CHAPTER 3

From Linear Models to Multi-layer Perceptrons

3.1 LIMITATIONS OF LINEAR MODELS: THE XOR PROBLEM

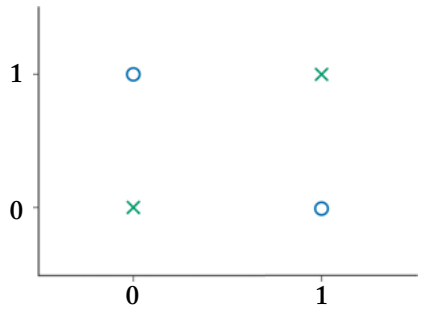
The hypothesis class of linear (and log-linear) models is severely restricted. For example, it cannot represent the XOR function, defined as:

$$\text{xor}(0,0) = 0$$
$$\text{xor}(1,0) = 1$$
$$\text{xor}(0,1) = 1$$
$$\text{xor}(1,1) = 0.$$

That is, there is no parameterization $\boldsymbol{w} \in \mathbb{R}^2, b \in \mathbb{R}$ such that:

$$(0,0) \cdot \boldsymbol{w} + b < 0$$
$$(0,1) \cdot \boldsymbol{w} + b \geq 0$$
$$(1,0) \cdot \boldsymbol{w} + b \geq 0$$
$$(1,1) \cdot \boldsymbol{w} + b < 0.$$

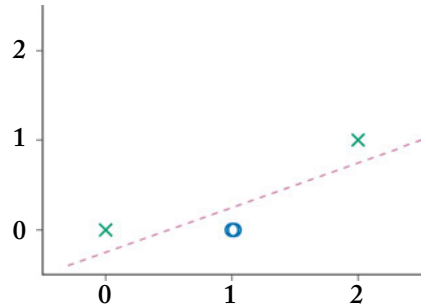
To see why, consider the following plot of the XOR function, where blue Os denote the positive class and green Xs the negative class.



It is clear that no straight line can separate the two classes.

3.2 NONLINEAR INPUT TRANSFORMATIONS

However, if we transform the points by feeding each of them through the nonlinear function $\phi(x_1, x_2) = [x_1 \times x_2, x_1 + x_2]$, the XOR problem becomes linearly separable.



The function ϕ mapped the data into a representation that is suitable for linear classification. Having ϕ at our disposal, we can now easily train a linear classifier to solve the XOR problem.

$$\hat{\boldsymbol{y}} = f(\boldsymbol{x}) = \phi(\boldsymbol{x})\boldsymbol{W} + \boldsymbol{b}.$$

In general, one can successfully train a linear classifier over a dataset which is not linearly separable by defining a function that will map the data to a representation in which it is linearly separable, and then train a linear classifier on the resulting representation. In the XOR example the transformed data has the same dimensions as the original one, but often in order to make the data linearly separable one needs to map it to a space with a much higher dimension.

This solution has one glaring problem, however: we need to manually define the function ϕ, a process which is dependent on the particular dataset, and requires a lot of human intuition.

3.3 KERNEL METHODS

Kernelized Support Vectors Machines (SVMs) [Boser and et al., 1992], and Kernel Methods in general [Shawe-Taylor and Cristianini, 2004], approach this problem by defining a set of generic mappings, each of them mapping the data into very high dimensional—and sometimes even infinite—spaces, and then performing linear classification in the transformed space. Working in very high dimensional spaces significantly increase the probability of finding a suitable linear separator.

One example mapping is the *polynomial mapping*, $\phi(\boldsymbol{x}) = (\boldsymbol{x})^d$. For $d = 2$, we get $\phi(x_1, x_2) = (x_1x_1, x_1x_2, x_2x_1, x_2x_2)$. This gives us all combinations of the two variables, allowing to solve the XOR problem using a linear classifier, with a polynomial increase in the number of parameters. In the XOR problem the mapping increased the dimensionality of the input (and

hence the number of parameters) from 2–4. For the language identification example, the input dimensionality would have increased from 784 to 784^2 =614,656 dimensions.

Working in very high dimensional spaces can become computationally prohibitive, and the ingenuity in kernel methods is the use of the *kernel trick* [Aizerman et al., 1964, Schölkopf, 2001] that allows one to work in the transformed space without ever computing the transformed representation. The generic mappings are designed to work on many common cases, and the user needs to select the suitable one for its task, often by trial and error. A downside of the approach is that the application of the kernel trick makes the classification procedure for SVMs dependent linearly on the size of the training set, making it prohibitive for use in setups with reasonably large training sets. Another downside of high dimensional spaces is that they increase the risk of overfitting.

3.4 TRAINABLE MAPPING FUNCTIONS

A different approach is to define a *trainable* nonlinear mapping function, and train it in conjunction with the linear classifier. That is, finding the suitable representation becomes the responsibility of the training algorithm. For example, the mapping function can take the form of a parameterized linear model, followed by a nonlinear activation function g that is applied to each of the output dimensions:

$$\begin{aligned} \hat{\boldsymbol{y}} &= \phi(\boldsymbol{x})\boldsymbol{W} + \boldsymbol{b} \\ \phi(\boldsymbol{x}) &= g(\boldsymbol{x}\boldsymbol{W}' + \boldsymbol{b}'). \end{aligned} \tag{3.1}$$

By taking $g(x) = \max(0, x)$ and $\boldsymbol{W}' = \left(\begin{smallmatrix} 1 & 1 \\ 1 & 1 \end{smallmatrix}\right)$, $\boldsymbol{b}' = (-1\ 0)$ we get an equivalent mapping to $(x_1 \times x_2, x_1 + x_2)$ for the our points of interest (0,0), (0,1), (1,0), and (1,1), successfully solving the XOR problem. The entire expression $g(\boldsymbol{x}\boldsymbol{W}' + \boldsymbol{b}')\boldsymbol{W} + \boldsymbol{b}$ is differentiable (although not convex), making it possible to apply gradient-based techniques to the model training, learning both the representation function and the linear classifier on top of it at the same time. This is the main idea behind deep learning and neural networks. In fact, Equation (3.1) describes a very common neural network architecture called a *multi-layer perceptron* (MLP). Having established the motivation, we now turn to describe multi-layer neural networks in more detail.