Artificial Neural Networks

Kristian Wichmann

November 12, 2016

1 Overview of artificial neural networks

An *artificial neural network* (ANN) superficially mimics the workings of the brain by being made up of *neurons* and *dendrites/axons*[1].

The neurons are organized into *layers*. These come in three categories:

- The *input layer*, into which data the data to be processed enters.
- *Hidden layers* - which are layers between the input and output layers.
- The *output layer* in which the result of the ANN's data processing can be read.

An ANN has at least three layer: Input, output and at least one hidden layer. An ANN with two or more hidden layers is known as a *deep network*.

Figure 1 shows a neural network with three laters: 3 cells in the input layer, 4 cells in the hidden layer, and 2 cells in the output layer.

The cells in each layer are connected by dendrites/axons, which are represented as arrows in figure 1: Every cell in each layer is connected to all the cells in the next layer.

[1]In real neural networks, dendrites carry inputs to the neuron, while axons carry outputs from the neuron.

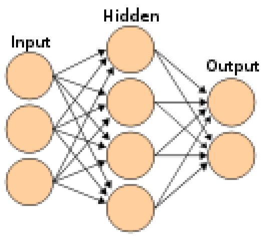

Figure 1: An ANN with three layers.

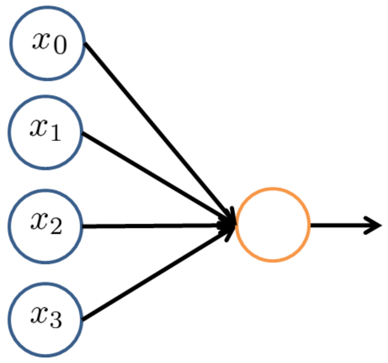

Figure 2: Neuron with three inputs and a bias term

As the data put into the input layer is translated into *activity* levels in the cells. These activities then propagate via the axons through the network, causing different activity levels in each layer. The is repeated, until reaching the conclusion: The activity levels in the output layer. These are then interpreted accordingly.

2 Mathematical formulation

2.1 Neuron activity

The activity of a neuron is affected by all dendrites pointing to the cell, i.e. all cells in the previous layer. To compute the activity, all the inputs are weighted corresponding to the strength of each dendrite. In addition a *bias* term may be added - a constant term.

The result is then put into an *activation function* to yield the activity of the neuron.

Mathematically, this may be formulated:

$$a_{\text{neuron}} = f\left(\sum_{i=1}^{n} \theta_i x_i + x_0\right) \tag{2.1}$$

Here, a is the activation, the x_i are the inputs (n total), the θ_i the corresponding weights (strength of the dendrites), x_0 is the bias term and finally f is the activation function.

The situation is outlined in figure 2. Note that the bias has been shown as an input, even if this is not really the case: It is merely a constant term.

The inputs and weights are often written in vector form:

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}, \quad \theta = \begin{pmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{pmatrix} \tag{2.2}$$

Hence, equation 2.1 may be written:

$$a_{\text{neuron}} = f\left(\theta^T x + x_0\right) \tag{2.3}$$

2.2 Activation function

For our activation function, we will choose the *logistic function*. This is also known as the *sigmoid function*[2]. It is defined as:

$$g(z) = \frac{1}{1 + e^{-z}} \tag{2.4}$$

The function is graphed in figure 3. It can be though of as a "smoothed out step function"[3].

The reason for the choice of this function is two-fold. First of all, it introduces *activation saturation* into the model: Even if $\theta^T x + x_0$ turns out to be very large, the corresponding activity will still be close to 1. Similarly large negative values produce activities close to 0.

However, many functions could have achieved activation saturation. But $g(z)$ also has an additional nice mathematical property. Consider its derivative:

$$g'(z) = \frac{1' \cdot (1 + e^{-z}) - 1 \cdot (1 + e^{-z})'}{(1 + e^{-z})^2} = \frac{e^{-z}}{(1 + e^{-z})^2} = \frac{e^{-z}}{1 + e^{-z}} g(z) \tag{2.5}$$

It turns out the first factor is equal to $1 - g(z)$:

$$1 - g(z) = 1 - \frac{1}{1 + e^{-z}} = \frac{1 + e^{-z}}{1 + e^{-z}} - \frac{1}{1 + e^{-z}} = \frac{1 + e^{-z} - 1}{1 + e^{-z}} = \frac{e^{-z}}{1 + e^{-z}} \tag{2.6}$$

So we have:

$$g'(z) = g(z)\left(1 - g(z)\right) \tag{2.7}$$

[2]Though technically, any function with an s-shaped graph could be called a sigmoid function.

[3]A neuron with an actual step function as an activation function is known as a *perceptron*.

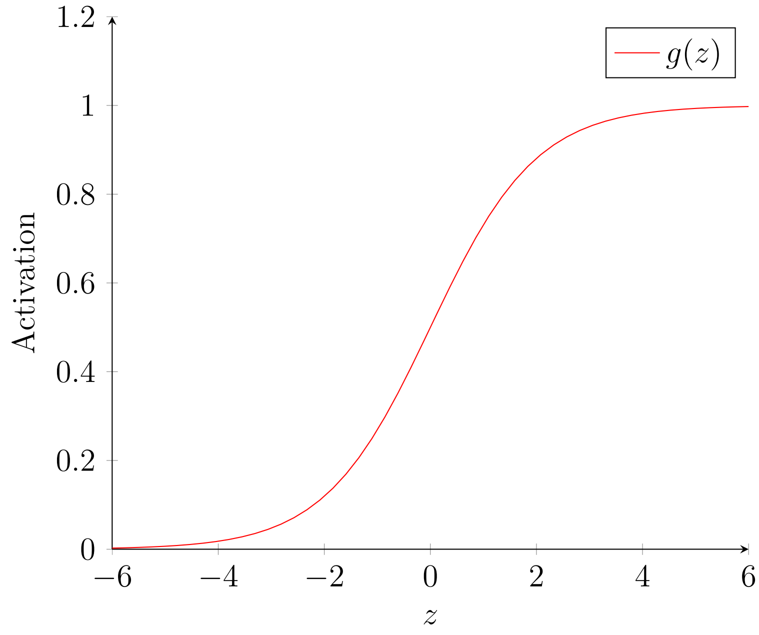

Figure 3: The sigmoid function.

2.3 Matrix form of layer activities

So, given cell number i in layer L, its activation depends on the input activities from layer $L-1$, the activity is:

$$a_i^{(L)} = g\left(\left(\theta_i^{(L)}\right)^T a^{(L-1)} + b_i^{(L)}\right) \tag{2.8}$$

Here, the activities from layer $L-1$ has been collected into the vector $a^{(L-1)}$ and the biases into the vector b^L. However, this may simply be expressed through matrix multiplication:

$$a^{(L)} = g\left(\theta^{(L)} a^{(L-1)} + b^{(L)}\right) \tag{2.9}$$

Here, the matrix $\theta^{(L)}$ has rows consisting of weights for each cell. Also, g is to be applied component-wise.

2.3.1 Dimensionalities

If layer $L-1$ contains m_{L-1} cells and layer L contains m_L cells, this means the dimensionality of the matrices and vectors above are:

$$a^{(L-1)} \in \mathbb{R}^{m_{L-1}\times 1}, \quad a^{(L)}, b^{(L)} \in \mathbb{R}^{m_L \times 1}, \quad \theta^{(L)} \in \mathbb{R}^{m_L \times m_{L-1}} \tag{2.10}$$

2.4 Output layer as a function of input layer

If there is a total of K layers in the ANN. Then the input layer is $x \equiv a^{(1)}$ and the output layer $y \equiv a^{(K)}$. Then we may express y as a function of x through repeated application of equation 2.8. This process is known as *forward propagation*:

$$y = g\left(\theta^{(K)}\left(\cdots g\left(\theta^{(3)} g\left(\theta^{(2)} x + b^{(2)}\right) + b^{(3)}\right)\cdots\right) + b^{(K)}\right) \equiv h_{\theta,b}(x) \quad (2.11)$$

Here the notation $h_{\theta,b}(x)$ been introduced. Here h stands for *hypothesis*, i.e. the hypothesis that the particular choices of θ and b are appropriate.

3 ANN used for classification

As an example, consider an ANN trained for picture classification. Se we imagine an ANN which has a representation of a picture for inputs x. The output layer is a series of n numbers between 0 and 1. We may interpret these as probabilities. Maybe an ANN has been trained to recognize cats, dogs and horses. In this case, the size of the output layer is 3, and y_1, y_2 and y_3 are interpreted as the probabilities of the input picture containing a cat, dog or a horse respectively. If we know the picture only contains one animal, we would simply pick the highest value and use the corresponding animal as prediction.

4 Training an ANN

So the question is now how to train an ANN for classification? This is a supervised machine learning problem, as we imagine we have a large dataset to train the network on.

The actual training is done using *maximum likelihood estimation* of the weights and biases. In practice, this will be done using gradient or a similar optimization algorithm. The partial derivatives needed for this is found using what is known as the *backpropagation algorithm.*

4.1 Likelihood of a Bernoulli process

Let's take a step back and consider a Bernoulli process, i.e. an experiment with two outcomes: success and failure. The probability of success is called

p, and hence the probability of failure must be $1 - p$. This may be summed up in the equation:

$$P(x|p) = p^x (1-p)^{1-x}, \quad x = 0, 1 \tag{4.1}$$

Here $x = 1$ indicates success and $x = 0$ failure.

The "probability mindset" here is, that if we know p we know the distribution. The idea of likelihood swaps this around: Instead we ask what the parameter p might be given the outcome x:

$$\mathcal{L}(p|x) = p^x (1-p)^{1-x} \tag{4.2}$$

Based on the observation, we will then pick the value of p that makes the observation most likely. This is known as maximum likelihood estimation[4].

Often, it is more convenient to minimize the *log-likelihood*, i.e. minus the logarithm of the likelihood[5]. Since log is monotone, this is equivalent. For the Bernoulli process:

$$L(p|x) = -\log \mathcal{L}(p|x) = -x \log p - (1-x) \log(1-p) \tag{4.3}$$

This can also be seen as a *cost function*, i.e. something we wish to minimize. Figure 4 graphs the function for the two values of x.

4.2 Single item training set

Now consider a single item training set, i.e. an input x that has been classified at option a. This means that ideally, the output layer should have y_a equal to 1 and all other y_i equal to 0.

[4]For a single observation this means the estimate is $p = 1$ in case of success and $p = 0$ in case of failure. In other words $p = x$.

[5]Not all presentations include the minus sign. It is kept here to make L a cost function.

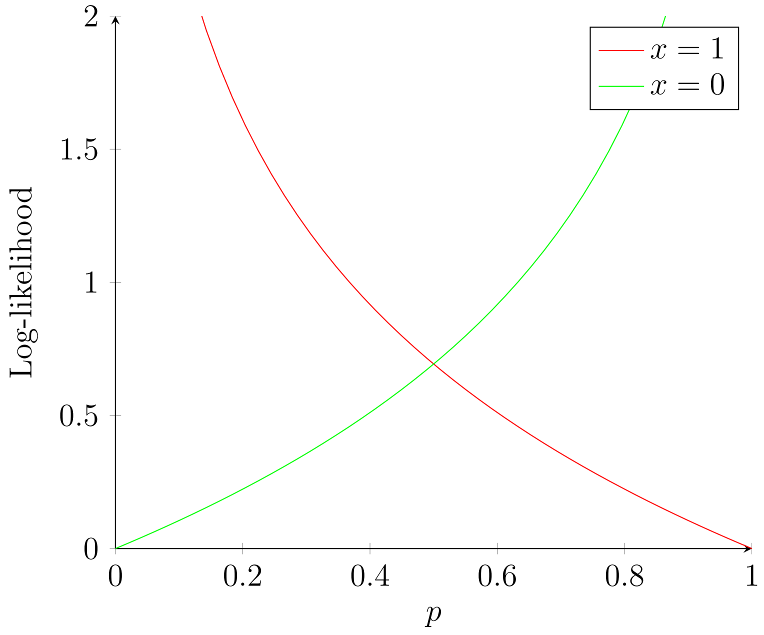

Figure 4: Log-likelihoods for the Bernoulli process.