Department of Mathematics and Computer Science
University of Southern Denmark, Odense

October 13, 2022
Marco Chiarandini

DM573, Introduction to Computer Science Autumn 2022

Lecture Notes on Machine Learning

Contents

In the task of machine learning that deals with *supervised learning*, we are given a set of observations consisting of values of some input variables (features) and values of a corresponding response, and the goal is to determine whether there is an association (for example, a function) between those input variables and the response. In other terms, we wish to determine an accurate model to describe those training data and then to use this model to predict the response on a new unseen input. Here, we look at two types of such models: *linear regression* and *neural networks*.

1 Linear Regression

1.1 Single input variable

Consider an event $y \in \mathcal{Y}$ that we assume depends on a variable $x \in \mathcal{X}$. For example, let y be the value of the final grade of a student in an exam and x the number of hours the student devoted to the study of the subject.

A *learning model* on a set of training samples $\{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}$ seeks a goal function $g : \mathcal{X} \to \mathcal{Y}$ that best approximates an unknown function f from which the training set is assumed to have been generated. Then, for all those situations, in which an input x is readily available, but the output y is unknown, we can predict y using $\hat{y} = g(x)$, where $\hat{y}$ represents the resulting prediction for y using the estimated function g.

In linear regression the set of candidate functions $\mathcal{H}$, from which g has to be selected, is *represented* by all functions h of the form $h_{a,b}(x) := ax + b$. Hence, the form of g is fixed while the parameters a and b can be adjusted to find a function from the family $\mathcal{H}$ that for any input x approximates well the response y.

The most common way to *evaluate* a function $h \in \mathcal{H}$ on any data point (x, y) is the *squared error* that accounts for the *loss* between the measured value y and the predicted value $\hat{y} = h_{a,b}(x)$:

$$L(y, h_{a,b}(x)) = (y - \hat{y})^2 = (y - h_{a,b}(x))^2. \tag{1}$$

Here, we considered (x, y) as variables to write a general model that is valid both for data already observed belonging to the training set and for data not yet observed like those we might be using

for the final assessment of the model or a prediction to use in practice. On the given set of m training samples we can evaluate a function $h_{a,b}$ by calculating the *total training error*:

$$\hat{L}_{a,b} = \sum_{i=1}^{m} \hat{L}(y_i, h_{a,b}(x_i)) = \sum_{i=1}^{m} (y_i - h_{a,b}(x_i))^2 = \sum_{i=1}^{m} (y_i - \hat{y}_i)^2. \tag{2}$$

Note that in Eq. (2), we used the hat symbol to indicate that, differently from Eq. (1), the values of the variables x and y are now known. (Hence, the hat symbol is used here with a different meaning than the one you might have used in the gymnasium to represent tvær vektor.)

Finding the function $g \in \mathcal{H}$ that *minimizes* $\hat{L}_{a,b}$ corresponds to finding the values of the parameters a and b for which $\hat{L}_{a,b}$ is minimum. Hence, now x_i and y_i are given and known, while a and b are unknown variables whose value we want to estimate. We could proceed by trial and error. However, in this case we can use Calculus and the theory of partial derivatives to find that the values of a and b that minimize $\hat{L}_{a,b}$ can be expressed in closed form. They are given in the slides. We do not need to remember those formulas as any software, including several Python modules (eg, scikit-learn), have them already implemented in their methods.

The regression analysis above can be enhanced by including:

- multiple input variables (features) $(x_1, x_2, \ldots, x_p)$
- higher degree terms to represent polynomial functions
- basis functions (generalization of polynomial functions, not treated here).

In all these cases the parameter estimation problem (not the function) remains linear and therefore relatively easy to solve. For this reason, these enhancements are all regarded as *Multiple linear regression.*

1.2 Multiple input variables

We consider now the case where instead of one single input variable we have several. In the exam grade example, beside the number of study hours we might suspect that also the age of the student has an influence on the final grade. So the data set consists of the values of two input variables, the number of study hours and the age, and the values of the corresponding response.

Consistently with the previous section, we use m to represent the number of distinct data points, or observations, in our training sample. We let p denote the number of variables that are available for use in making predictions. So, in our example, $p = 2$ and, if we collected these measurements for 100 students, then $m = 100$.

We let x_{ij} represent the value of the jth variable for the ith observation, where $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, p$. Throughout this document i is used to index the samples or observations (from 1 to m) and j is used to index the variables (from 1 to p).

We can write the measurements of an observation i for $i = 1, \ldots, m$ as a vector $\vec{x}_i$ of length p, containing the p variable values for the ith observation. That is,[1]

$$\vec{x}_i = \begin{bmatrix} x_{i1} & x_{i2} & \ldots & x_{ip} \end{bmatrix}.$$

For example, for the exam grade data, $\vec{x}_i$ is a vector of length 2, consisting of the values of age and study hours for the ith individual.

Then our observed data consists of $\{(\vec{x}_1, y_1), (\vec{x}_2, y_2), ..., (\vec{x}_m, y_m)\}$, where each $\vec{x}_i$ is a vector of length p. (If $p = 1$, then x_i is simply a scalar and the notation falls back to the one used in the previous section.)

We can subdivide the learning task in three parts.

[1] Vectors are by convention represented as columns and the ^{T} notation is used to denote the transpose of a vector. However, in this document we represent vectors as row vectors.

1. *Representation* of the hypothesis set $\mathcal{H}$. In linear regression, $\mathcal{H}$ is made by all linear models of the p features:

$$\begin{aligned} h(x_1, x_2, \ldots, x_p) &:= \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_p x_p \\ &= h_{\theta_0,\theta_1,\theta_2,\ldots,\theta_p}(x_1, x_2, \ldots, x_p) \end{aligned} \tag{3}$$

Eq. (3) can be written in a simpler way using the vector notation introduced above and recalling the definition of scalar product between vectors. Define $x_0 = 1$, $\vec{x} = \big[x_0, x_1, x_2, \ldots, x_p\big]$ and $\vec{\theta} = \big[\theta_0, \theta_1, \theta_2, \ldots, \theta_p\big]$, then we can rewrite (3) as:

$$h_{\vec{\theta}}(\vec{x}) = \vec{\theta} \cdot \vec{x} = \sum_{j=0}^{p} \theta_j x_j$$

The parameter θ_0 defining the intercept is called in machine learning the *bias*. The number of θ coefficients to determine for this linear model is $p + 1$.

For the exam grade case:

$$h(x_1, x_2) := \theta_0 + \theta_1 x_1 + \theta_2 x_2 = \sum_{j=0}^{2} \theta_j x_j = \vec{\theta} \cdot \vec{x} = h_{\vec{\theta}}(\vec{x}) = h_{\theta_0,\theta_1,\theta_2}(x_1, x_2)$$

and the number of θ coefficients is 3.

2. *Evaluation:* The loss function can be defined as:

$$L(y, h_{\vec{\theta}}(\vec{x})) = \big(y - h_{\vec{\theta}}(\vec{x})\big)^2$$

3. *Optimization*: We calculate the loss function on the training samples as the sum of squared errors:

$$\hat{L}_{\vec{\theta}} = \sum_{i=1}^{p} \big(y_i - h_{\vec{\theta}}(\vec{x}_i)\big)^2$$

and we look for the coefficients $\vec{\theta}$ that minimize the total training error $\hat{L}_{\vec{\theta}}$. In mathematical notation:

$$\min_{\vec{\theta}} \hat{L}_{\vec{\theta}}.$$

Again, with notions from Calculus and Linear Algebra the optimal values for $\vec{\theta}$ can be expressed in closed form as a function of the training sample data.

1.3 Higher degree polynomial functions

1. *Representation* of the hypothesis space $\mathcal{H}$. Let $\vec{x}^T = \big[1, x, x^2, \ldots, x^p\big]$ then

$$h_{\vec{\theta}}(x) = \text{poly}(\vec{\theta}, \vec{x}) = \theta_0 + \theta_1 x + \cdots + \theta_p x^p$$

where $p \leq m - 1$. Each term acts like a different variable in the previous case.

2. *Evaluation:* the loss function can be redefined as

$$L(y, h_{\vec{\theta}}(\vec{x})) = \Big(y - \text{poly}(\vec{\theta}, \vec{x})\Big)^2.$$

3. *Optimization:* $\hat{L}$ takes the form:

$$\hat{L}(\vec{\theta}) = \sum_{i=1}^{m} \Big(y_i - \text{poly}(\vec{\theta}, \vec{x}_i)\Big)^2,$$

which is a function of $p + 1$ coefficients $\theta_0, \cdots, \theta_p$. Minimizing in $\vec{\theta}$ the optimal value for these coefficients can again be expressed in closed form.

2 Artificial Neural Networks

2.1 Artificial neuron models: perceptron and sigmoid neurons

2.1.1 Perceptrons

A *perceptron* is a type of artificial neuron. It takes several binary inputs, $x_1, x_2, \ldots, x_p$ and produces a single binary output.

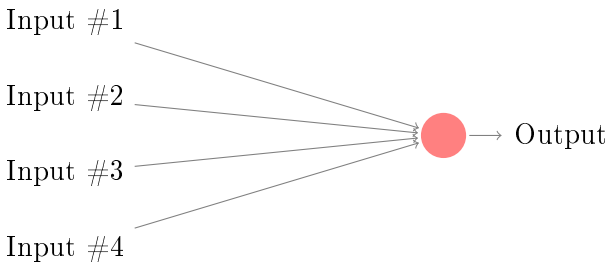

Figure 1: A Perceptron.

In the example shown in Figure 1 the perceptron has four inputs, x_1, x_2, x_3, x_4. In general it could have more or fewer inputs. In the 1950s and 1960s Warren McCulloch and Walter Pitts and later Frank Rosenblatt proposed a simple rule to compute the output. They introduced weights, $w_1, w_2, \ldots, w_p$, real numbers expressing the importance of the respective inputs to the output. The neuron's output, 0 or 1, is determined by whether the weighted sum of the inputs, $\sum_{j=1}^{p} w_j x_j$, is less than or greater than some threshold value. Just like the weights, the threshold is a real number which is a parameter of the neuron. To put it in more precise algebraic terms:

$$\text{output} := \begin{cases} 0 & \text{if } \sum_{j=1}^{p} w_j x_j \leq w_0 \\ 1 & \text{if } \sum_{j=1}^{p} w_j x_j > w_0 \end{cases} \tag{4}$$

We can simplify the way we describe perceptrons. The condition $\sum_{j=1}^{p} w_j x_j > w_0$ is cumbersome, and we can make two notational changes to simplify it. The first change is to write $\sum_{j=1}^{p} w_j x_j$ as a dot product, $\vec{w} \cdot \vec{x} = \sum_{j=1}^{p} w_j x_j$, where $\vec{w}$ and $\vec{x}$ are vectors whose components are the weights and inputs, respectively. The second change is to denote the threshold as w_0, to bring it to the left side of the inequality, to define $x_0 = -1$ and to redefine the vectors $\vec{w}$ and $\vec{x}$ such that they include also w_0 and x_0, respectively. Thus, we can finally write:

$$\text{output} := \begin{cases} 0 & \text{if } \vec{w} \cdot \vec{x} \leq 0 \\ 1 & \text{if } \vec{w} \cdot \vec{x} > 0 \end{cases} \tag{5}$$

The threshold w_0 has the same role as the intercept in the linear regression (see also Sec. 2.1.3). For consistency with the theory of linear regression we can represent w_0 by the *bias* w_0', and set $w_0' = -w_0$ and $x_0 = 1$. Introducing the bias is only a small change in how we describe perceptrons, but it simplifies more advanced theory on neural networks and it is therefore preferred to the use of a threshold. Note also that programs to compute the weights of neural networks return the bias rather than the threshold.[2] From the point of view of the application, you can think of the bias as a measure of how easy it is to get the perceptron to output a 1. Or to put it in more biological terms, the bias is a measure of how easy it is to get the perceptron to fire. For a perceptron with a really big bias, it's extremely easy for the perceptron to output a 1. But if the bias is negative with a large absolute value, then it's difficult for the perceptron to output a 1.

[2]However, in our exercises where calculations are to be carried out by hand we will consistently use the threshold notation.

The perceptron recalled above is only one type of artificial neuron. In class, we discussed also the *sigmoid (or logistic) neuron*. Sigmoid neurons are similar to perceptrons, but modified so that small changes in their weights and bias cause only a small change in their output. That's a crucial fact to allow a network of sigmoid neurons to learn.

2.1.2 Sigmoid neurons

Sigmoid neurons can be depicted in the same way as we depicted perceptrons in Figure 1.

Just like a perceptron, the sigmoid neuron has inputs, $x_1, x_2, \ldots, x_p$. But instead of being just 0 or 1, these inputs can also take on any values between 0 and 1. So, for instance, $0.638, 0.432, \ldots, 0.578$ is a valid input for a sigmoid neuron. Also just like a perceptron, the sigmoid neuron has weights for each input, $w_1, w_2, \ldots, w_p$ and an overall bias, w_0' for $x_0 = 1$. But the output is not 0 or 1. Instead, it is $\sigma(\vec{w} \cdot \vec{x})$, where σ is called the *sigmoid or logistic function*, and is defined by:

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

To put it all a little more explicitly, the output of a sigmoid neuron with inputs $x_1, x_2, \ldots, x_p$, weights $w_1, w_2, \ldots, w_p$ and threshold w_0 is

$$\text{output} := \frac{1}{1 + \exp\left(-\sum_{j=1}^{p} w_j x_j + w_0\right)}$$

To understand the similarity to the perceptron model, suppose $z := \vec{w} \cdot \vec{x}$ is a large positive number. Then $e^{-z} \approx 0$ and so $\sigma(z) \approx 1$. In other words, when $z := \vec{w} \cdot \vec{x}$ is large and positive, the output from the sigmoid neuron is approximately 1, just as it would have been for a perceptron. Suppose on the other hand that $z := \vec{w} \cdot \vec{x}$ is very negative. Then $e^{-z} = \infty$, and $\sigma(z) \approx 0$. So when $z := \vec{w} \cdot \vec{x}$ is very negative, the behavior of a sigmoid neuron also closely approximates a perceptron. It is only when $z := \vec{w} \cdot \vec{x}$ is of modest size that there's much deviation from the perceptron model.

How should we interpret the output from a sigmoid neuron? Obviously, one big difference between perceptrons and sigmoid neurons is that sigmoid neurons don't just output 0 or 1. They can have as output any real number between 0 and 1, so, for example, values such as 0.173 and 0.689 are legitimate outputs. This can be useful, for example, if we want to use the output value to represent the average intensity of the pixels in an image input to a neural network. But sometimes it can be a trouble. Suppose we want the output from the network to indicate either "the input image is a 9" or "the input image is not a 9". Obviously, it would be easiest to do this if the output was a 0 or a 1, as in a perceptron. But in practice we can set up a convention to deal with this, for example, by deciding to interpret any output of at least 0.5 as indicating a "9", and any output less than 0.5 as indicating "not a 9".

What about the algebraic form of σ? How can we understand that? In fact, the exact form of σ is not so important — what really matters is the shape of the function when plotted. The shape is shown in Figure 2, left. This shape is a smoothed out version of a step function shown in Figure 2, right. If σ had in fact been a step function, then the sigmoid neuron would be a perceptron, since the output would be 1 or 0 depending on whether $\vec{w} \cdot \vec{x}$ was positive or negative. By using the actual σ function we get a smoothed out perceptron. Indeed, it is the smoothness of the σ function that is the crucial fact, not its detailed form.

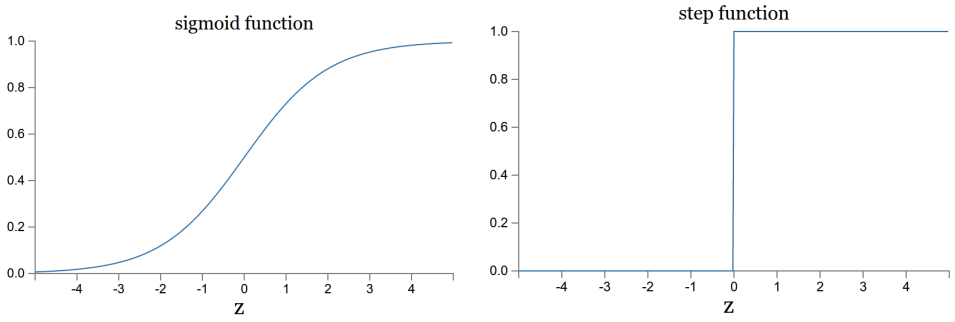

Figure 2: The graph of a sigmoid function, left, and of a step function, right.

2.1.3 Linear separators

In a binary classification task, the *single* neuron implements a *linear separator* in the space of the input variables. Indeed, for a perceptron the decision boundary is

$$\sum_{j=1}^{p} w_j x_j = w_0$$

That is, if the left hand side of the equation above is less or equal than the threshold then the neuron outputs 0, otherwise it outputs 1.

In the case of two inputs, x_1 and x_2, this becomes:

$$w_1 x_2 + w_2 x_2 = w_0,$$

which corresponds to the equation of a line in the Cartesian plane:

$$x_2 = -\frac{w_1}{w_2} x_1 + \frac{1}{w_2} w_0$$

(you might have seen this with y in place of x_2 and x in place of x_1.) In 3 dimensions the equation:

$$w_1 x_2 + w_2 x_2 + w_3 x_3 = \text{constant}$$

represents a plane. In more dimensions the equation represents what is called an hyperplane. In all cases, the equation remains linear in $\vec{x}$ and therefore it is called *linear separator*.

A sigmoid neuron uses most commonly the value 0.5 as the discriminant for outputting 1 or 0. Then the decision boundary becomes:

$$\frac{1}{1 + \exp\left(-\sum_{j=1}^{p} w_j x_j - w_0'\right)} = 0.5$$

Solving in $\vec{x}$ we obtain an equation of the form:

$$\sum_{j=1}^{p} w_j x_j = \text{constant} + w_0' = \text{constant}$$

which is again linear in $\vec{x}$. Therefore, also the *single* sigmoid neuron is a linear separator.

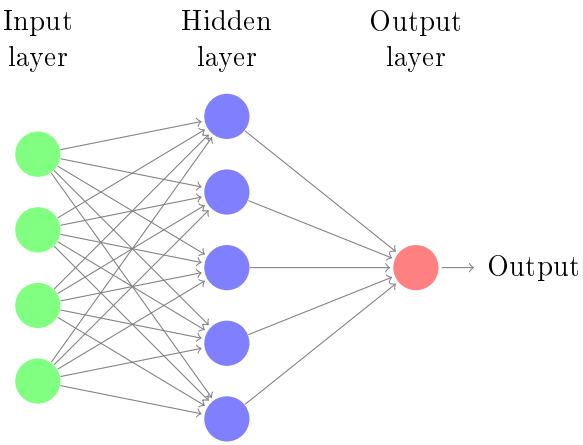

Figure 3: The structure of neural networks.

2.2 Multilayer neural networks

Figure 3 shows a general structure of a feed-forward neural network. Somewhat confusingly, and for historical reasons, such *multiple layer networks* are sometimes called *multilayer perceptrons* or MLPs, despite being made up of sigmoid neurons, not perceptrons. The latter use a step function as activation function while the former use a sigmoid (also called logistic) function.

The leftmost layer in this network is called the *input layer* and the neurons within the layer are called *input neurons*. The notation for input neurons, in which we have an output, but no inputs, is a shorthand. It doesn't actually mean a perceptron with no inputs. It's better to think of input neurons as not really being neurons at all, but rather special units which are simply defined to output the desired values, which are the inputs to the network. The rightmost or *output layer* contains the *output neurons*, or, as in this case, a single output neuron. The middle layer is called a *hidden layer*, since the neurons in this layer are neither inputs nor outputs. The term "hidden" perhaps sounds a little mysterious but it really means nothing more than "not an input or an output". The network in the figure has just a single hidden layer, but some networks can have multiple hidden layers.

Most commonly, in computer programs that implement neural networks, all nodes of the hidden layers are sigmoid neurons whose outputs are left unchanged from the sigmoid function and lay in the interval $[0, 1]$. The output neurons are, instead, defined depending on the task. In a binary classification task, a single sigmoid neuron can provide a probability measure for the prediction to be one or the other class. In multi-way classification, the output of a single sigmoid neuron can be assessed by a partition of the interval $[0, 1]$. Alternatively, there can be as many sigmoid neurons as there are classes and the prediction decided by the neuron that returns the largest value. In a regression task, a single output neuron can implement an identity function or, equivalently, a linear function. In general, these decisions that determine the actual structure or architecture of the neural network are passed as parameters to the programs.

References

The part on the neural networks is largely based on: Michael A. Nielsen, "Neural Networks and Deep Learning", Determination Press, 2015. `http://neuralnetworksanddeeplearning.com/chap1.html`