A
2	1	0	1	1	B
3	1	1	0	1	B
4	1	1	1	0	A

MultiLayer Perceptron

Multilayer Perceptron

The perceptron is the most rudimentary form of a feedforward neural network used to classify two linearly separable classes.

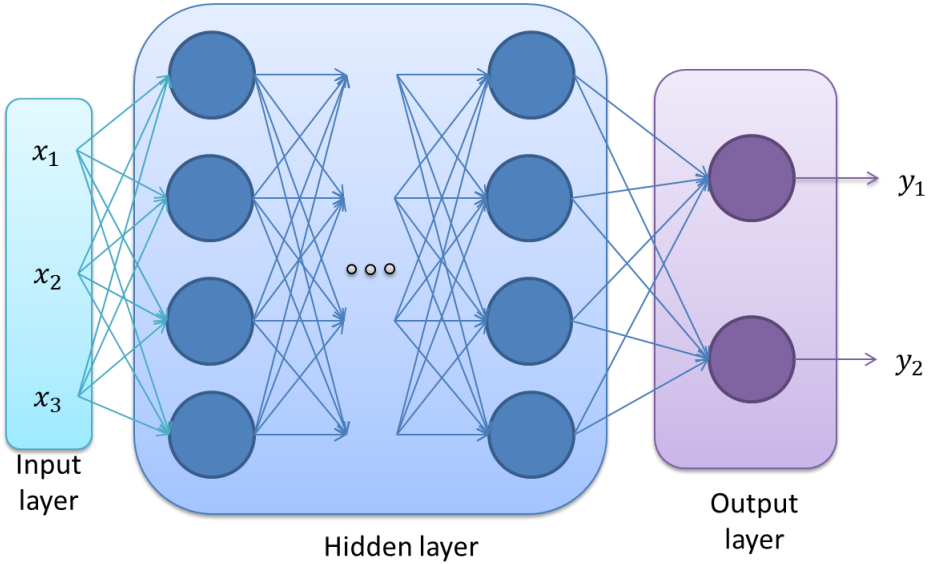
We have just seen that problems with more than two classes (such as the iris flowers example) and with non-linearly separable classes (e.g. XOR gate) cannot be addressed with a perceptron.

The MLP is an immediate extension of the single-layer perceptron network, capable of handling more classes and proposing non-linear separation surfaces for classes, precisely because it has layers of intermediate (hidden) neurons.

Multilayer Perceptron

The MLP network has universal approximation capability, meaning it can find any function necessary to solve the proposed problem. A great number of practical problems can be solved with a single intermediate layer.

A typical architecture of an MLP is:



Input layer: receives information from the database.

Hidden or intermediate layer: responsible for extracting the necessary information from the data.

Output layer: generates the final result of the network.

Multilayer Perceptron

Despite the ability to model complex functions, it needs to be adjusted correctly.

The learning of the MLP occurs through algorithms capable of determining the weights of each neuron, both in the intermediate layers and the output layer, so that all parameters of the network are tuned iteratively.

We will see later on that employing more layers in deep learning structures helps eliminate the need for handling initial data, reducing or even eliminating preprocessing and feature extraction steps. However, this comes with the requirement of a big database.

Database

The quality of the database directly impacts the quality of the solutions that the model can propose.

The **accuracy**, **completeness**, **representativeness**, **consistency** and **relevance** of data are essential for training a reliable and effective neural network. A poor-quality database may result in biased or skewed models that do not generalize well to new, unseen data.

Proper preprocessing steps, including normalization, scaling, handling missing values, filtering, etc., are essential to prepare the data for effective learning.

Database partition

The database that can be divided into two or three subsets:

1. **Training Data** - used to adjust its parameters, such as the number of layers, number of neurons, weights of each neuron, activation functions, optimizer function, etc.

2. **Validation Data** - used to get an idea of the neural network's performance on unknown data and check for overfitting. It also assists in the process of adjusting neural network parameters and evaluating its flexibility. The training goal is to minimize the error or maximize the accuracy with the validation set.

3. **Test Data** - this set is created when the database has sufficient samples. It is used to assess the accuracy of the neural network and estimate the probability of error. The test set provides us a more accurate prediction of the error rate or accuracy that the model should exhibit with unseen data.

Database partition

There is no rule on how to perform the best partitioning of the dataset.

A criterion often adopted is to use:

- 70% - 80% of the samples for training,
- 15% - 25% for validation,
- 5% to 10% for testing.

It is important to maintain equitable representation of the classes or outputs in the partitions.

Database partition

There is no rule on how to perform the best partitioning of the dataset.

A criterion often adopted is to use:

- 70% - 80% of the samples for training,
- 15% - 25% for validation,
- 5% to 10% for testing.

It is important to maintain equitable representation of the classes or outputs in the partitions.

Database partition

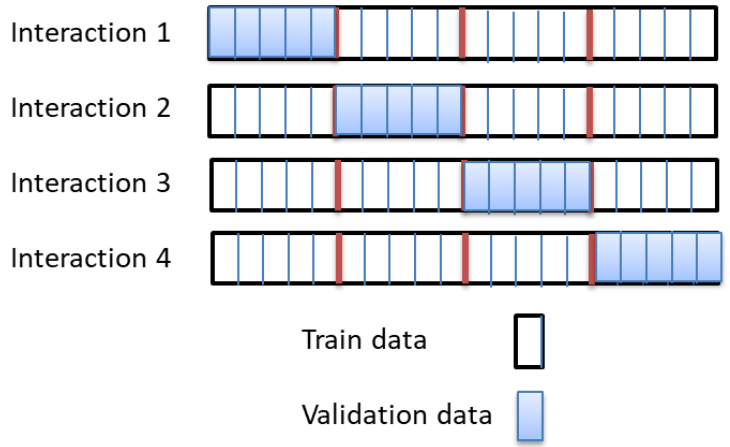
The choice of samples from the database used in each partition set can be **fixed** and **predetermined**. This approach may hinder the network's generalization ability, making it more sensitive to outlier samples in the database.

A more elaborate technique known as **cross-validation** involves dividing the dataset into *k* groups and performing training/validation for each group.
In this method, *k*−1 groups are used to train and to adjust the weights and 1 group is used for validation.
In this approach, all samples participate in both the training and validation stages of the network.

Normally, the neural network's performance is given by the average performance of the *k* obtained neural networks.

Cross-validation

Example of cross-validation with 4 data groups. In each of the 4 iterations, the set of samples participating in training and validation changes.

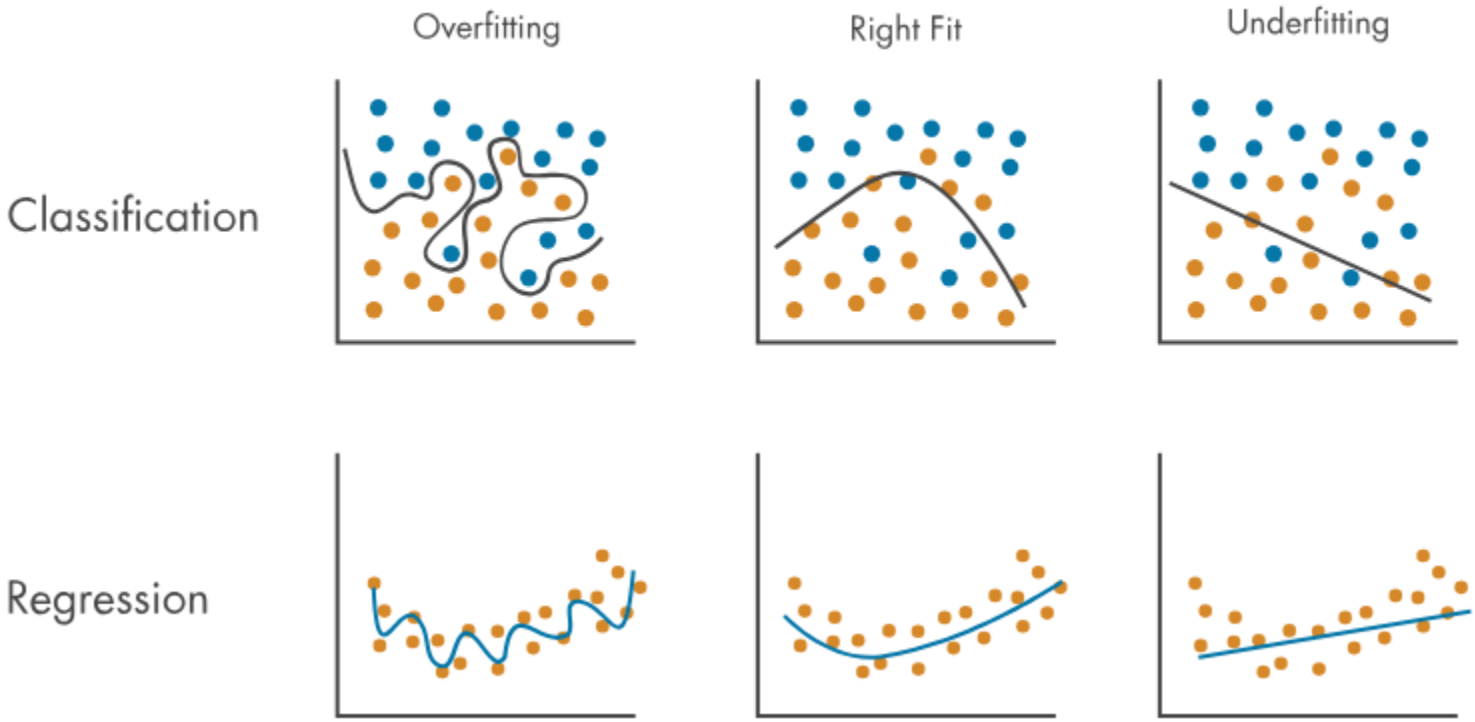


Number of neurons

Selecting the appropriate number of neurons is a critical factor in ensuring the neural network's ability to **generalize** effectively. As the number of neurons in the intermediate layer increases, the model's mathematical **flexibility** and overall **complexity** grow. However, this comes with the risk of **overfitting** the data, resulting in models that struggle to generalize well to new data.

There is no fixed rule for determining the ideal number of neurons, and the adjustment process is empirical. Essentially, the goal is to find a minimum structure that can provide a suitable solution for the given training dataset.

Fit



Source: https://www.mathworks.com/discovery/overfitting.html


QUICO COM OVERFITTING
"AHHH EU SABIA ESSE COM MAÇÃS"

Overfitting

Generally, overfitting occurs when the neural network has too many parameters or does not receive a sufficient dataset.

How to detect:
High accuracy in training and low accuracy in validation and testing.

Possible solutions:
1. Increase the training dataset.
2. Reduce the dimensions (number of layers or neurons) of the neural network.
3. Remove outliers from database.

Synaptic Weight Initialization

A common approach is to initialize the neural network weights with **small and randomly distributed** values (typically around zero).

This initialization promotes the following initial properties of the neural network:

- The mapping tends to have **no predefined bias** and approaches a hyperplane.
- Small values prevent neurons from being in the **saturation** region (in the case of sigmoidal functions), facilitating the weight adjustment process.

Cost Function or Loss Function

The idea of a cost function is related to the expenses incurred in producing a particular product. If we can produce the same product with the same quality and using fewer resources, the cost could be reduced.

The same principle applies to machine learning. A computer may have various methods to perform a task, and one of them will be able to accomplish the task with the desired accuracy in the required time. The method that performs the process the best is the one with the lowest cost.

Cost Function or Loss Function

Humans learn through repetitions ⇔ successes and errors.

The learning algorithm learns through the output of the cost function.

The cost function, or loss function, enables the measurement of the difference between the correct response and the one returned by the algorithm. It serves as a guide for the machine learning process.

Cost Function or Loss Function

Mean Squared Error (MSE): This technique heavily penalizes large errors.

$$C(w) = \frac{1}{N}\sum_{x}^{N} \|y(x) - r\|^2$$

Mean Absolute Error (MAE): Similar to MSE, but it uses the absolute value of errors, making the metric less sensitive to outliers.

$$C(w) = \frac{1}{N}\sum_{x}^{N} |y(x) - r|$$

where N is the number of samples, r are the actual values, and $y(x)$ are the values predicted by the model.

Cost Function or Loss Function

Cross-Entropy is commonly used in classification problems and evaluates the difference between the probability distribution predicted by the model and the true distribution.

The closer to zero, the better.

But, … What is Entropy?

Information

In information theory, the information is inversely proportional to the probability of the occurrence of an event:

$$I \propto \frac{1}{p(x)}$$

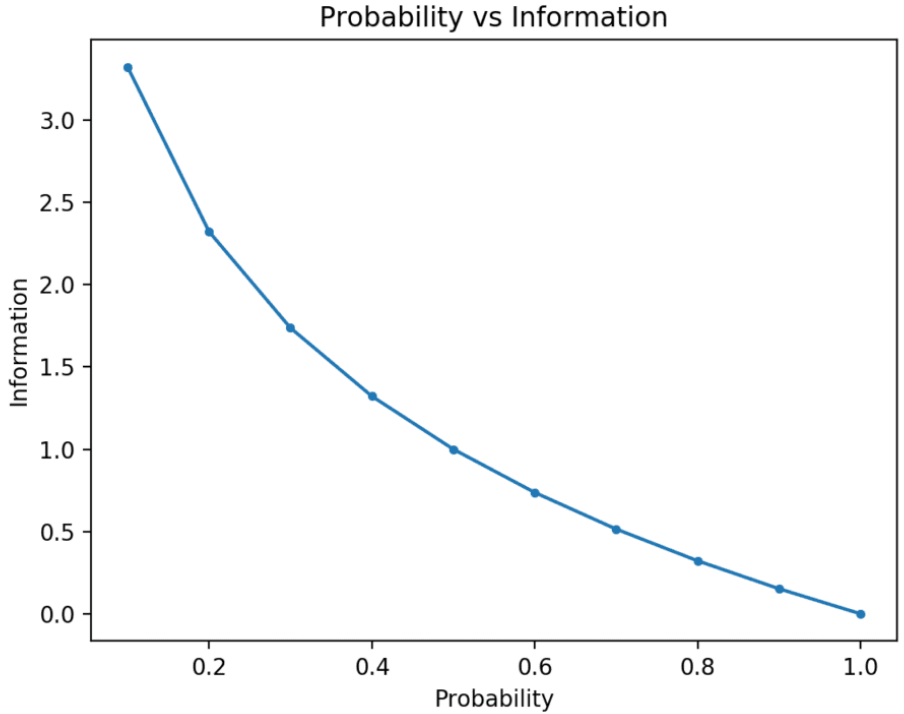
where $p(x)$ is the probability of the event x.

The calculation of information is often denoted as $h(x)$, and in Shannon's Theory, it is convenient to represent information using a base-2 logarithm:

$$h(x) = \log\left(\frac{1}{p(x)}\right) = -\log(p(x))$$

Note that: $0 \leq p(x) \leq 1 \Rightarrow h(x) \geq 0$.

Information



$$h(x) = \log\left(\frac{1}{p(x)}\right) = -\log(p(x))$$

Entropy

Entropy is the number of bits required to transmit a randomly selected event from a probability distribution. A skewed distribution has a low entropy, whereas a distribution where events have equal probability has a larger entropy.

Calculating the information for a random variable is called “information entropy”, “Shannon entropy”, or simply “entropy”.

The entropy can be calculated for a random variable *X,* as follows:

$$H(X) = E[I] = -\sum_{x \in X} p(x)\log(p(x))$$

Example: Entropy

Consider tossing a coin with known probabilities of coming up heads or tails; this can be modelled as a Bernoulli process.

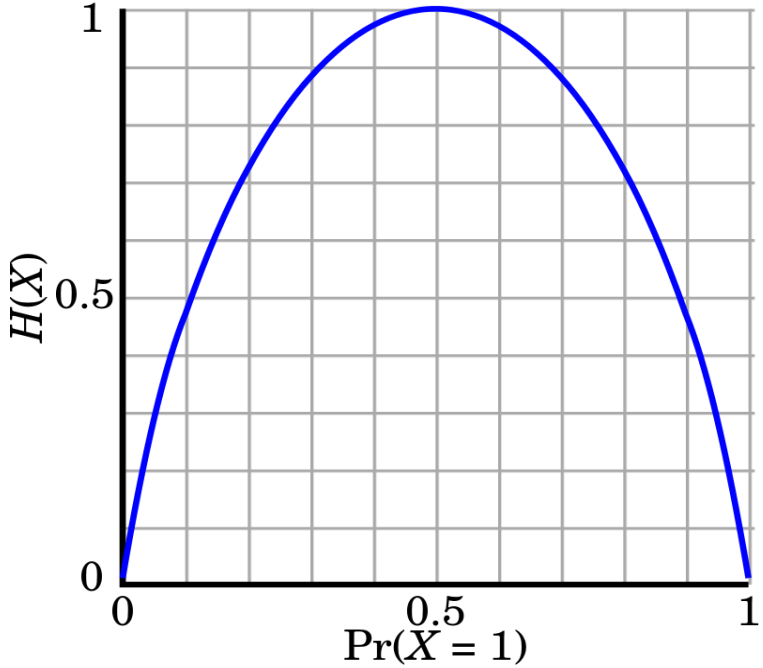
The entropy of the unknown result of the next toss of the coin is maximized if the coin is fair (that is, if heads and tails both have equal probability 1/2).

This is the situation of maximum uncertainty as it is most difficult to predict the outcome of the next toss; the result of each toss of the coin delivers one bit of information.

$$H(X) = -\sum_{x \in X} p(x) \log\big(p(x)\big) = -\sum_{x \in \{1,2\}} \frac{1}{2} \log\left(\frac{1}{2}\right) = -2.\frac{1}{2}(-1) = 1$$

Example: Entropy

Entropy of a coin flip can be graphed as:



$$H(X) = -\sum_{x \in X} p(x) \log\big(p(x)\big)$$

Verify considering some different values for $p(x)$!

Cross-Entropy

Cross-entropy is a measure used in information theory and statistics to quantify the difference between two probability distributions, in our context, the difference between the predicted probability distribution and the true probability distribution .

As a loss function, the cross-entropy measures the performance of a classification model whose output is a probability value between 0 and 1. The cross-entropy loss increases as the predicted probability diverges from the actual label.

Cross-Entropy

There are two main types of cross-entropy loss:

Binary Cross-Entropy: Used for binary classification problems, where there are only two possible classes (0 or 1).

$$C(w) = -\frac{1}{N}\sum_{x=1}^{N}\left[r.log\left(y(x)\right) + (1-r)\ln\left(1-y(x)\right)\right]$$

where N is the number of samples, r is the true label (0 or 1), and $y(x)$ represents the predicted probability of class 1.

Cross-Entropy

Categorical Cross-Entropy: Used for multi-class classification problems.

$$C(w) = -\frac{1}{N}\sum_{x=1}^{N}\sum_{c=1}^{M} r(x,c)\, log\big(y(x,c)\big)$$

where N is the number of samples, M is the number of classes, $r(x,c)$ gives the binary probability (0, 1) of whether class c is the correct classification for the sample x , and $y(x,c)$ represents the predicted probability of sample x belonging to class c.

Note that $r(x,c)$ is often encoded as a one-hot vector, where all elements are 0 except for the index corresponding to the correct class, which is 1.

Cost Function or Loss Function

After detecting an error, the algorithms gradually correct the synaptic weights. Feedback loops allow the neural network to refine its responses and become more accurate.

Throughout the iterations, the level of error obtained by the cost function decreases.

The cost function guides the learning process but requires other algorithms, such as optimization, to make accurate adjustments to the synaptic weights and biases of the neural network

Optimizer

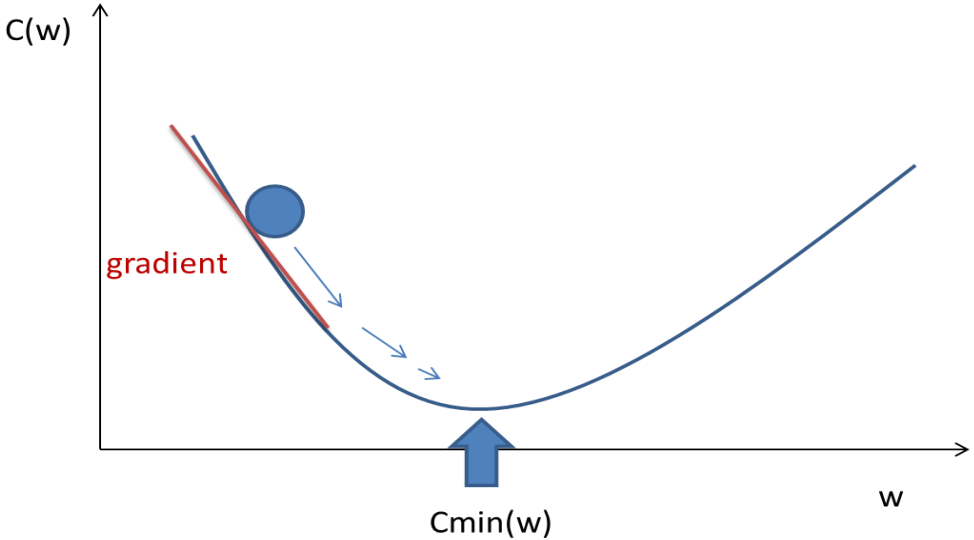
Obviously, the lower the error, the better our model.

The goal of optimization is to find the parameters that minimize our cost function.

The gradient indicates the direction of the steepest ascent of the surface, so simply moving in the opposite direction given by the gradient allows us to find the minimum of the cost function.

Gradient

The gradient is a vector that points in the direction of the steepest ascent of a function. It is calculated by taking the partial derivatives of the function with respect to each of its parameters. The negative gradient points in the direction of the steepest descent, which is the direction toward the minimum of the function.



Gradient

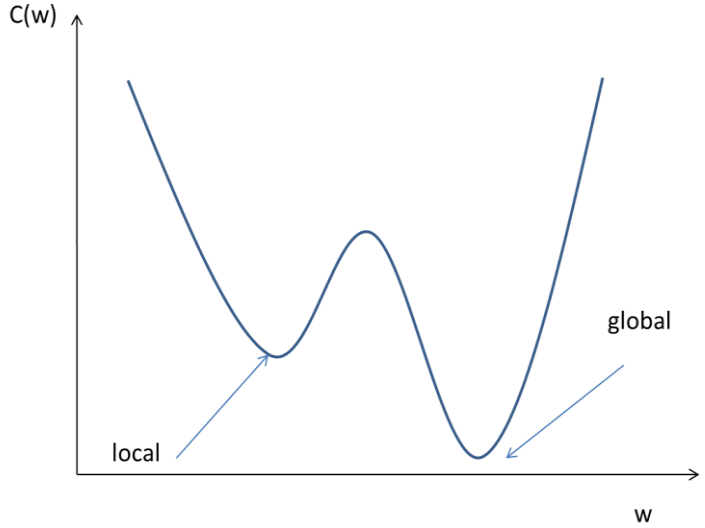
We can make an analogy with a blindfolded person on a mountain trying to find a valley. One possible method would be to feel the slope of the terrain with their feet and take a small step in a downward direction. This process repeats until they reach a flat place, indicating that they have reached a valley.

The size of the step (**learning rate**) is one of the most important **hyperparameters** to decide when training a model, because if its value is too small, progress can be very slow, and if it is too large, we may even worsen the loss.

Gradient

The minimum point can be unique or there can be multiple local minima.

In the latter case, it's crucial to ensure that the optimization algorithm doesn't become trapped in suboptimal local minima.



Gradient descent

Gradient descent is one of the most employed methods in optimization processes in machine learning.

It is an effective technique for reducing the cost function at each iteration. After each step, it checks the errors of the cost function and updates the synaptic weights in search of the minimum cost.

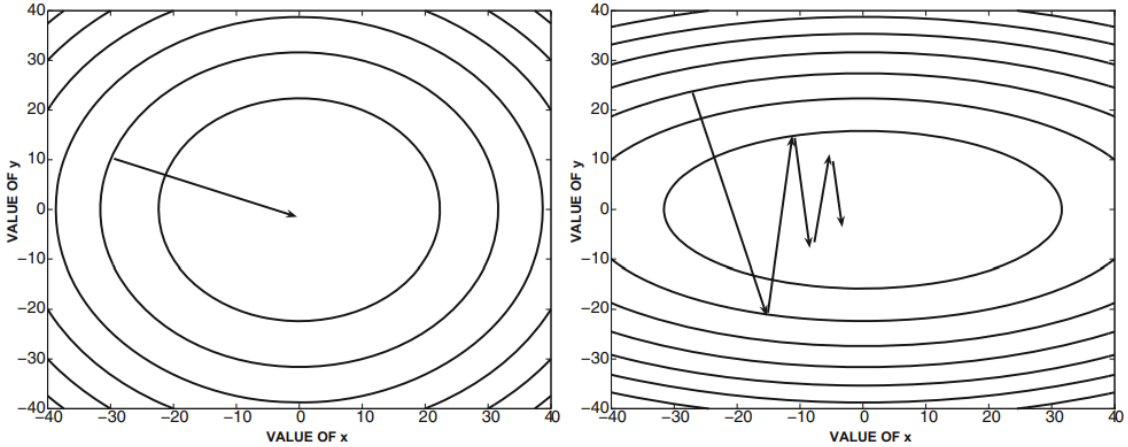
The algorithm considers all samples, and this requires loading all training data into memory.

This can slow down the process when working with very large datasets.

Gradient descent

The steepest-descent direction is the optimal direction only from the perspective of infinitesimal steps. A steepest-descent direction can sometimes become an ascent direction after a small update in parameters. As a result, many course corrections are needed.

The problem of oscillation and zigzagging is quite ubiquitous whenever the steepest-descent direction moves along a direction of high curvature in the loss function.



(a) Loss function is circular bowl $L = x^2 + y^2$

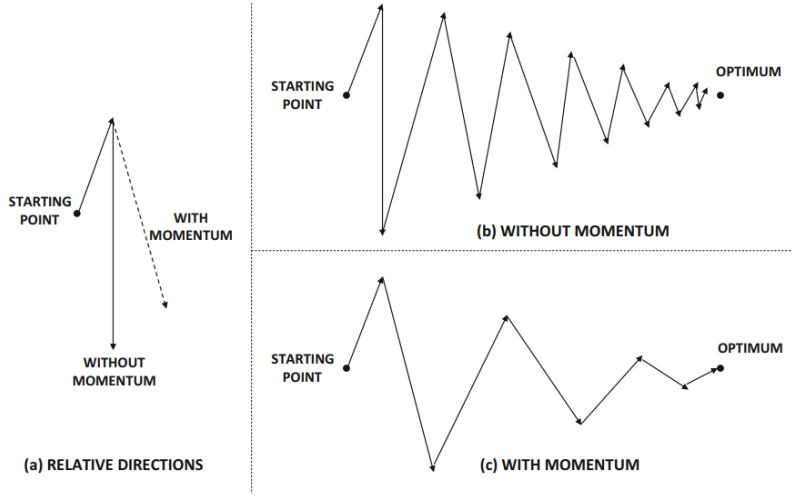
(b) Loss function is elliptical bowl $L = x^2 + 4y^2$

Source: Aggarwal, Charu C. "Neural networks and deep learning", 2018.

Momentum-Based Learning

The zigzagging is a result of highly contradictory steps that cancel out one another and reduce the effective size of the steps in the correct (long-term) direction.

So, it makes more sense to move in an “averaged” direction of the last few steps, so that the zigzagging is smoothed out.



Source: Aggarwal, Charu C. "Neural networks and deep learning”, 2018.

Momentum-Based Learning

The normal updates for gradient descent is gives by:

$$\bar{W} \Leftarrow \bar{W} - \alpha \frac{\partial L}{\partial \bar{W}}$$

While, momentum-based descent:

$$\bar{W} \Leftarrow \bar{W} + \beta \bar{V} - \alpha \frac{\partial L}{\partial \bar{W}}$$

where $\bar{W}$ is the parameter vector, L is the loss function (defined over a mini-batch of instances), α is the learning rate and $\beta \in (0,1)$ is the momentum (smoothing parameter).

Momentum-Based Learning

With momentum-based descent, the learning is accelerated, because one is generally moving in a direction that often points closer to the optimal solution and the useless “sideways” oscillations are muted.

The basic idea is to give greater preference to consistent directions over multiple steps, which have greater importance in the descent.

This allows the use of larger steps in the correct direction without causing overflows or “explosions” in the sideways direction.

Thus, the momentum-based updates can reach the optimal solution in fewer updates.

Source: Aggarwal, Charu C. "Neural networks and deep learning”, 2018.

Parameter-Specific Learning Rates

Another option to get the consistency in the gradient direction can be achieved setting different learning rates for different parameters.

The idea is that parameters with large partial derivatives are often oscillating and zigzagging, whereas parameters with small partial derivatives tend to be more consistent but move in the same direction.

Methods, such as Adam, RMSProp, Adagrad and Adadelta, are variants that explore the same concept of gradient descent.

Source: Aggarwal, Charu C. "Neural networks and deep learning", 2018.

Backpropagation

Our brain learns through the formation and modification of synapses between neurons, which occurs as a result of received stimuli and experience based on trial and error.

The backpropagation algorithm aims to mathematically model this human learning process.

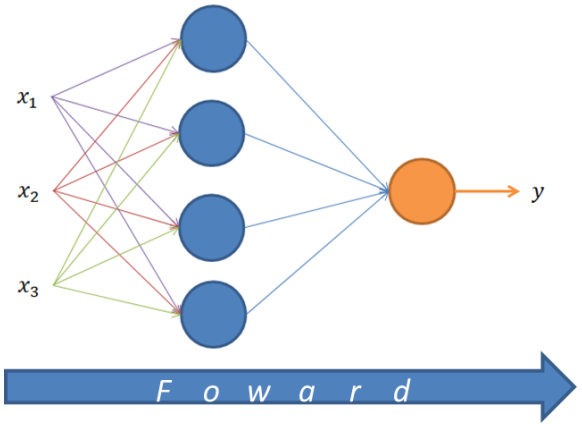
This algorithm consists of iteration in two phases, one forward and the other backward.

Forward phase

1- The data sample is processed by the network, generating an output.

2 - The result found in the neurons of the output layer is compared to the desired result (label). The error is the difference between the values calculated in the output layer and the actual values desired for these neurons. Also, the derivative of the loss function with respect to the output is computed.

3- Thus, it is necessary to backtrack to adjust the weights of the neurons throughout the network in the backward phase. For this, the derivative of the loss function needs to be computed with respect to the weights in all layers in the backwards phase.

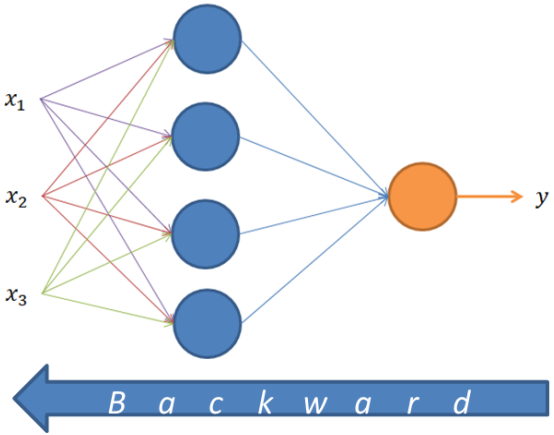


Backward phase

The main goal of the backward phase is to learn the gradient of the loss function with respect to the different weights by using the chain rule of differential calculus.

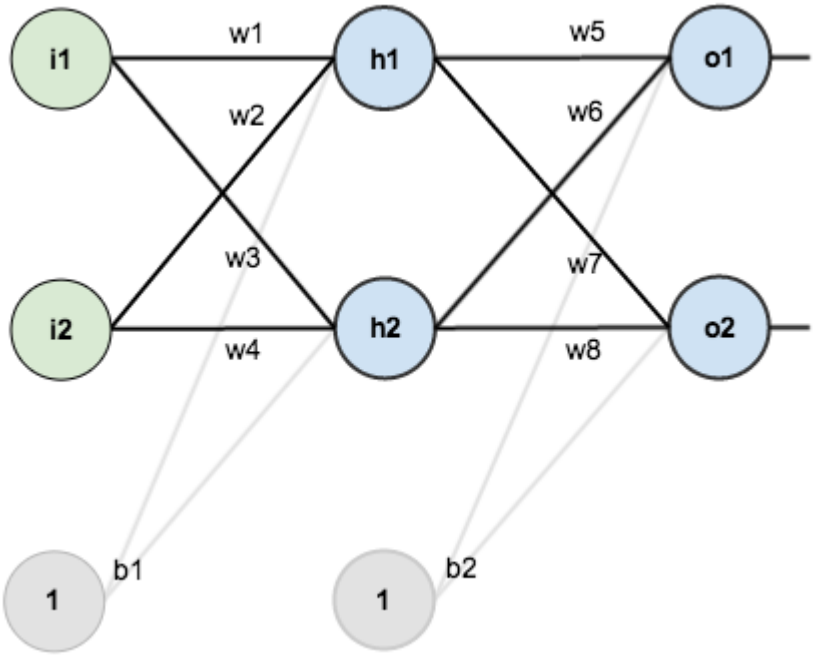
These gradients are used to update the weights. Each weight is updated iteratively, recalculating the gradients in each iteration and considering the learning rate until the network is trained or some stopping criterion occurs.

The algorithm follows this dynamic of updating weights from back to front, starting from the output layer and ending at the input layer, back propagating the error obtained by the network so that the neural network can converge in the training process.



How backpropagation algorithm works?

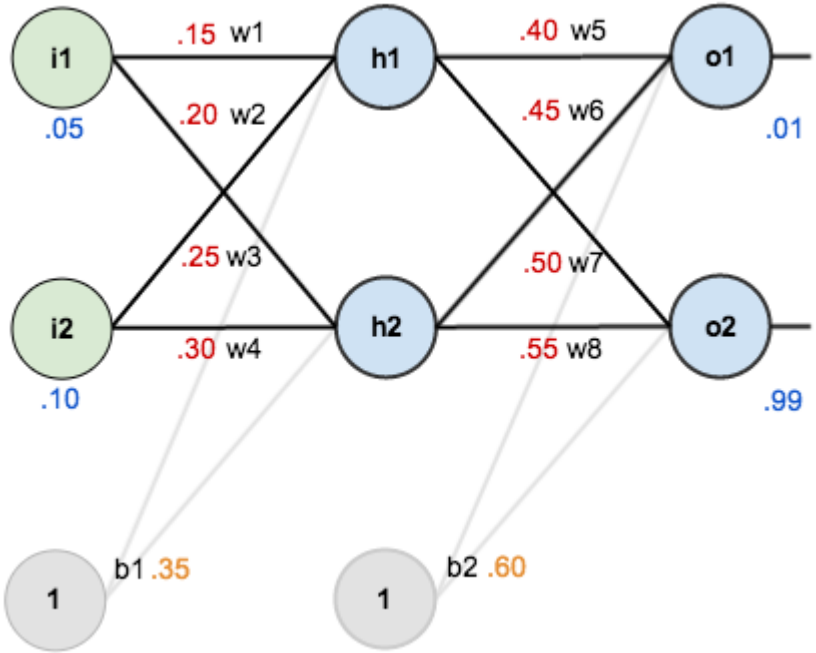
Consider a particular case.



Source: https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

Example: How backpropagation algorithm works?

In order to have some numbers to work with, here are the initial weights, the biases, and training inputs/outputs:



Example: How backpropagation algorithm works?

Forward:

To begin, lets see what the neural network currently predicts given the weights and biases above and inputs of 0.05 and 0.10. To do this we'll feed those inputs forward though the network.

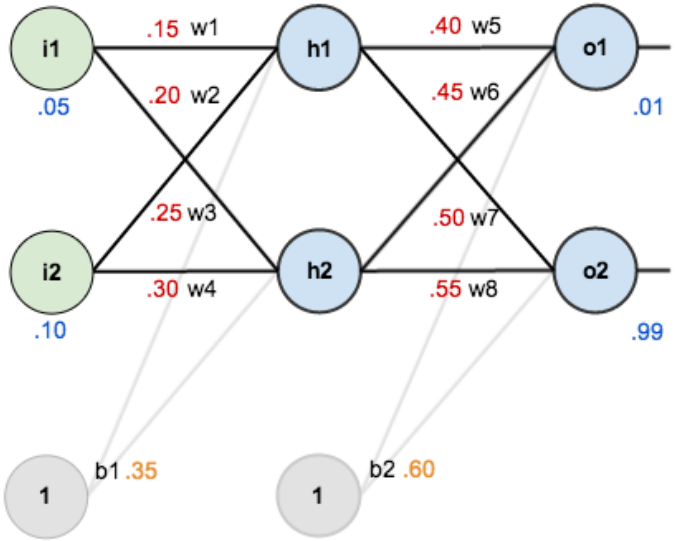
$$h_1 = w_1 \cdot i_1 + w_2 \cdot i_2 + b_1 \cdot 1$$
$$= 0.15 \cdot 0.05 + 0.2 \cdot 0.1 + 0.35 \cdot 1 = 0.3775$$

$$h_2 = w_3 \cdot i_1 + w_4 \cdot i_2 + b_1 \cdot 1$$
$$= 0.05 \cdot 0.25 + 0.10 \cdot 0.30 + 0.35 \cdot 1 = 0.3925$$

Applying the activation function:

$$f(h_1) = \frac{1}{1+e^{-h_1}} = 0.5932$$

$$f(h_2) = \frac{1}{1+e^{-h_2}} = 0.5969$$



Example: How backpropagation algorithm works?

Forward:

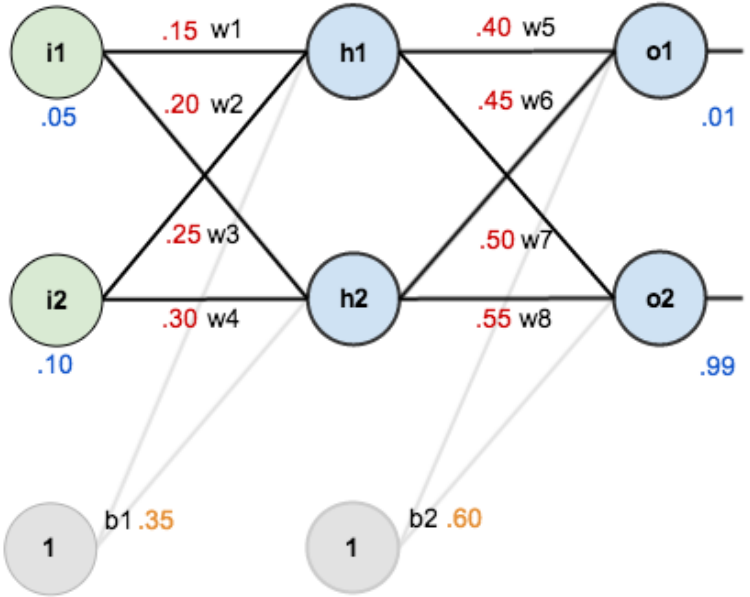
Now, for the output layer neurons, using the output from the hidden layer neurons as inputs.

$o_1 = w_5 \cdot f(h_1) + w_6 \cdot f(h_2) + b_2 \cdot 1$

$= 0.40 \cdot 0.5932 + 0.45 \cdot 0.5969 + 0.60 \cdot 1 = 1.106$

$o_2 = w_7 . f(h_1) + w_8 . f(h_2) + b_2 . 1$

$= 0.50 \cdot 0.5932 + 0.55 \cdot 0.5969 + 0.60 \cdot 1 = 1.225$



Applying the activation function:

$f(o_1) = \frac{1}{1+e^{-o_1}} = 0.7514$

$f(o_2) = \frac{1}{1+e^{-o_2}} = 0.7732$

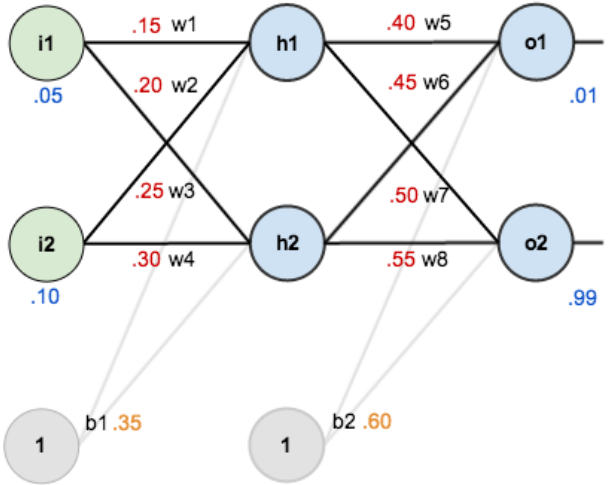
Example: How backpropagation algorithm works?

Forward:

Calculating the Total Error:

$$E = \sum \frac{1}{2}(target - output)^2 = E_{o1} + E_{o2}$$

$$= \frac{1}{2}[(0.01 - 0.7514)^2 + (0.99 - 0.7732)^2] = 0.3049$$



The $\frac{1}{2}$ is included so that exponent is cancelled when we differentiate later on. The result is eventually multiplied by a learning rate anyway so it doesn't matter that we introduce a constant here.

Example: How backpropagation algorithm works?

Backward:

Our goal with backpropagation is to update each of the weights in the network so that they cause the actual output to be closer the target output, thereby minimizing the error for each output neuron and the network as a whole.

We want to know how much a change in w_5 affects the total error, i.e. the gradient with respect w_5: $\frac{\partial E}{\partial w_5}$

By applying the chain rule:

$$\frac{\partial E}{\partial w_5} = \frac{\partial E}{\partial f(o_1)} \cdot \frac{\partial f(o_1)}{\partial o_1} \cdot \frac{\partial o_1}{\partial w_5}$$

Example: How backpropagation algorithm works?

Backward:

First, how much does the total error change with respect to the output?

$$E = \sum \frac{1}{2}(target - output)^2 = E_{o1} + E_{o2}$$

$$= \frac{1}{2}(target_{o1} - f(o_1))^2 + \frac{1}{2}(target_{o2} - f(o_2))^2$$

So,

$$\frac{\partial E}{\partial f(o_1)} = 2 \cdot \frac{1}{2}(target_{o1} - f(o_1))^{2-1} \cdot (-1) + 0$$

$$= -(target_{o1} - f(o_1)) = -(0.01 - 0.7514) = 0.7414$$

Example: How backpropagation algorithm works?

Backward:

Next, how much does the output of $f(o_1)$ change with respect to its total net input?

The partial derivative of the logistic function is:

$$f(o_1) = \frac{1}{1 + e^{-o_1}}$$

$$\frac{\partial f(o_1)}{\partial o_1} = f(o_1) \cdot (1 - f(o_1))$$
$$= 0.7514 \cdot (1 - 0.7514) = 0.1868$$

Example: How backpropagation algorithm works?

Backward:

Finally, how much does the total net input of o_1 change with respect to w_5?

$$o_1 = w_5 \cdot h_1 + w_6 \cdot h_2 + b_2 \cdot 1$$

$$\frac{\partial o_1}{\partial w_5} = h_1 + 0 + 0 = 0.5932$$

Putting it all together:

$$\frac{\partial E}{\partial w_5} = \frac{\partial E}{\partial f(o_1)} \cdot \frac{\partial f(o_1)}{\partial o_1} \cdot \frac{\partial o_1}{\partial w_5}$$
$$= 0.7414 \cdot 0.1868 \cdot 0.5932 = 0.08217$$

Example: How backpropagation algorithm works?

Backward:

To decrease the error, we then subtract this value from the current weight and multiplied by the learning rate (here $\eta = 0.5$):

$$w_5{}^{+} = w_5 + \eta \frac{\partial E}{\partial w_5}$$
$$= 0.4 - 0.5 \cdot 0.08217 = 0.3589$$

We can repeat this process to get the new weights $w_6{}^{+}$, $w_7{}^{+}$ and $w_8{}^{+}$.

$$w_6{}^{+} = 0.408$$
$$w_7{}^{+} = 0.511$$
$$w_8{}^{+} = 0.561$$

Example: How backpropagation algorithm works?

Backward: Next, we'll continue the backwards pass by calculating new values for w_1, w_2, w_3 and w_4.

$$\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial f(h_1)} \cdot \frac{\partial f(h_1)}{\partial h_1} \cdot \frac{\partial h_1}{\partial w_1}$$

We're going to use a similar process, but slightly different to account for the fact that the output of each hidden layer neuron contributes to the output (and therefore error) of multiple output neurons. We know that $f(h_1)$ affects both $f(o_1)$ and $f(o_2)$, therefore the $\frac{\partial E}{\partial f(h_1)}$ needs to take into consideration its effect on the both output neurons:

$$\frac{\partial E}{\partial f(h_1)} = \frac{\partial E_{o1}}{\partial f(h_1)} + \frac{\partial E_{o2}}{\partial f(h_1)}$$

Example: How backpropagation algorithm works?

Starting with:

$$\frac{\partial E_{o1}}{\partial f(h_1)} = \frac{\partial E_{o1}}{\partial o_1} \cdot \frac{\partial o_1}{\partial f(h_1)}$$

Using values we calculated earlier:

$$\frac{\partial E_{o1}}{\partial o_1} = \frac{\partial E_{o1}}{\partial f(o_1)} \cdot \frac{\partial f(o_1)}{\partial o_1}$$
$$= 0.7414 \,.\, 0.1868 = 0.1385$$

and

$$\frac{\partial o_1}{\partial f(h_1)} = w_5 = 0.40$$

So:

$$\frac{\partial E_{o1}}{\partial f(h_1)} = 0.1385 \cdot 0.40 = 0.0554$$

Example: How backpropagation algorithm works?

Following the same process for $\frac{\partial E_{o2}}{\partial f(h_1)}$, we get:

$$\frac{\partial E_{o1}}{\partial f(h_1)} = -0.019$$

Therefore:

$$\frac{\partial E}{\partial f(h_1)} = \frac{\partial E_{o1}}{\partial f(h_1)} + \frac{\partial E_{o2}}{\partial f(h_1)} = 0.0554 + (-0.019) = 0.0363$$

Now, we need to calculate $\frac{\partial f(h_1)}{\partial h_1}$, considering: $f(h_1) = \frac{1}{1+e^{-h_1}}$

$$\frac{\partial f(h_1)}{\partial h_1} = f(h_1) \cdot (1 - f(h_1)) = 0.5932 \cdot (1 - 0.5932) = 0.2413$$

Example: How backpropagation algorithm works?

Now, as: $h_1 = w_1 \cdot i_1 + w_2 \cdot i_2 + b_1 \cdot 1$

$$\frac{\partial h_1}{\partial w_1} = i_1 = 0.05$$

Putting it all together:

$$\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial f(h_1)} \cdot \frac{\partial f(h_1)}{\partial h_1} \cdot \frac{\partial h_1}{\partial w_1}$$
$$= 0.0363 \cdot 0.2413 \cdot 0.05 = 0.000438$$

Example: How backpropagation algorithm works?

Backward:

We can update w_1:

$$w_1{}^{+} = w_1 + \eta \frac{\partial E}{\partial w_1}$$

$$= 0.15 - 0.5 \cdot 0.000438 = 0.1498$$

We can repeat this process to get the new weights $w_2{}^{+}$, $w_3{}^{+}$ and $w_4{}^{+}$.

$$w_2{}^{+} = 0.199$$

$$w_3{}^{+} = 0.249$$

$$w_4{}^{+} = 0.299$$

Example: How backpropagation algorithm works?

Finally, we've updated all of our weights!

When we fed forward the 0.05 and 0.1 inputs originally, the error on the network was 0.3049.

After this first round of backpropagation, the total error is now down to ~0.2910. It might not seem like much, but after repeating this process 10,000 times, for example, the error plummets to $3.5.10^{-5}$.

At this point, when we feed forward 0.05 and 0.1, the two outputs neurons generate 0.015912196 (vs. 0.01 target) and 0.984065734 (vs. 0.99 target).

Exercise

Implement the backpropagation algorithm and perform *N* iterations to check the weight updates and the convergence of the output with the target. Indicate the value of *N* required to achieve an error less than 0.1.

Stop Criteria

The most employed stop criteria for the training stage are:

- Achievement of **performance** considered sufficient for the neural network.
- Maximum number of training **epochs**, where an epoch is defined by the presentation of all samples in the training set to the neural network.

Generally, in conjunction with the above criteria, the following may be considered:

- Magnitude of the **gradient vector** below a certain threshold.
- Training **error** progress below a certain threshold.
- Adjustment of **synaptic weights** below a certain threshold.

The mentioned thresholds should be defined through the performance of some tests considering the database and the application.

Backpropagation

The update of synaptic weights can occur in various ways, considering the training dataset:

- **Online or pattern-by-pattern**: Weight update occurs after each example traverses the network. This mode is perfect when the training set is very large. However, it is sensitive to outliers, so it is necessary to keep the learning rate low. As a consequence, the algorithm is slow to converge to an optimal solution.

- **Batch**: Weight update occurs after processing all samples in the training set. This technique makes optimization fast and less subject to variations.

- **Minibatch or stochastic**: Weight update occurs after the network processes a subset of random samples from the training set. This approach combines the advantages of online mode (low memory usage) with batch mode (fast convergence) by introducing a random element (sub-sampling).

Exercise

Let's become more familiar with neural networks and how their parameters influence problem-solving.

Access the TensorFlow Playground at the link: https://playground.tensorflow.org/ and evaluate how each parameter influences the solution found by the neural network, considering the XOR logic gate problem.

- Number of neurons
- Number of layers
- Activation function
- Size of the training dataset
- Learning rate

Note that there are several configurations that allow solving the problem.
What is the most basic configuration you found that allowed solving the problem?

TensorFlow and Keras

TensorFlow is a machine learning library developed by Google Brain.

And Keras is a library, which runs code on TensorFlow and is user-friendly to implement machine learning algorithms .

Links:
- TensorFlow Keras Documentation: https://www.tensorflow.org/api_docs/python/tf/keras
- Keras Documentation: https://keras.io/

Exercise

Let's design an MLP using the Keras framework to solve the XOR logic gate problem.

Exercise

Approximating the sine function with an MLP.

Exercise: Handwritten digit classification

The MNIST database contains a set of images with handwritten digits. It consists of 60 thousand samples for the training set and 10 thousand samples for the test set.

Exercise: Handwritten digit classification

The MNIST database was extracted from a larger handwritten character database called NIST managed by the National Institute of Standards and Technology.

- All images have been preprocessed and have a fixed size of 28 x 28 pixels.
- The size of the digits has also been normalized in terms of size, and they have been centered in the image.
- The pixels are in grayscale, ranging from 0 to 255.

Design an MLP capable of classifying images by identifying the handwritten digit in them.

Exercise: California Housing Prediction

The dataset comprises information from all block groups in California based on the 1990 Census. A block group is the smallest geographical unit for which the U.S. Census Bureau publishes sample data (a block group typically has a population of 600 to 3,000 people).

The dataset includes 8 variables: median income, housing median age, total rooms, total bedrooms, population, households, latitude and longitude.

The target variable is the median house value for California districts, expressed in hundreds of thousands of dollars ($100,000).

Utilize the provided dataset to train a MLP model to predict the median house value based on the given variables. Adjust the model architecture and parameters as needed to improve predictive accuracy.

Project 3 - Detection of failures in steel plates