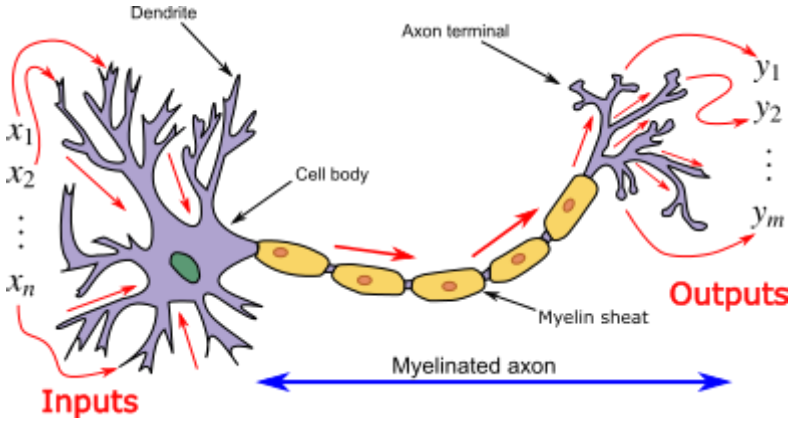
2.1. Biological neurons vs. artificial neurons

2.1.1. Biological neurons

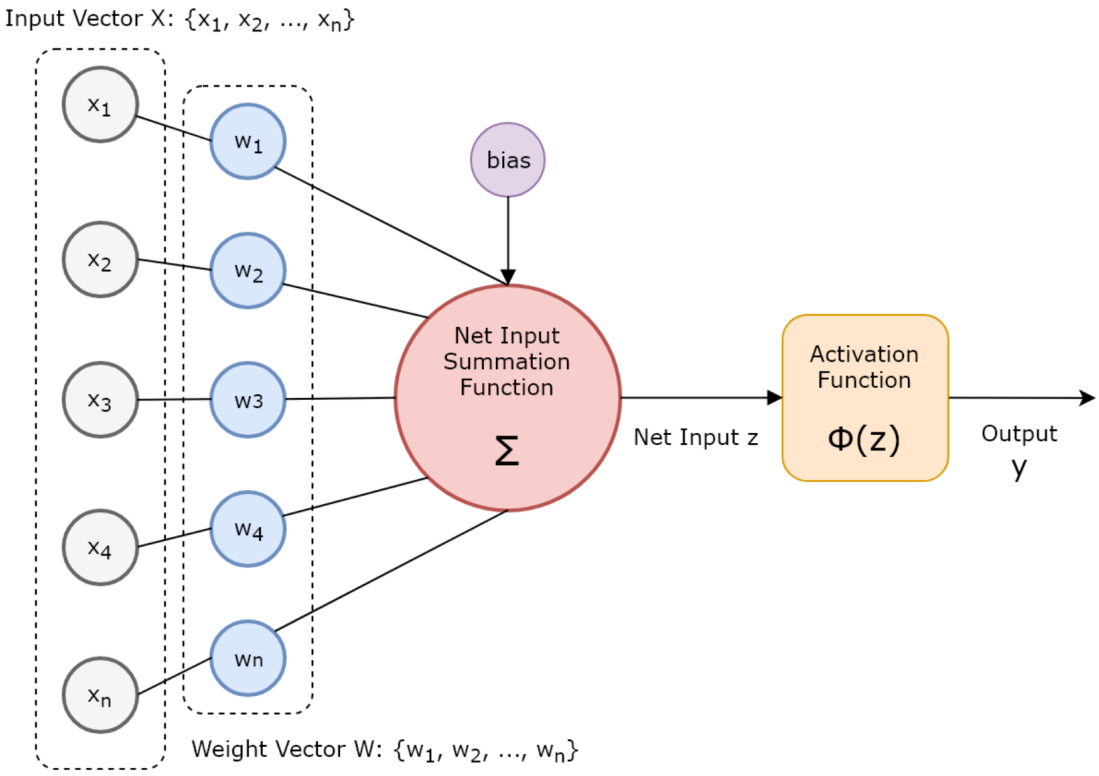
- Our brain is made up of basic cells called **neurons**.
- Biological neurons are composed of the following parts (see image):
  - **Dentrites**: These are the points where information enters the cell.
  - **Cell body**: The place where the cell processes the different inputs to produce an output.
  - **Axon**: The elongated part of the cell that transmits the cell's output signal.
  - **Axon terminals**: The connections that the axon has with other neurons.



- As we can see, the neuron receives various input signals, processes them and transmits them as an output signal to other neurons.
- **Historical note**
  - By meticulously staining and examining thin slices of brain tissue, the Spanish physician Santiago Cajal, was the first to identify neurons
  - The image is a hand-drawn diagram from Cajal's publication (in 1894) showing the growth of a neuron (a–e) and contrasting neurons from frog (A), lizard (B), rat (C), and human (D) samples.

2.1.2. Artificial neurons

- - Based on this scheme, researchers in artificial intelligence decided to create artificial neuron networks trying to reproduce the functioning of the brain in machines.
- An artificial neuron is composed of the following elements:
  - **Input values**. Which are supplied to the neuron.
  - **Weights**. Associated to each input value.
  - **Sum function**. That adds the values of the inputs multiplied by their respective weights (scalar/dot product).
  - **Bias**. Numerical value used to adjust the result of multiplying the inputs by the weights to obtain better predictions.
  - **Activation function**. Function that converts the neuron input into the neuron output.



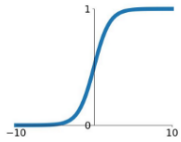
2.1.3. Activation functions

- Activation functions are used to avoid linearities and make the neural network operation more complex.
- Currently the most commonly used is the **rectified linear unit (ReLU)** as it has been proved to better train multi-layered neural networks.
  - It is computationally efficient as it is easy to compute and only a certain number of neurons are activated.
  - Large values of x correspond to large values of ReLU, so the function does not saturate as occurs with other alternatives such as the sigmoid function or the hyperbolic tangent.
  - Therefore ReLU accelerates the convergence of gradient descent towards the global minimum of the loss function (let's see this right now).
- The drawback is the **Dying ReLU problem**:
  - The negative side of the graph makes the gradient value zero.

- Due to this reason, during the backpropagation process, the weights and biases for some neurons are not updated. This can create dead neurons which never get activated. It decreases the model's ability to fit or train from the data properly.
- Some modifications are considered to solve this problem such as the Leaky ReLU or the parametric ReLU.

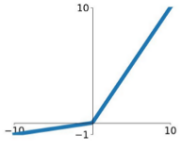
Sigmoid

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



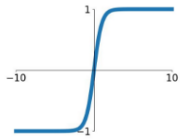
Leaky ReLU

$$\max(0.1x, x)$$



tanh

$$\tanh(x)$$

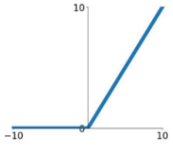


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

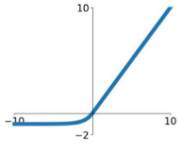
ReLU

$$\max(0, x)$$

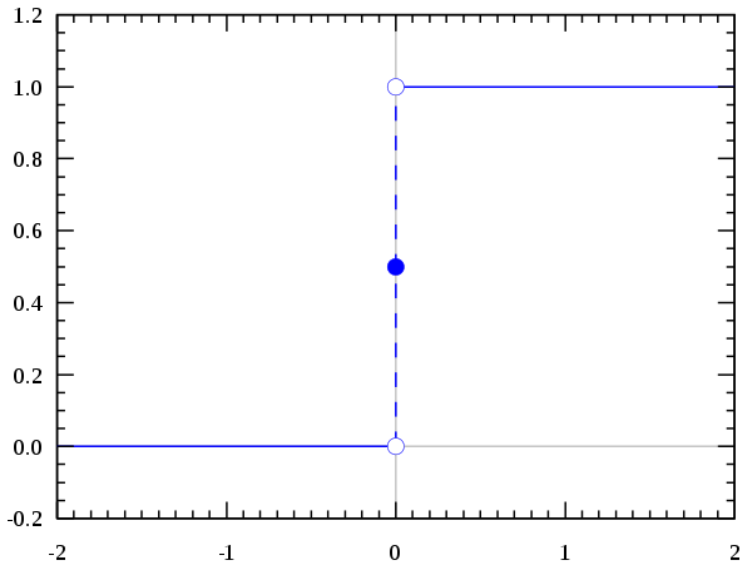


ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

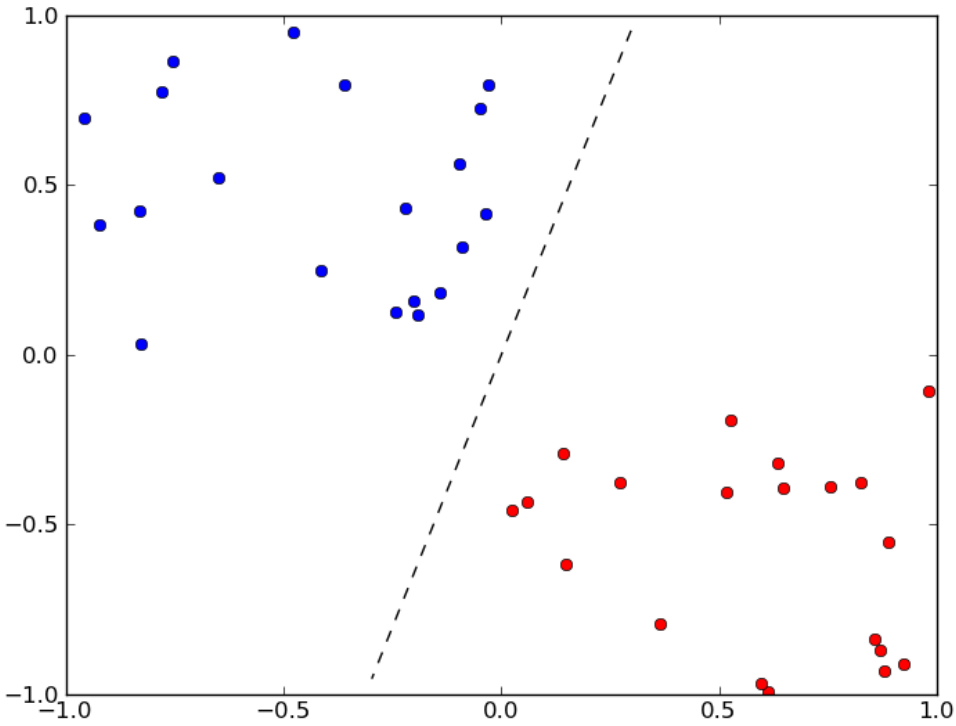


- **Terminology**
  - Historically, a an artificial neuron using the Heaviside step function (see image) as the activation function is called a "**Perceptron**".
  - An architecture that has several neurons is called a "**single-layer perceptron**" and is the simplest feedforward neural network.

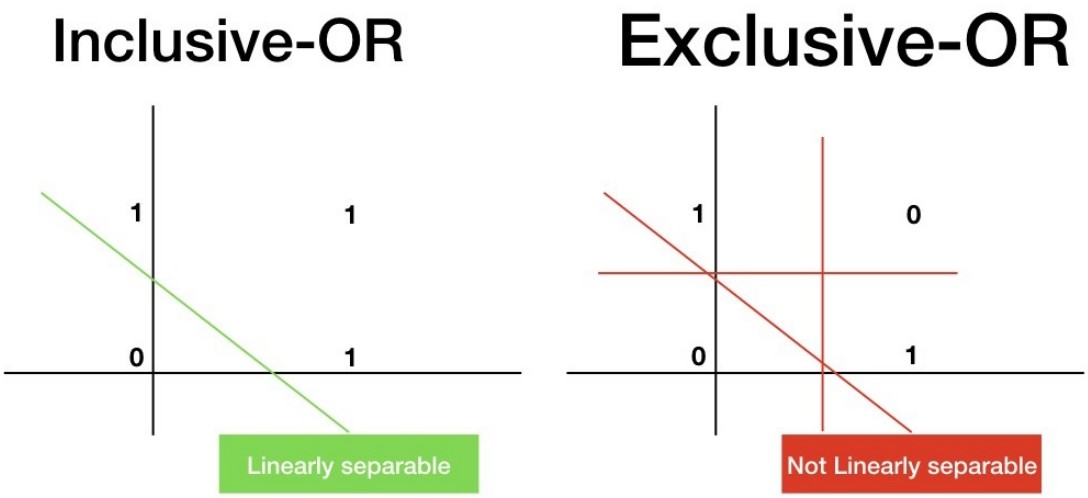


2.1.4. Problems with single layer neural networks

- Single layer perceptrons/neural networks act as a linear discriminator, making a decision about membership of one group or another based on the value of a linear combination of the features.



- **XOR problem**
  - Multilayer neural networks could not solve problems that were not linearly separable, and the simplest of these is the exclusive *OR* or *XOR* problem.
  - The *XOR* function accepts two binary inputs and the result is 1 if either of them is 1 but not both at the same time.
  - If we represent this function on a 2D graph we see that we cannot separate the two kinds of output with a single line, so they are not linearly separable.



a	b	c
0	0	0
0	1	1
1	0	1
1	1	1

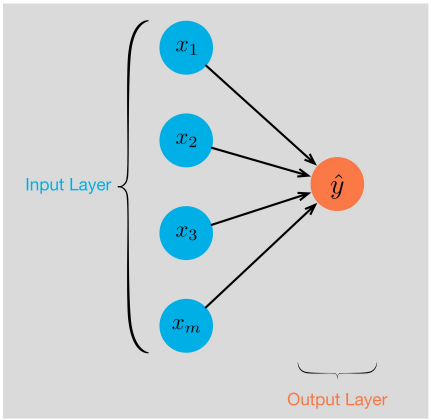
a	b	c
0	0	0
0	1	1
1	0	1
1	1	0

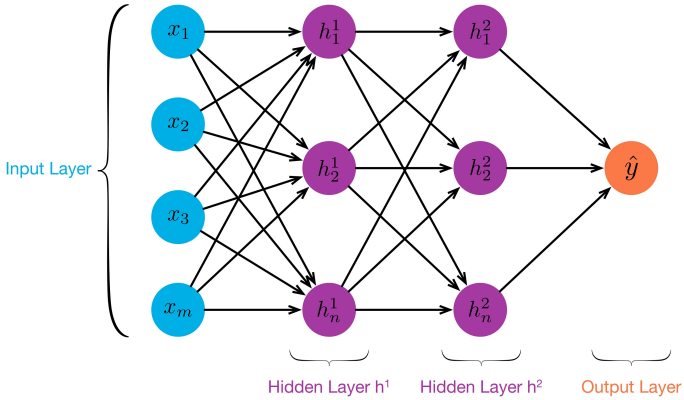
2.2. Multilayer neural networks

- In 1986, Geoffrey Hinton, David Rumelhart, and Ronald Williams published a paper "*Learning representations by back-propagating errors*", which introduced hidden layers and backpropagation.

2.2.1. Hidden layers

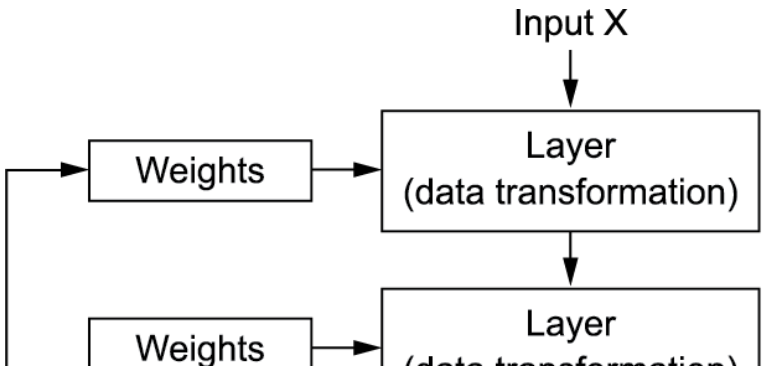
- Hidden Layers are are neuron nodes stacked in between inputs and outputs, allowing neural networks to learn not linearly separable problems (such as XOR logic).
- In the image we can see the difference between a single layer neural network and a multi layer neural network with two hidden layers.
- Information flows forward (*forward pass*) from the input layers to the output layers.





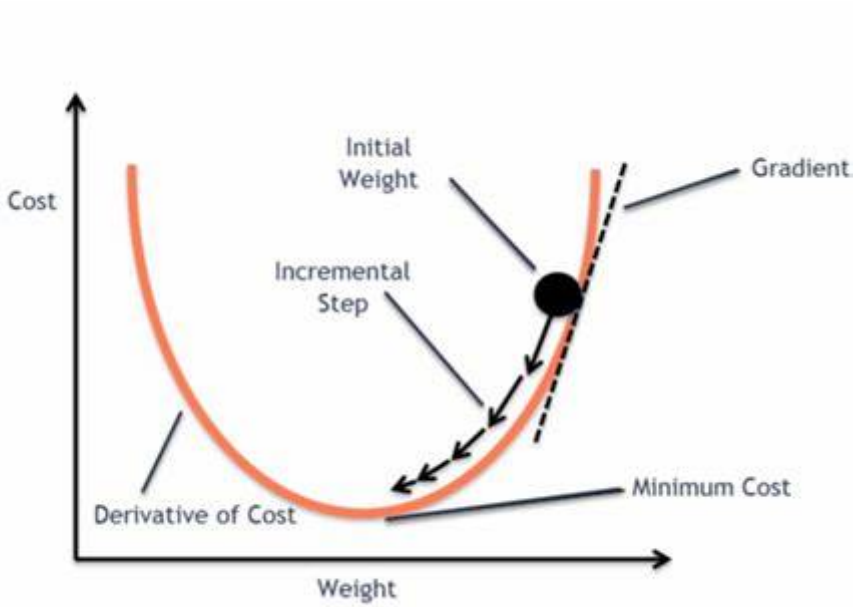
2.2.2. Backpropagation

- Backpropagation, is a procedure to repeatedly adjust the weights so as to minimize the difference between actual output and desired output.
- In the image we can see that the model is composed of layers (with their repective weights) that are chained together.
- The layers map the input data to predictions (*forward pass*).
- The loss function then compares these predictions to the targets, producing a loss value: a measure of how well the model's predictions match what was expected.
- The optimizer uses this loss value to update the model's weights. This is the *backwards pass*, or the backpropagation of the error.



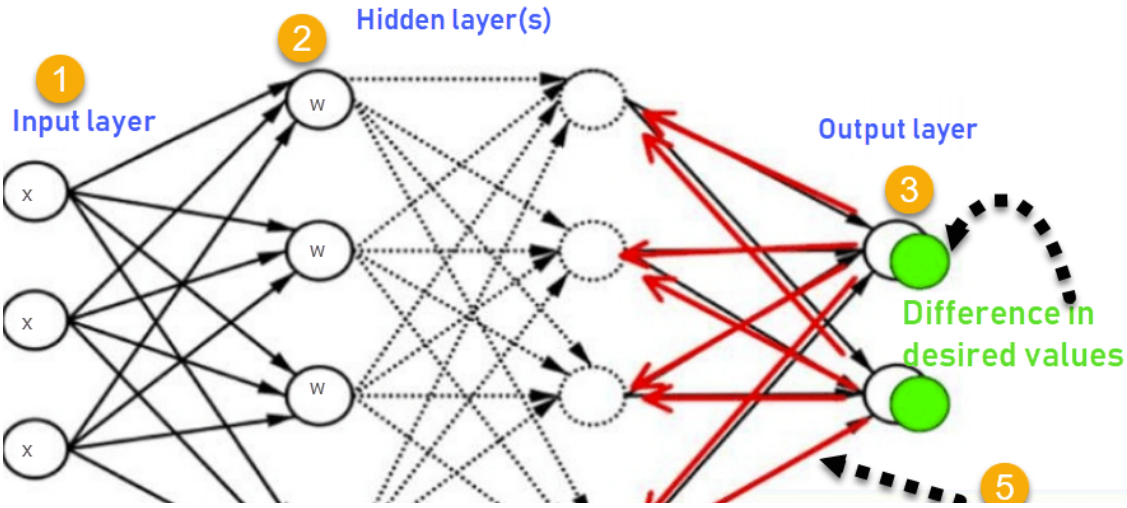
Gradient descent

- Backpropagation is based on gradient descent, an iterative optimization algorithm for finding a local minimum of a differentiable function.
- The idea is to take repeated steps in the opposite direction of the gradient of the function at the current point, because this is the direction of steepest descent.
- The gradient is a multivariable generalization of the derivative and, like the derivative, represents the slope of the tangent line to the graph of a function (see image).



Backwards propagation of the error

- We need to be able to efficiently adjust parameters through multiple layers of artificial neurons.
- Backpropagation courses through a neural network in the opposite direction of forward propagation.
- Using backpropagation, we move layer-by-layer backwards through the network, starting at the cost in the output layer, and we find the gradients of every single parameter.
- A given parameter's gradient can then be used to adjust the parameter up or down whichever of the two directions is associated with a reduction in cost.
- In a nutshell, backpropagation uses cost to calculate the relative contribution by every single parameter to the total cost, and then it updates each parameter accordingly.



2.2.3. Hyperparameters

Parameters of a neural network

- The network parameters are the weights of the network (including here the *bias*)
- These parameters are automatically estimated or learned from the training set.
- Once the network is trained the parameters are used to convert the network inputs into the network predictions.

Hyperparameters of a neural network

- Hyperparameters are other parameters that exist in the network but their estimation is not the objective of the learning process.
- There are two types of hyperparameters: structural hyperparameters and learning hyperparameters.

Structural (model) hyperparameters

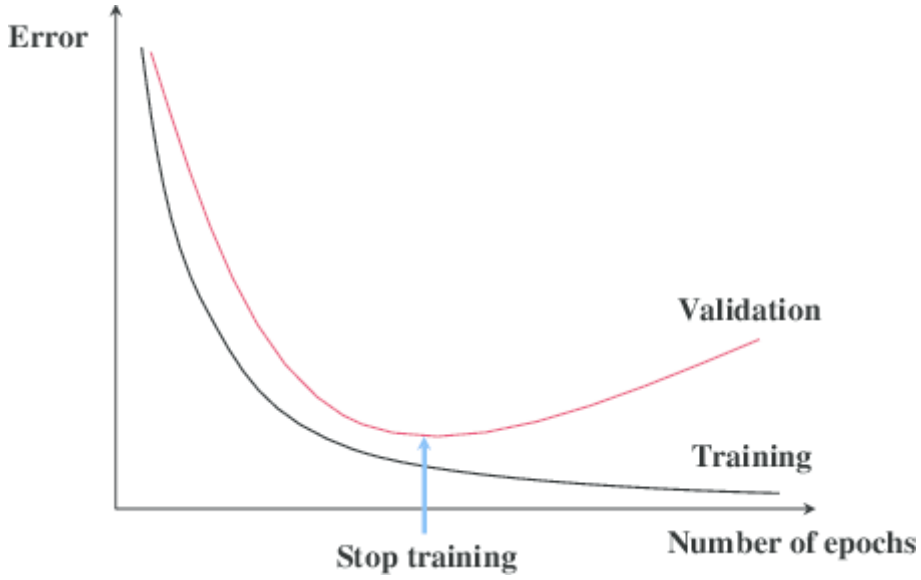
- Structural hyperparameters define the structure and topology of the neural network.
- These include: number of layers, number of neurons per layer, activation functions, initialization of weights, etc.
- With the structural parameters there is a **compromise between the capacity of a network and its generalization ability**
  - The capacity of the network is determined by the number of trainable parameters of the network (number of layers, number of neurons in each layer).
  - A network with a small capacity (few layers and/or few neurons) will have limited resources to learn the patterns present in the data, and its results will underfit.
  - A network with a large capacity (many layers and/or many neurons) will have a large capacity to learn the patterns present in the data, but also to overfit the training data and lose generalization capabilities.
  - So there is a trade-off, we have to have a network with enough capacity to learn the patterns in the data, but not with too much capacity so that it overfits the training data.

Learning (algorithm) hyperparameters

- Learning hyperparameters are the parameters that control the learning algorithm and affect the speed and quality of the learning process.
- Among these we can mention: epochs, batch size, learning rate, etc.

Epochs

- An iteration (*epoch*) represents one pass of all the training data through the neural network.
- Normally when training a model we make multiple iterations or epochs.
- To determine the number of *epochs* to use the main metric is the **validation error**.
- The idea is to continue training as long as the validation error decreases. If the value remains constant the network is not gaining generalization capability, if the error increases the network is overfitting.



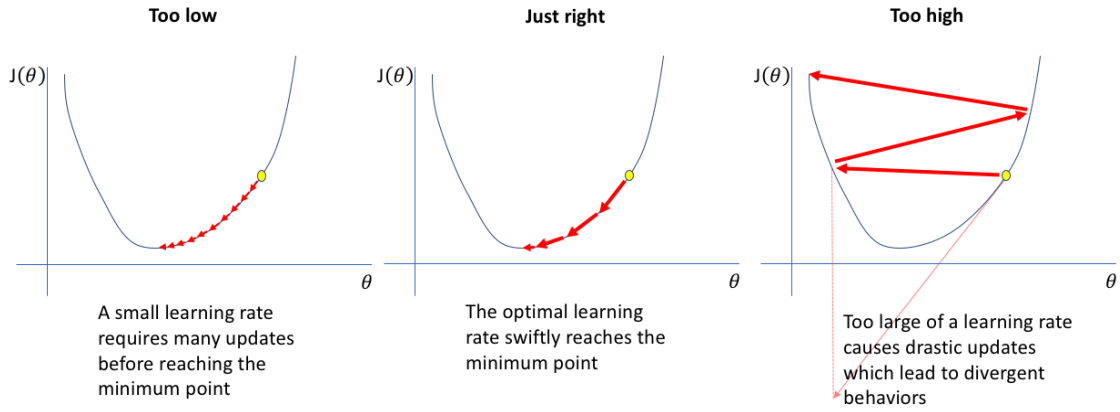
Batch size

- The batch size is a hyperparameter that determines the number of examples to be processed before updating the internal parameters of the model.
- We can classify our gradient descent algorithm as follows according to the size of the *batch*:
  - **Batch Gradient Descent**. The size of the batch is equal to the size of the training set. It is the traditional gradient descent algorithm.
  - **Stochastic Gradient Descent**: The size of the batch is one.
  - **Mini-Batch Gradient Descent**: The batch size is greater than one but smaller than the training set.
- The last case is the most common case, where batch sizes that are powers of two (32, 64, 128, 256, 512, etc.) are usually used.
- The size of the batch influences the following:
  - *Large sizes*:
    - The calculation is computationally more efficient since the model is updated fewer times.
    - Although more efficient, it also requires more memory to store the results.
    - Fewer updates imply a slower learning speed.
    - The slower update frequency results in a more stable and accurate error gradient.
  - *Small sizes*

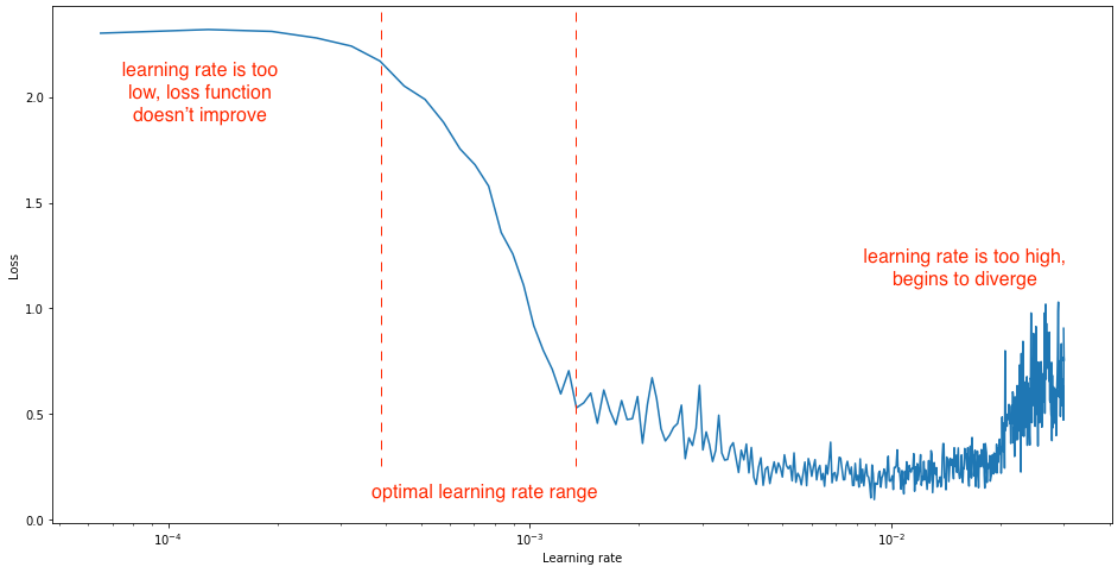
- The computation is computationally more inefficient since the model is updated more times.
- Memory requirements are lower.
- More updates imply faster learning speed.
- Higher update frequency may incorporate noise in the learning process and cause the model error to vary and may complicate reaching a minimum.

Learning rate

- The learning rate is a value that allows us to determine the amount of change we are going to make in the weights in the error backpropagation step.
- Small learning rates cause learning to be slow.
- Fast learning rates may cause divergent behavior preventing the model from advancing to the minimum of error.



- Choosing the right learning rate is complicated as the way to do it is typically by trial and error.
- Typical rate values start at `0.1` and go down to `0.000001` with `0.001` being a usual value.



2.2.4. Hyperparameters tunning

- **Hyperparameter tuning** consists of finding the exact point of complexity in the model such that there is a balance between underfitting and overfitting.

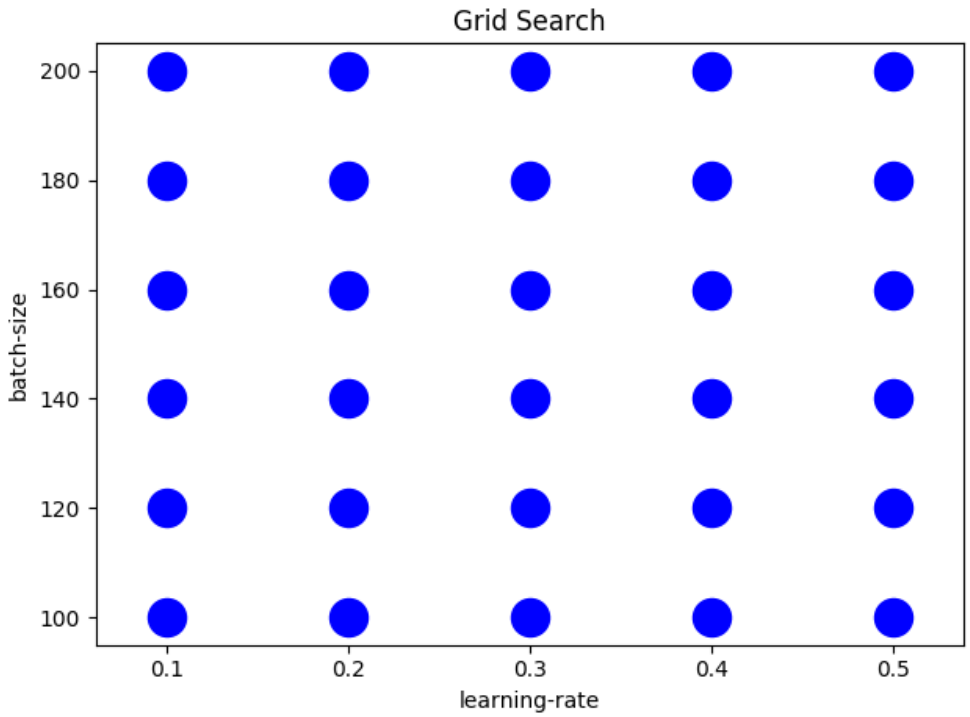
- When building a deep learning model, you have to make many seemingly arbitrary decisions: number of layers, number of neurons in each layer, learning rate, batch size, optimizer, etc.
- In practice, experienced machine learning engineers and researchers build intuition over time as to what works and what doesn't when it comes to these choices—they develop hyperparameter-tuning skills. But even if you have very good intuition, your initial decisions will be almost always suboptimal.
- We need an automatic and systematic way to explore the space of possible decisions and choose the best optimized hyperparameters for our problem. That is hyperparameter tunning.

Methods for hyperparameter tunning

- Hyperparameter optimization is like building a model over our model.
  - The input is a a tuple of hyperparameters.
  - The output is the associated loss with these hyperparameters
  - Our objetive is to find a tuple of hyperparameters that yields an optimal model which minimizes a predefined loss function on given test data.
- The problem is that the hyperparameter space is typically made up of discrete decisions and thus isn't continuous or differentiable.
  - Hence, you typically can't do gradient descent in hyperparameter space.
  - Instead, you must rely on **gradient-free optimization techniques**, which naturally are far less efficient than gradient descent.
- Many different techniques are possible:

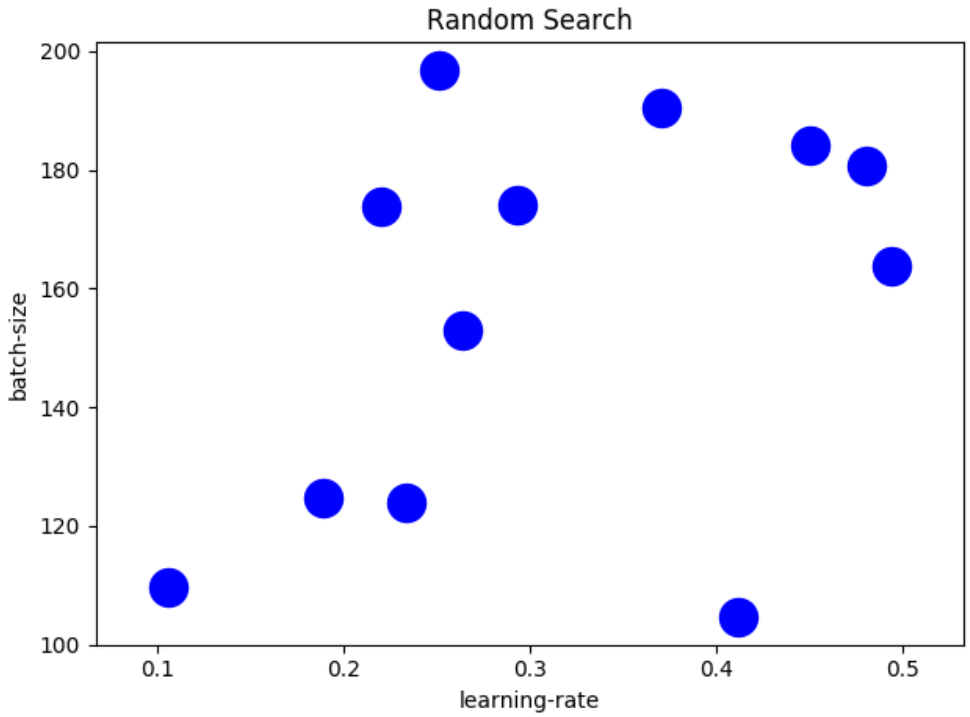
Grid search

- The traditional way of performing hyperparameter optimization has been grid search.
- It is simply an exhaustive searching through a manually specified subset of the hyperparameter space of a learning algorithm.
- For example, in the image we want to establish the best combination of batch size and learning rate, so we search between all the possible values of batch size (from 100 to 200) and learning rate (from 0.1 to 0.5)



Random search

- Random Search replaces the exhaustive enumeration of all combinations by selecting them randomly.
- It is less expensive computationally and can outperform Grid search, especially when only a small number of hyperparameters affects the final performance of the machine learning algorithm.



Bayesian optimization

- Bayesian optimization is a global optimization method for noisy black-box functions.
- Applied to hyperparameter optimization, Bayesian optimization builds a probabilistic model of the function mapping from hyperparameter values to the objective evaluated on a validation set.
- By iteratively evaluating a promising hyperparameter configuration based on the current model, and then updating it, Bayesian optimization aims to gather observations revealing as much information as possible about this function and, in particular, the location of the optimum.
- It tries to balance exploration (hyperparameters for which the outcome is most uncertain) and exploitation (hyperparameters expected close to the optimum).
- In practice, Bayesian optimization has been shown to obtain better results in fewer evaluations compared to grid search and random search, due to the ability to reason about the quality of experiments before they are run.

Evolutionary optimization

- Evolutionary optimization is a methodology for the global optimization of noisy black-box functions.
- In hyperparameter optimization, evolutionary optimization uses evolutionary algorithms to search the space of hyperparameters for a given algorithm.
- Evolutionary hyperparameter optimization follows a process inspired by the biological concept of evolution