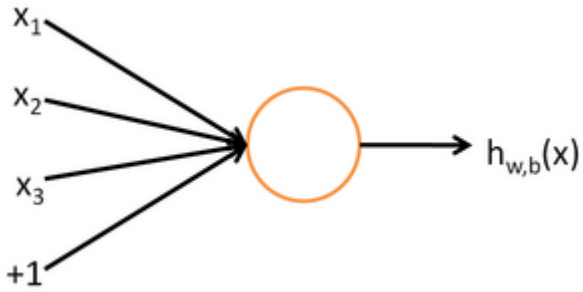
Neural Networks

From Ufldl

Consider a supervised learning problem where we have access to labeled training examples $(x^{(i)},y^{(i)})$. Neural networks give a way of defining a complex, non-linear form of hypotheses $h_{W,b}(x)$, with parameters W,b that we can fit to our data.

To describe neural networks, we will begin by describing the simplest possible neural network, one which comprises a single "neuron." We will use the following diagram to denote a single neuron:



This "neuron" is a computational unit that takes as input x_1,x_2,x_3 (and a +1 intercept term), and outputs $h_{W,b}(x) = f(W^T x) = f(\sum_{i=1}^{3} W_i x_i + b)$, where $f : \Re \mapsto \Re$ is called the **activation function**. In these notes, we will choose $f(\cdot)$ to be the sigmoid function:

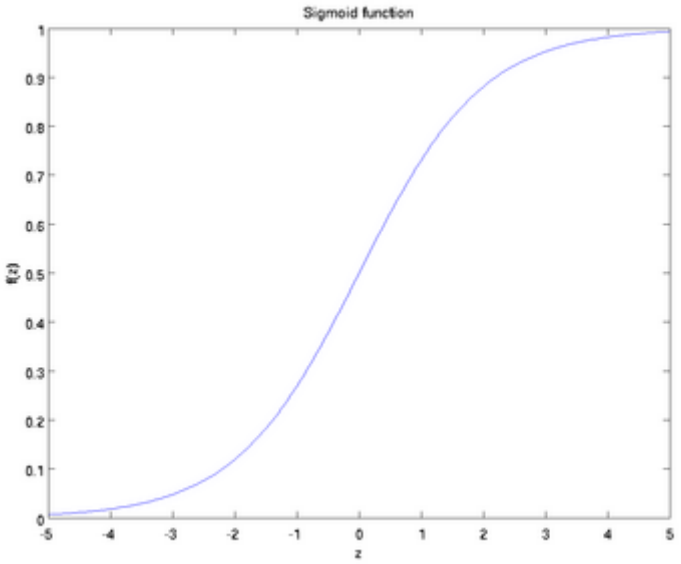
$$f(z) = \frac{1}{1+\exp(-z)}.$$

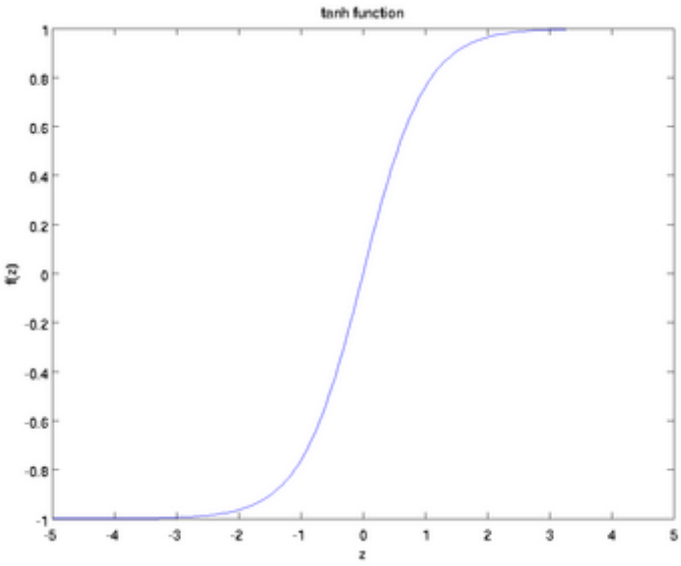
Thus, our single neuron corresponds exactly to the input-output mapping defined by logistic regression.

Although these notes will use the sigmoid function, it is worth noting that another common choice for f is the hyperbolic tangent, or tanh, function:

$$f(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}},$$

Here are plots of the sigmoid and tanh functions:





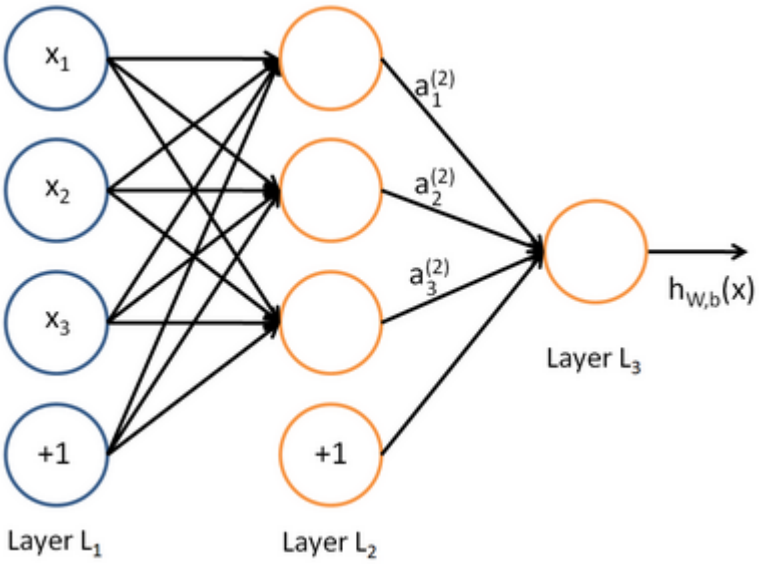
The tanh(z) function is a rescaled version of the sigmoid, and its output range is $[-1,1]$ instead of $[0,1]$.

Note that unlike some other venues (including the OpenClassroom videos, and parts of CS229), we are not using the convention here of $x_0 = 1$. Instead, the intercept term is handled separately by the parameter b.

Finally, one identity that'll be useful later: If $f(z) = 1 / (1 + \exp(-z))$ is the sigmoid function, then its derivative is given by $f'(z) = f(z)(1 - f(z))$. (If f is the tanh function, then its derivative is given by $f'(z) = 1 - (f(z))^2$.) You can derive this yourself using the definition of the sigmoid (or tanh) function.

Neural Network model

A neural network is put together by hooking together many of our simple "neurons," so that the output of a neuron can be the input of another. For example, here is a small neural network:



In this figure, we have used circles to also denote the inputs to the network. The circles labeled "+1" are called **bias units**, and correspond to the intercept term. The leftmost layer of the network is called the **input layer**, and the rightmost layer the **output layer** (which, in this example, has only one node). The middle layer of nodes is called the **hidden layer**, because its values are not observed in the training set. We also say that our example neural network has 3 **input units** (not counting the bias unit), 3 **hidden units**, and 1 **output unit**.

We will let n_l denote the number of layers in our network; thus $n_l = 3$ in our example. We label layer l as L_l, so layer L_1 is the input layer, and layer L_{n_l} the output layer. Our neural network has parameters $(W,b) = (W^{(1)},b^{(1)},W^{(2)},b^{(2)})$, where we write $W_{ij}^{(l)}$ to denote the parameter (or weight) associated with the connection between unit j in layer l, and unit i in layer $l+1$. (Note the order of the indices.) Also, $b_i^{(l)}$ is the bias associated with unit i in layer $l+1$. Thus, in our example, we have $W^{(1)} \in \Re^{3\times 3}$, and $W^{(2)} \in \Re^{1\times 3}$. Note that bias units don't have inputs or connections going into them, since they always output the value +1. We also let s_l denote the number of nodes in layer l (not counting the bias unit).

We will write $a_i^{(l)}$ to denote the **activation** (meaning output value) of unit i in layer l. For $l = 1$, we also use $a_i^{(1)} = x_i$ to denote the i-th input. Given a fixed setting of the parameters W,b, our neural network defines a hypothesis $h_{W,b}(x)$ that outputs a real number. Specifically, the computation that this neural network represents is given by:

$$
\begin{aligned}
a_1^{(2)} &= f(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)}) \\
a_2^{(2)} &= f(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_2^{(1)}) \\
a_3^{(2)} &= f(W_{31}^{(1)}x_1 + W_{32}^{(1)}x_2 + W_{33}^{(1)}x_3 + b_3^{(1)}) \\
h_{W,b}(x) = a_1^{(3)} &= f(W_{11}^{(2)}a_1^{(2)} + W_{12}^{(2)}a_2^{(2)} + W_{13}^{(2)}a_3^{(2)} + b_1^{(2)})
\end{aligned}
$$

In the sequel, we also let $z_i^{(l)}$ denote the total weighted sum of inputs to unit i in layer l, including the bias term (e.g., $z_i^{(2)} = \sum_{j=1}^{n} W_{ij}^{(1)}x_j + b_i^{(1)}$), so that $a_i^{(l)} = f(z_i^{(l)})$.

Note that this easily lends itself to a more compact notation. Specifically, if we extend the activation function $f(\cdot)$ to apply to vectors in an element-wise fashion (i.e., $f([z_1,z_2,z_3]) = [f(z_1),f(z_2),f(z_3)]$), then we can write the equations above more compactly as:

$$
\begin{aligned}
z^{(2)} &= W^{(1)}x + b^{(1)} \\
a^{(2)} &= f(z^{(2)}) \\
z^{(3)} &= W^{(2)}a^{(2)} + b^{(2)} \\
h_{W,b}(x) &= a^{(3)} = f(z^{(3)})
\end{aligned}
$$

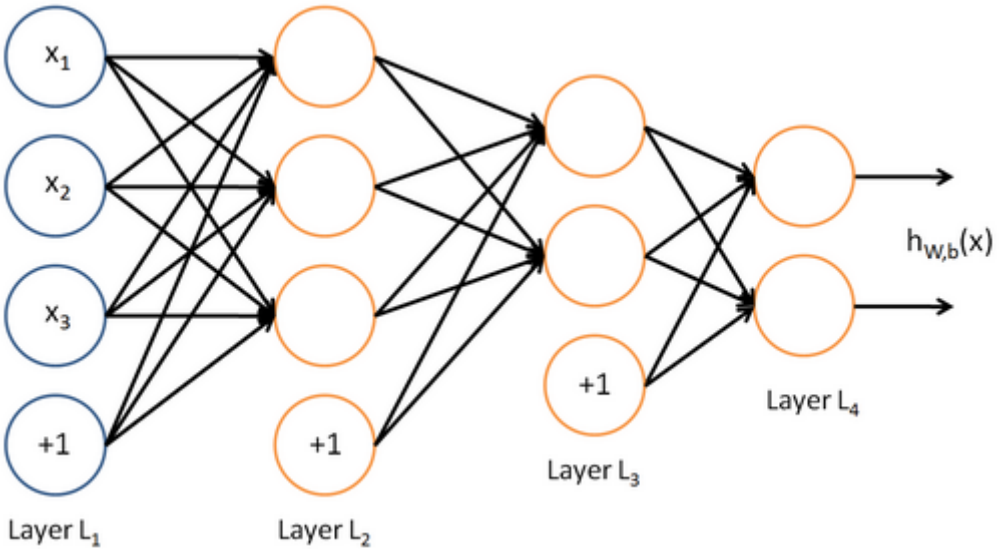
We call this step **forward propagation.** More generally, recalling that we also use $a^{(1)} = x$ to also denote the values from the input layer, then given layer l's activations $a^{(l)}$, we can compute layer $l + 1$'s activations $a^{(l + 1)}$ as:

$$
\begin{aligned}
z^{(l+1)} &= W^{(l)}a^{(l)} + b^{(l)} \\
a^{(l+1)} &= f(z^{(l+1)})
\end{aligned}
$$

By organizing our parameters in matrices and using matrix-vector operations, we can take advantage of fast linear algebra routines to quickly perform calculations in our network.

We have so far focused on one example neural network, but one can also build neural networks with other **architectures** (meaning patterns of connectivity between neurons), including ones with multiple hidden layers. The most common choice is a n_l-layered network where layer 1 is the input layer, layer n_l is the output layer, and each layer l is densely connected to layer $l + 1$. In this setting, to compute the output of the network, we can successively compute all the activations in layer L_2, then layer L_3, and so on, up to layer L_{n_l}, using the equations above that describe the forward propagation step. This is one example of a **feedforward** neural network, since the connectivity graph does not have any directed loops or cycles.

Neural networks can also have multiple output units. For example, here is a network with two hidden layers layers L_2 and L_3 and two output units in layer L_4:



To train this network, we would need training examples $(x^{(i)}, y^{(i)})$ where $y^{(i)} \in \Re^2$. This sort of network is useful if there're multiple outputs that you're interested in predicting. (For example, in a medical diagnosis application, the vector x might give the input features of a patient, and the different outputs y_i's might indicate presence or absence of different diseases.)

Language : 中文