Data Analysis and Machine Learning: Elements of machine learning

Morten Hjorth-Jensen[1,2]

[1]Department of Physics, University of Oslo
[2]Department of Physics and Astronomy and National Superconducting Cyclotron Laboratory, Michigan State University

Nov 2, 2017

What is Machine Learning?

Machine learning is the science of giving computers the ability to learn without being explicitly programmed. The idea is that there exist generic algorithms which can be used to find patterns in a broad class of data sets without having to write code specifically for each problem. The algorithm will build its own logic based on the data.

Machine learning is a subfield of computer science, and is closely related to computational statistics. It evolved from the study of pattern recognition in artificial intelligence (AI) research, and has made contributions to AI tasks like computer vision, natural language processing and speech recognition. It has also, especially in later years, found applications in a wide variety of other areas, including bioinformatics, economy, physics, finance and marketing.

Types of Machine Learning

The approaches to machine learning are many, but are often split into two main categories. In *supervised learning* we know the answer to a problem, and let the computer deduce the logic behind it. On the other hand, *unsupervised learning* is a method for finding patterns and relationship in data sets without any prior knowledge of the system. Some authours also operate with a third category, namely *reinforcement learning*. This is a paradigm of learning inspired by behavioural psychology, where learning is achieved by trial-and-error, solely from rewards and punishment.

Another way to categorize machine learning tasks is to consider the desired output of a system. Some of the most common tasks are:

- Classification: Outputs are divided into two or more classes. The goal is to produce a model that assigns inputs into one of these classes. An example

is to identify digits based on pictures of hand-written ones. Classification is typically supervised learning.

- Regression: Finding a functional relationship between an input data set and a reference data set. The goal is to construct a function that maps input data to continuous output values.
- Clustering: Data are divided into groups with certain common traits, without knowing the different groups beforehand. It is thus a form of unsupervised learning.

Artificial neurons

The field of artificial neural networks has a long history of development, and is closely connected with the advancement of computer science and computers in general. A model of artificial neurons was first developed by McCulloch and Pitts in 1943 to study signal processing in the brain and has later been refined by others. The general idea is to mimic neural networks in the human brain, which is composed of billions of neurons that communicate with each other by sending electrical signals. Each neuron accumulates its incoming signals, which must exceed an activation threshold to yield an output. If the threshold is not overcome, the neuron remains inactive, i.e. has zero output.

This behaviour has inspired a simple mathematical model for an artificial neuron.

$$y = f\left(\sum_{i=1}^{n} w_i x_i\right) = f(u) \tag{1}$$

Here, the output y of the neuron is the value of its activation function, which have as input a weighted sum of signals $x_i, \ldots, x_n$ received by n other neurons.

Neural network types

An artificial neural network (NN), is a computational model that consists of layers of connected neurons, or *nodes*. It is supposed to mimic a biological nervous system by letting each neuron interact with other neurons by sending signals in the form of mathematical functions between layers. A wide variety of different NNs have been developed, but most of them consist of an input layer, an output layer and eventual layers in-between, called *hidden layers*. All layers can contain an arbitrary number of nodes, and each connection between two nodes is associated with a weight variable.

Feed-forward neural networks

The feed-forward neural network (FFNN) was the first and simplest type of NN devised. In this network, the information moves in only one direction: forward through the layers.

Nodes are represented by circles, while the arrows display the connections between the nodes, including the direction of information flow. Additionally, each arrow corresponds to a weight variable, not displayed here. We observe that each node in a layer is connected to *all* nodes in the subsequent layer, making this a so-called *fully-connected* FFNN.

A different variant of FFNNs are *convolutional neural networks* (CNNs), which have a connectivity pattern inspired by the animal visual cortex. Individual neurons in the visual cortex only respond to stimuli from small sub-regions of the visual field, called a receptive field. This makes the neurons well-suited to exploit the strong spatially local correlation present in natural images. The response of each neuron can be approximated mathematically as a convolution operation.

CNNs emulate the behaviour of neurons in the visual cortex by enforcing a *local* connectivity pattern between nodes of adjacent layers: Each node in a convolutional layer is connected only to a subset of the nodes in the previous layer, in contrast to the fully-connected FFNN. Often, CNNs consist of several convolutional layers that learn local features of the input, with a fully-connected layer at the end, which gathers all the local data and produces the outputs. They have wide applications in image and video recognition

Recurrent neural networks

So far we have only mentioned NNs where information flows in one direction: forward. *Recurrent neural networks* on the other hand, have connections between nodes that form directed *cycles*. This creates a form of internal memory which are able to capture information on what has been calculated before; the output is dependent on the previous computations. Recurrent NNs make use of sequential information by performing the same task for every element in a sequence, where each element depends on previous elements. An example of such information is sentences, making recurrent NNs especially well-suited for handwriting and speech recognition.

Other types of networks

There are many other kinds of NNs that have been developed. One type that is specifically designed for interpolation in multidimensional space is the radial basis function (RBF) network. RBFs are typically made up of three layers: an input layer, a hidden layer with non-linear radial symmetric activation functions and a linear output layer ("linear" here means that each node in the output layer has a linear activation function). The layers are normally fully-connected and there are no cycles, thus RBFs can be viewed as a type of fully-connected FFNN. They are however usually treated as a separate type of NN due the unusual activation functions.

Other types of NNs could also be mentioned, but are outside the scope of this work. We will now move on to a detailed description of how a fully-connected FFNN works, and how it can be used to interpolate data sets.

Mathematical model

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b_i\right) = f(u) \tag{2}$$

In an FFNN of such neurons, the *inputs* x_i are the *outputs* of the neurons in the preceding layer. Furthermore, a MLP is fully-connected, which means that each neuron receives a weighted sum of the outputs of *all* neurons in the previous layer.

Mathematical model

First, for each node i in the first hidden layer, we calculate a weighted sum u_i^1 of the input coordinates x_j,

$$u_i^1 = \sum_{j=1}^{2} w_{ij}^1 x_j + b_i^1 \tag{3}$$

This value is the argument to the activation function f_1 of each neuron i, producing the output y_i^1 of all neurons in layer 1,

$$y_i^1 = f_1(u_i^1) = f_1\left(\sum_{j=1}^{2} w_{ij}^1 x_j + b_i^1\right) \tag{4}$$

where we assume that all nodes in the same layer have identical activation functions, hence the notation f_l

$$y_i^l = f_l(u_i^l) = f_l\left(\sum_{j=1}^{N_{l-1}} w_{ij}^l y_j^{l-1} + b_i^l\right) \tag{5}$$

where N_l is the number of nodes in layer l. When the output of all the nodes in the first hidden layer are computed, the values of the subsequent layer can be calculated and so forth until the output is obtained.

Mathematical model

The output of neuron i in layer 2 is thus,

$$y_i^2 = f_2\left(\sum_{j=1}^{3} w_{ij}^2 y_j^1 + b_i^2\right) \tag{6}$$

$$= f_2\left[\sum_{j=1}^{3} w_{ij}^2 f_1\left(\sum_{k=1}^{2} w_{jk}^1 x_k + b_j^1\right) + b_i^2\right] \tag{7}$$

where we have substituted y_m^1 with. Finally, the NN output yields,

$$y_1^3 = f_3\left(\sum_{j=1}^{3} w_{1m}^3 y_j^2 + b_1^3\right) \tag{8}$$

$$= f_3\left[\sum_{j=1}^{3} w_{1j}^3 f_2\left(\sum_{k=1}^{3} w_{jk}^2 f_1\left(\sum_{m=1}^{2} w_{km}^1 x_m + b_k^1\right) + b_j^2\right) + b_1^3\right] \tag{9}$$

Mathematical model

We can generalize this expression to an MLP with l hidden layers. The complete functional form is,

$$y_1^{l+1} = f_{l+1}\left[\sum_{j=1}^{N_l} w_{1j}^3 f_l\left(\sum_{k=1}^{N_{l-1}} w_{jk}^2 f_{l-1}\left(\ldots f_1\left(\sum_{n=1}^{N_0} w_{mn}^1 x_n + b_m^1\right)\ldots\right) + b_k^2\right) + b_1^3\right] \tag{10}$$

which illustrates a basic property of MLPs: The only independent variables are the input values x_n.

Mathematical model

This confirms that an MLP, despite its quite convoluted mathematical form, is nothing more than an analytic function, specifically a mapping of real-valued vectors $\vec{x} \in \mathbb{R}^n \to \vec{y} \in \mathbb{R}^m$. In our example, $n = 2$ and $m = 1$. Consequentially, the number of input and output values of the function we want to fit must be equal to the number of inputs and outputs of our MLP.

Furthermore, the flexibility and universality of a MLP can be illustrated by realizing that the expression is essentially a nested sum of scaled activation functions of the form

$$h(x) = c_1 f(c_2 x + c_3) + c_4 \tag{11}$$

where the parameters c_i are weights and biases. By adjusting these parameters, the activation functions can be shifted up and down or left and right, change slope or be rescaled which is the key to the flexibility of a NN.

$$f_o = f(u_o) = u_o \tag{12}$$

Matrix-vector notation. We can introduce a more convenient notation for the activations in a NN.

Additionally, we can represent the biases and activations as layer-wise column vectors $\vec{b}_l$ and $\vec{y}_l$, so that the i-th element of each vector is the bias b_i^l and activation y_i^l of node i in layer l respectively.

We have that W_l is a $N_{l-1} \times N_l$ matrix, while $\vec{b}_l$ and $\vec{y}_l$ are $N_l \times 1$ column vectors. With this notation, the sum in becomes a matrix-vector multiplication, and we can write the equation for the activations of hidden layer 2 in

$$\vec{y}_2 = f_2(\mathrm{W}_2\vec{y}_1 + \vec{b}_2) = f_2\left(\begin{bmatrix} w_{11}^2 & w_{12}^2 & w_{13}^2 \\ w_{21}^2 & w_{22}^2 & w_{23}^2 \\ w_{31}^2 & w_{32}^2 & w_{33}^2 \end{bmatrix} \cdot \begin{bmatrix} y_1^1 \\ y_2^1 \\ y_3^1 \end{bmatrix} + \begin{bmatrix} b_1^2 \\ b_2^2 \\ b_3^2 \end{bmatrix}\right) \tag{13}$$

and we see that the activation of node i in layer 2 is

$$y_i^2 = f_2\Big(w_{i1}^2 y_1^1 + w_{i2}^2 y_2^1 + w_{i3}^2 y_3^1 + b_i^2\Big) = f_2\left(\sum_{j=1}^{3} w_{ij}^2 y_j^1 + b_i^2\right) \tag{14}$$

which is in accordance with. Note that This is not just a convenient and compact notation, but also a useful and intuitive way to think about MLPs: The output is calculated by a series of matrix-vector multiplications and vector additions that are used as input to the activation functions. For each operation $\mathrm{W}_l\vec{y}_{l-1}$ we move forward one layer.