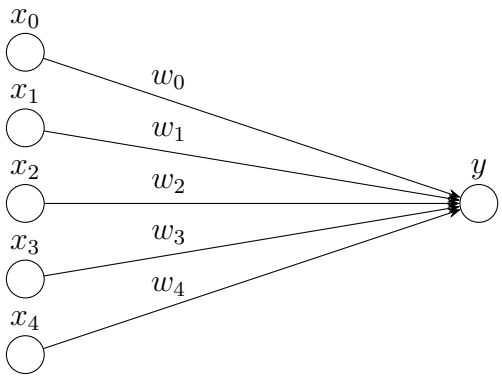
Chapter 1

Introduction

1.1 Neural Networks

Neural networks represent a mathematical tool in machine learning that is useful for performing function approximation; they can be thought of as a generalization of linear regression. The power of a neural network stems from three main properties that we will go over: nonliterary, differentiability, and hidden layers. These key properties allow neural networks to be able to improve its approximation of a set of values via "training."

To kick things off we will start with a simple example of a neural network. In the simplest form, a neural network consists of a **vector input** $\vec{x}$, a set of **weights** w_i, and a final **output** y. Below is a graphical representation of a such network.



In such a graphical representation, the set of x_i nodes are called the **input layer** (or simply the first layer) and the y node is called the **output layer**. In the above diagram, the output layer consists of one node, but as we will see it can also consist of multiple nodes.

The output is obtained as a function of the input and weights as below.

$$y = \sum_i w_i x_i$$

From a statistics perspective, this is actually just a **linear model**. In statistics, one "trains" such a linear model via linear regression on some dataset. If you have taken a statistics course, you might remember that this strategy works on simple examples (e.g., a suspiciously-already-linear weather

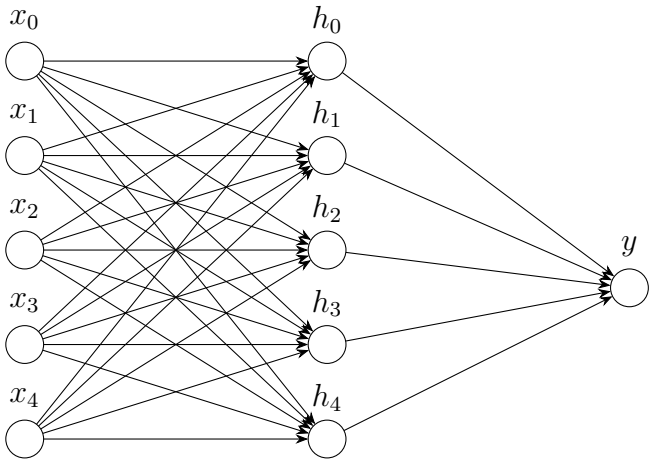
dataset in a Pearson textbook), but linear models do not generalize and often fail to capture complex behavior.

As we will see, neural networks are different from linear models since they add properties of hidden layers and nonlinearity.

1.2 Hidden Layers

Neural networks extend our previous notion of a linear model via **hidden layers**, which can be defined as one or more layers between the input and output layer. Below, we have a neural network which has one hidden layer. The hidden layer can have a variable number of nodes.

In this example, each input node w_i connects to each node h_j in the hidden layer via a weight w_{ij}. These weights are illustrated by the arrows, although we are choosing to suppress the w_{ij} notation in the diagram below to not over complicate the figure.



The calculation of a hidden layer is simply

$$h_j = \sum_i w_{ij} x_i$$

Intuitively, this means that each input value x_i makes a weighted contribution of w_{ij} to the value h_j. Something to observe at this point is that we can summarize the entire hidden layer calculation as a matrix equation:

$$\vec{h} = \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \end{bmatrix} = \begin{bmatrix} \sum_i w_{i1} x_i \\ \sum_i w_{i2} x_i \\ \sum_i w_{i3} x_i \\ \sum_i w_{i4} x_i \\ \sum_i w_{i5} x_i \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} & w_{15} \\ w_{21} & w_{22} & w_{23} & w_{24} & w_{25} \\ w_{31} & w_{32} & w_{33} & w_{34} & w_{35} \\ w_{41} & w_{42} & w_{43} & w_{44} & w_{45} \\ w_{51} & w_{52} & w_{53} & w_{54} & w_{55} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = W\vec{x} \tag{1.1}$$

This suggest the concept of a **weight matrix** W, which is the key to calculating the hidden layer $\vec{h}$ from the input $\vec{x}$.

As we can see, between each layer we have a matrix of weights. We can denote U as the matrix

of the weights between the output layer and the hidden layer, so that $y = U\vec{h}$. However, note that

$$y = U\vec{h} = UW\vec{x} = W'\vec{x}$$

where $W' = UW$. This reduces our above network, with three layers, to a boring network, with two layers (similar to the one we started with), just with a different weight matrix W'. The reason is because our network is linear, which means no matter how many layers we add it will always reduce to the same boring network we started with. Thus in order to get something interesting with hidden layers we need to add some nonlinearity.

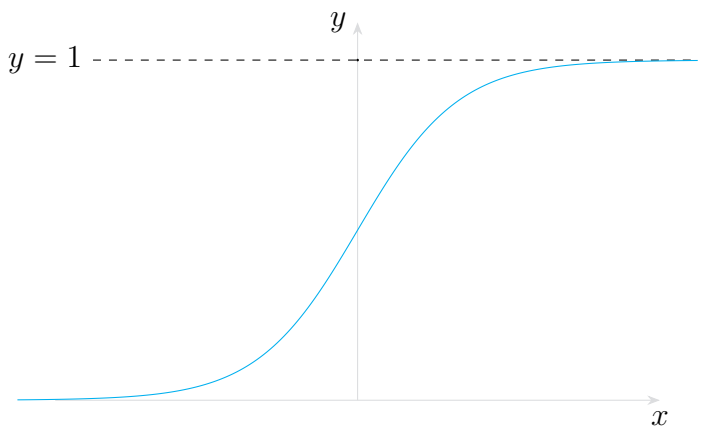
1.3 Nonlinearity

Neural networks achieve our desired property of nonlinearity via usage of **activation fuctions**. A few common examples of such functions are the sigmoid, tanh or RELU functions, and the choice of an activation function depends on what one is trying to achieve with a neural network.

The sigmoid function is given by

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

The graph of the sigmoid function is given below.



Using our previous network, we can modify

1.4 Backpropagation