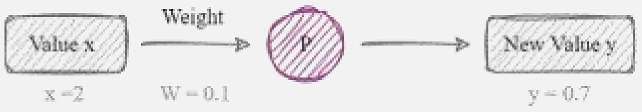
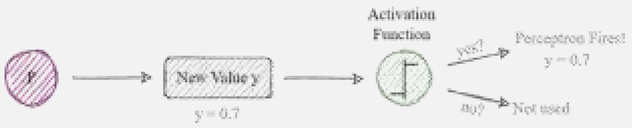
Artificial Neural Network (ANN)

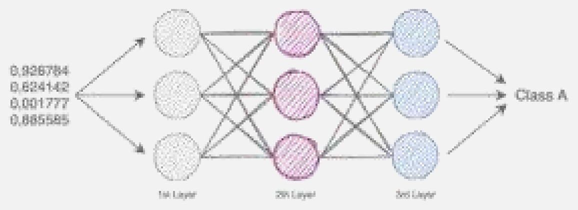
The architecture of the human brain inspires ANN approaches. It attempts to imitate the functioning of neurons using unitary blocks called "perceptron". The perceptron is a mathematical function. It takes numerical data and multiplies it by weight coefficients to produce a new value.



This value is then passed to a so-called activation function which determines whether the perceptron should be activated. If the perceptron is activated, the new value can be used as input to another perceptron.

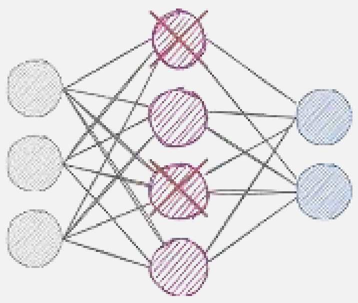


The connection of the perceptron to each other forms a network of perceptrons organized in layers, called an artificial neural network, as shown below.



Organization of perceptrons into layers.

For the artificial neural network to be usable, the weight coefficients of each perceptron must be determined. They are determined by a learning phase whose objective is to adapt the weight coefficients of each perceptron so that the predicted results correspond to an expected result that we provide (E.g., the output should be 1, which corresponds the class A). The provided "training" data used in this learning phase may lead to a configuration of weight coefficients too specific. In such a case, the ANN gives excellent results on the training dataset but significantly worse results on other datasets. This is called overfitting. A so-called "dropout" layer is used to avoid overfitting, which consists of randomly deactivating perceptron according to a given probability.



To improve learning, we use an optimizer, an algorithm to find the value of weight coefficients that minimize errors when mapping inputs and outputs. We must also determine a cost function to calculate the difference between the predicted and expected values. Finally, we define the evaluation metric (such as accuracy) used during training. To set up an artificial neural network, there are thus, five main steps:

1. Preparing the data to make it usable (e.g., by transforming the pixel values into the interval [0;1]).
2. Defining the different perceptron layers
3. Defining the type of activation function for each perceptron layer.
4. Defining dropout layers to avoid overfitting.
5. Setting up-up the learning phase by defining the optimizer, the cost function, and the evaluation metrics.

Easy enough, hun? 😆

If I had to list the main advantages of the ANN, I would say that:

- ✓ It can solve complex non-linear problems.
- ✓ It is highly scalable.
- ✓ It produces the same accuracy rate with small and large data sizes and has a high parallel processing capability.

- ✓ It produces fast predictions after training.
- ✓ It matches any numerical tabular data input to a numerical tabular data output (mainly through the activation function).

Advantages unquestionably come with limits which are that:

- ✓ It only works with tabular data. This means that we must transform the input data if we are working with 2D or 3D data, which leads to many parameters being trained (equivalent to the number of pixels for an image or the number of points for a point cloud).
- ✓ It does not use spatial features (such as pixel layout) or sequential information (what happens before or after).
- ✓ Its design is empirical; the network's structure cannot be determined other than by the experiment, trial, and error.
- ✓ This design is even more difficult because the ANN produces results with a so-called "tunnel effect". In other words, it does not explicitly produce a reason for the result. Therefore, it is also difficult to trust the trained network.
- ✓ Finally, in some cases, the computed gradient is infinitesimally slight, which prevents the weights from changing value and thus forces the training to stagnate. This problem is known as the "vanishing gradient problem".

What a massive load of new words in this section! You can come back anytime to it when needed.

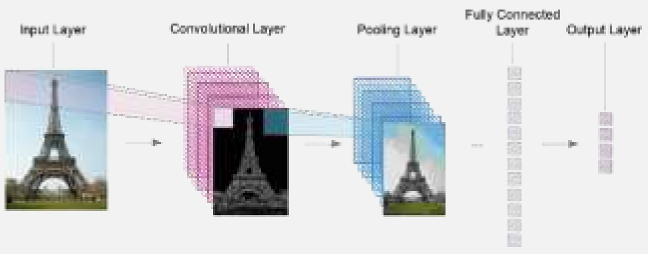
🤿 Dive-in: To hands-on fix the jargon you just went through (strength and honor to you 😊), you can through the ANN python project to recognize numbers and come back thereafter in beast mode.

Convolutional Neural Network (CNN)

Convolutional neural networks (CNN) are based on convolutional filters commonly used for stylish filters in classical image-processing solutions such as Photoshop or Gimp. Therefore, they are particularly suitable for image processing tasks 👍. Simply put, CNNs are composed of:

- The input layer, which contains the image(s) and is defined according to the shape of the image(s) to be processed.
- Convolutional layers, which extract features.
- Pooling layers, which simplify the results of the previous layers.
- Fully connected layers (also called "Dense"), which are used to adapt output.
- The output layer, which contains the result as the label corresponding to the input data.

Layers use activation functions such as rectified linear units (ReLu, which replaces all negative values with 0) to correct previous results. The main advantage of ReLu is the non-saturation of its gradient, which considerably speeds up the convergence of the gradient descent. Essentially, that means that it allows for reducing the risk of vanishing gradient. The following figure illustrates the interweaving of these layers.



CNN architecture