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

October 12, 2016

Marco Chiarandini

DM534 - Introduction to Computer Science

Lecture Notes on Machine Learning, Autumn 2016

The overall goal in the form of machine learning called *supervised learning* is to train a model to best fit a set of training data made of some input variables (features) and a response. The hope is then that such a model applied to a new unseen input will be able to predict well the corresponding response. Here, we look at two types of such models: linear regression and neural networks.

1 Linear Regression

(Based on slides 22-38.)
Consider an event $y \in \mathcal{Y}$ that we assume depends on a variable $x \in \mathcal{X}$. For example, let y be 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 $(x_1, y_1), \ldots, (x_m, y_m)$ 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. 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 well approximate the response y. (Note that we could write $ax + b$ also as $\vec{\theta} \cdot \vec{x}$ where $\vec{\theta}$ is the vector $\begin{bmatrix} a & b \end{bmatrix}$ and $\vec{x}$ is the vector $\begin{bmatrix} x & 1 \end{bmatrix}$. This notation simplifies considerably the formulas in more advanced models.)

The most common way to *evaluate* a function $h \in \mathcal{H}$ on any data point (x, y) is the squared error loss:

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

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. On the given set of data we can evaluate h with any parameters a and b by calculating the total error:

$$\hat{L}_{a,b} = \sum_{j=1}^{m} L(y_j, h_{a,b}(x_j)) = \sum_{j=1}^{m} (y_j - h_{a,b}(x_j))^2.$$

(Note that we used here the hat symbol as common in machine learning but this has nothing to do with the symbol you might have seen in the gymnasium for tvær vektor.)

To find the function $g \in \mathcal{H}$ that *optimizes*, here minimizes, $\hat{L}_{a,b}$ corresponds to find the values of the parameters a and b for which $\hat{L}_{a,b}$ is minimum. 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 on slide 26. We do not need to remember those formulas as any software, including several Python modules, have them already implemented in their methods.

The linear model above can be enhanced by including:

- several input variables (features) $(x_1, x_2, \ldots, x_p)$
- higher degree terms to represent polynomial functions
- basis functions.

Several input variables Let p be the number of features, $\vec{\theta}$ a vector of $p + 1$ parameters, where θ_0 is called the bias. Let x_{ij} be the value of feature i for sample j, for $i = 1..p, j = 1..m$. Finally, let y_j be the value of the response for sample $j = 1..m$.
Representation of hypothesis space $\mathcal{H}$:

$$h(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 = h_{\vec{\theta}}(x_1, x_2)$$

For conciseness, defining $x_0 = 1$ and $\vec{x} = \begin{bmatrix} x_0 & x_1 & x_2 \end{bmatrix}$:

$$h_{\vec{\theta}}(\vec{x}) = \vec{\theta} \cdot \vec{x} = \sum_{i=0}^{2} \theta_i x_i$$

The *evaluation* is again done with the loss function

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

The *optimization* stage seeks the parameters $\vec{\theta}$ that minimize:

$$\hat{L}_{\vec{\theta}} = \sum_{j=1}^{p} \left(y_j - h_{\vec{\theta}}(\vec{x}_j)\right)^2$$

or equivalently:

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

Since this function is linear in $\vec{\theta}$ a closed form solution exists. It requires notions from linear algebra, hence we do not write it here.

Higher degree polynomial functions *Representation* of hypothesis space $\mathcal{H}$:

$$h_{\vec{\theta}}(x) = p(\vec{\theta}, \vec{x}) = \theta_0 + \theta_1 x + \cdots + \theta_k x^k$$

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

$$\vec{x} = \begin{bmatrix} 1 & x & x^2 & \dots & x^k \end{bmatrix}^T$$

Evaluation: $\hat{L}$ takes the form:

$$\hat{L}(\vec{\theta}) = \sum_{j=1}^{m} (y_j - p(\vec{\theta}, \vec{x}_j))^2$$

which is a function of $k + 1$ parameters $\theta_0, \cdots, \theta_k$.
Optimization: Since the function is again linear in $\vec{\theta}$ the values of the optimal parameters can also be expressed in closed form.

Basis Functions
Basis functions combine several variables with a fixed set of nonlinear functions.
Representation:

$$h_{\vec{\theta}}(\vec{x}) = \theta_0 + \sum_{i=1}^{p} \theta_i x_i + \sum_{i=1}^{p} \sum_{k=1}^{p} \theta_{ik} x_i x_k + \sum_{i=1}^{p} \sum_{k=1}^{p} \sum_{\ell=1}^{p} \theta_{ik\ell} x_i x_k x_\ell$$

$$h_{\vec{\theta}}(\vec{x}) = \theta_0 + \sum_{j=1}^{p} \theta_j \phi_j(x) = \vec{\theta} \cdot \vec{\phi}(\vec{x})$$

Each function h is now a nonlinear function of the input vector $\vec{x}$ but h is linear in $\vec{\theta}$. Hence, the loss function is again solvable in closed form.

2 Artificial Neuron Models: Perceptron and Sigmoid Neurons

(Based on slides 49-53.)

2.1 Perceptrons

A *perceptron* is a type of artificial neuron. It takes several binary inputs, $x_1, x_2, \ldots$ and produces a single binary output.
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, $w_1, w_2, \ldots$, real numbers expressing the importance of the respective inputs to the output. The neuron's output, 0 or 1, is determined by whether

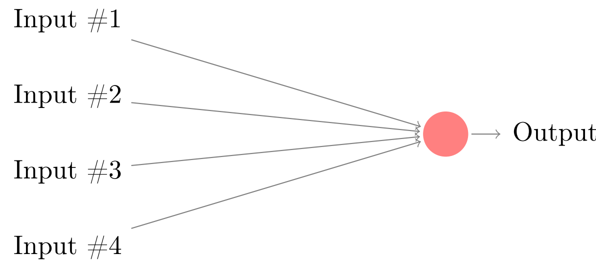

Figure 1: A Perceptron.

the weighted sum $\sum_j w_j x_j$ 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:

$$\text{output} := \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{threshold} \end{cases} \tag{1}$$

We can simplify the way we describe perceptrons. The condition $\sum_j w_j x_j >$ threshold is cumbersome, and we can make two notational changes to simplify it. The first change is to write $\sum_j w_j x_j$ as a dot product, $\vec{w} \cdot \vec{x} = \sum_j w_j x_j$, where $\vec{w}$ and $\vec{x}$ are vectors whose components are the weights and inputs, respectively. The second change is to move the threshold to the other side of the inequality, and to replace it by what's known as the *perceptron's bias*, $b = -$threshold. Using the bias instead of the threshold, the perceptron rule can be rewritten:

$$\text{output} := \begin{cases} 0 & \text{if } \vec{w} \cdot \vec{x} \leq 0 \\ 1 & \text{if } \vec{w} \cdot \vec{x} > 0 \end{cases} \tag{2}$$

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 very negative, then it's difficult for the perceptron to output a 1. Obviously, 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.
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.

2.2 Sigmoid Neurons

Sigmoid neurons can be depicted in the same way we depicted perceptrons in Figure 1.
Just like a perceptron, the sigmoid neuron has inputs, $a1, a2, \ldots$. But instead of being just 0 or 1, these inputs can also take on any values between 0 and 1. So, for instance, $0.638, \ldots$ is a valid input for a sigmoid neuron. Also just like a perceptron, the sigmoid neuron has weights for each input, $w_1, w_2, \ldots$ and an overall bias, b. But the output is not 0 or 1. Instead, it's $\sigma(\vec{w} \cdot \vec{x} + b)$, 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 $x_1, x_2, \ldots$ weights $w_1, w_2, \ldots$ and bias b is

$$\frac{1}{1 + \exp(-\sum_j w_j x_j - b)}$$

To understand the similarity to the perceptron model, suppose $z := \vec{w} \cdot \vec{x} + b$ is a large positive number. Then $e^{-z} \approx 0$ and so $\sigma(z) \approx 1$. In other words, when $z := \vec{w} \cdot \vec{x} + b$ 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 $z := \vec{w} \cdot \vec{x} + b$ is very negative. Then $e^{-z} = \infty$, and $\sigma(z) \approx 0$. So when $z := \vec{w} \cdot \vec{x} + b$ is very negative, the behaviour of a sigmoid neuron also closely approximates a perceptron. It's only when $z := \vec{w} \cdot \vec{x} + b$ 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

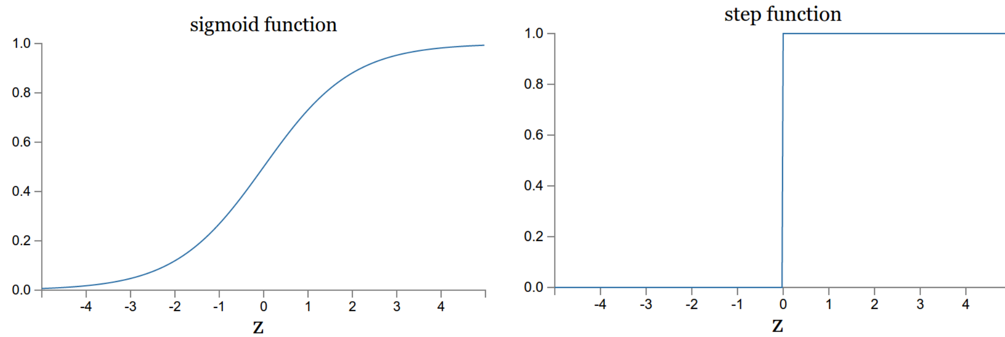

Figure 2: The graph of a sigmoid function, left, and of a step function, right.

number between 0 and 1, so 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 nuisance. 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'd be easiest to do this if the output was a 00 or a 11, 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 0.50.5 as indicating a "9", and any output less than 0.50.5 as indicating "not a 9".

What about the algebraic form of σ? How can we understand that? In fact, the exact form of σ isn't 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, as already implied above, a smoothed out perceptron. Indeed, it's the smoothness of the σ function that is the crucial fact, not its detailed form.

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

$$\sum_j w_j x_j = \text{threshold}$$

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:

$$w_1 x_2 + w_2 x_2 = \text{threshold},$$

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

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

(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:

$$\frac{1}{1 + \exp(-\sum_j w_j x_j - b)} = 0.5$$

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

$$\sum_j w_j x_j = \text{constant} - b$$

which therefore is also a linear separator in the same way as for the perceptron.

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`