Artificial Neural Networks

Kristian Wichmann

November 13, 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.

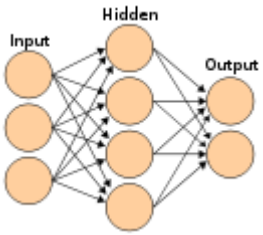

Figure 1: An ANN with three layers.

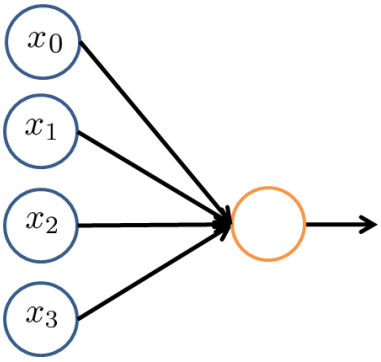

Figure 2: Neuron with three inputs and a bias term

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*.

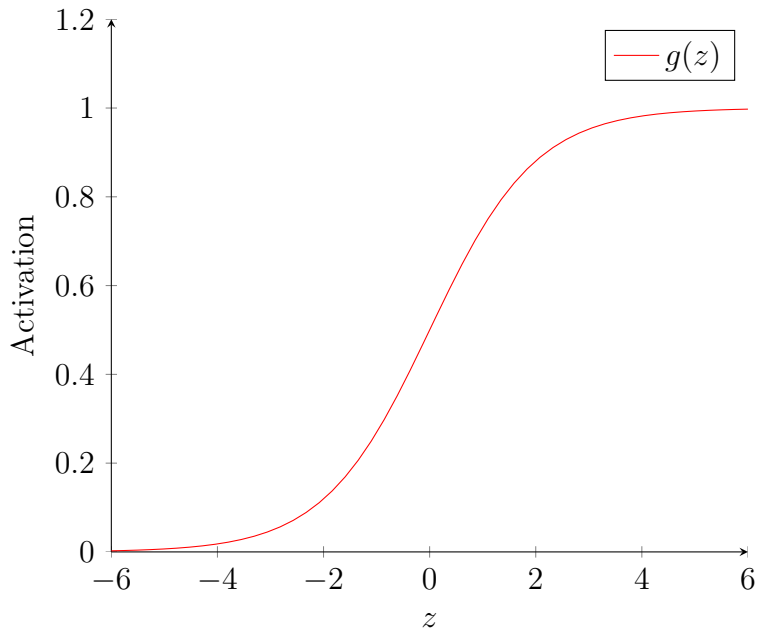

Figure 3: The sigmoid function.

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*:

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

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.

3.1 Training an ANN - overview

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*.

All of these topics will be the subject of the next set of sections.

4 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.

5 Single item training set

Now consider a single item training set, i.e. an input x that has been classified as option a. This corresponds to a Bernoulli process for each cell in the output y, with only the a'th one being a success. So the desired output is $y = \delta_a$.

Now, the log-likelihood of the i'th such Bernoulli process depends on the actual choice of weights and biases - more specifically, it depends on the $h^{(\theta,b)}$ function defined in equation 2.11:

$$L_i(\theta, b|y_i) = -y_i \log \left[h_i^{(\theta,b)}\right] - (1-y_i)\log\left[1 - h_i^{(\theta,b)}\right] \tag{5.1}$$

If the processes are assumed to be independent (a big if, but let's go with that for now), the log-likelihoods add up, and the total likelihood is:

$$L(\theta, b|y) = \sum_i \left\{-y_i \log \left[h_i^{(\theta,b)}\right] - (1-y_i)\log\left[1 - h_i^{(\theta,b)}\right]\right\} \tag{5.2}$$

[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.

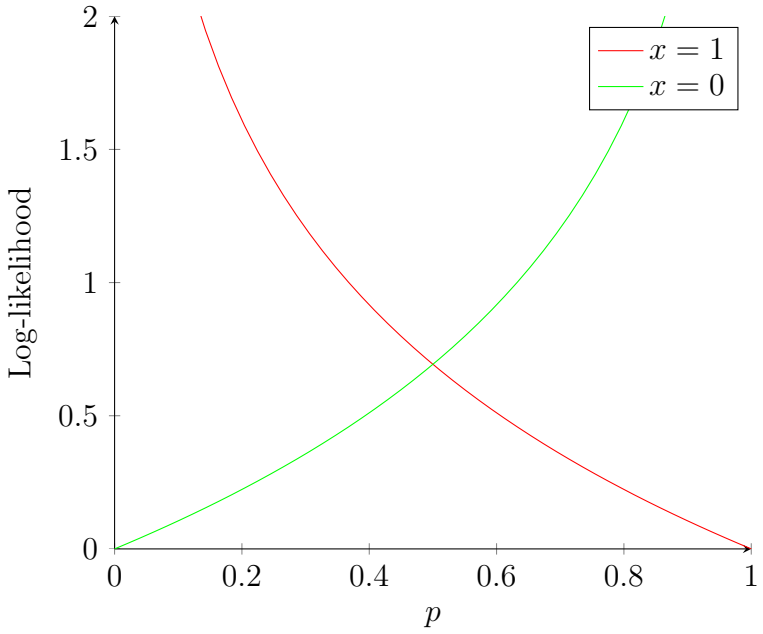

Figure 4: Log-likelihoods for the Bernoulli process.

The task is now to find the weights and biases that minimize this. To do so, we wish to use gradient descent, and so need the partial derivatives of L with respect to all weights and biases!

If α is any weight or bias, we can differentiate L with respect to it:

$$\frac{\partial L}{\partial \alpha} = \sum_i \left\{ -y_i \frac{1}{h_i^{(\theta,b)}} \frac{\partial h_i^{(\theta,b)}}{\partial \alpha} - (1 - y_i) \frac{1}{1 - h_i^{(\theta,b)}} \left(-\frac{\partial h_i^{(\theta,b)}}{\partial \alpha} \right) \right\} \tag{5.3}$$

Collecting terms:

$$\frac{\partial L}{\partial \alpha} = \sum_i \left\{ \frac{-y_i \left(1 - h_i^{(\theta,b)}\right) + (1 - y_1) h_i^{(\theta,b)}}{h_i^{(\theta,b)} (1 - h_i^{(\theta,b)})} \frac{\partial h_i^{(\theta,b)}}{\partial \alpha} \right\} \tag{5.4}$$

$$\sum_i \left\{ \frac{h_i^{(\theta,b)} - y_i}{h_i^{(\theta,b)} \left(1 - h_i^{(\theta,b)}\right)} \frac{\partial h_i^{(\theta,b)}}{\partial \alpha} \right\} \tag{5.5}$$

So the problem is reduced to finding partial derivatives of h.

5.1 Partial derivatives: Output layer

The values of the output layer can be expressed through their dependence on the last hidden layer as per equation 2.8:

$$h^{(\theta,b)} = a^{(K)} = g(\theta^{(K)} a^{(K-1)} + b^{(K)}) \tag{5.6}$$

Remember that this is a vector equation - we need to consider each component. The property of the sigmoid function expressed in equation 2.7 now comes in handy when we want to calculate partial derivatives:

$$\frac{\partial h_i^{(\theta,b)}}{\partial \alpha} = g(z_i)(1 - g(z_i))\frac{\partial z_i}{\partial \alpha}, \quad z_i = \sum_j \theta_{ij}^{(K)} a_j^{(K-1)} + b_i^{(K)} \tag{5.7}$$

But $g(z_i) = h_i^{(\theta,b)}$! So inserting into equation 5.5 there's a lot of cancellation:

$$\frac{\partial L}{\partial \alpha} = \sum_i \left\{ \left(h_i^{(\theta,b)} - y_i \right) \frac{\partial z_i}{\partial \alpha} \right\} = \sum_i \delta_i^{(K)} \frac{\partial z_i}{\partial \alpha} \tag{5.8}$$

Here, the *error* $\delta^{(K)} = h_i^{(\theta,b)} - y_i$ for the output layer has been introduced. This can be seen as the amount the hypothesis specified by the current weights and biases is off compared to what is desired. Be careful to distinguish between the error and Kronecker deltas in the following!

Equation 5.8 may be written in vector form:

$$\frac{\partial L}{\partial \alpha} = \left(\delta^{(K)}\right)^T \frac{\partial z}{\partial \alpha} \tag{5.9}$$

Finally, we need to find the partial derivatives of z. Again, remember that z is a vector:

$$z_i = \sum_j \theta_{ij}^{(K)} a_j^{(K-1)} + b_i^{(K)} \tag{5.10}$$

At this point we only consider weights and biases from the axons pointing at the output layer, i.e. with superscript K. But since the weights form a matrix and the biases a vector, the resulting derivative objects will be 3 and 2-dimensional respectively:

$$\frac{\partial z_i}{\partial \theta_{kl}^{(K)}} = \sum_j \delta_{ik}\delta_{jl} a_j^{(K-1)} = \delta_{ik} a_l^{(K-1)}, \quad \frac{\partial z_i}{\partial b_k^{(K)}} = \delta_{ik} \tag{5.11}$$

This means that the only non-zero derivatives are:

$$\frac{\partial z_i}{\partial \theta_{ij}^{(K)}} = a_j^{(K-1)}, \quad \frac{\partial z_i}{\partial b_i^{(K)}} = 1 \tag{5.12}$$

We may now calculate the derivatives of L:

$$\frac{\partial L}{\partial \theta_{ij}^{(K)}} = \sum_k \delta_k^{(K)} \delta_{ki} a_j^{(K-1)} = \delta_i^{(K)} a_j^{(K-1)} \tag{5.13}$$

$$\frac{\partial L}{\partial b_i^{(K)}} = \sum_k \delta_k^{(K)} \delta_{ki} = \delta_i^{(K)} \tag{5.14}$$

5.2 Partial derivatives: Hidden layers

On to the hidden layers. Luckily, it turns out that most of the calculations echo the output layer one closely.

Let's look at a weight or bias α from the last hidden layer. We need the derivative of h with respect to α. It is convenient to make the following definition:

$$z^{(N)} = \theta^{(N)} a^{(N-1)} + b^{(N)} \tag{5.15}$$

Then we may write:

$$h^{(\theta,b)} = g(z^{(K)}) \tag{5.16}$$

Now we wish to form the partial derivative with respect to a weight/bias α from the last hidden layer, i.e. layer $K-1$. Here in vector form:

$$\frac{\partial h^{(\theta,b)}}{\partial \alpha} = g(z^{(K)}) \odot (1 - g(z^{(K)}) \odot \frac{\partial z^{(K)}}{\partial \alpha} \tag{5.17}$$

Here, the symbol $\odot$ is used to mean the *Hadamard product* or *Schur product*. It simply takes two vectors of equal length and return another vector of the same length, where each component is the product of the relevant factor components:

$$\begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} \odot \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} = \begin{pmatrix} a_1 b_1 \\ a_2 b_2 \\ \vdots \\ a_n b_n \end{pmatrix} \tag{5.18}$$

So what is the derivative? It is almost the same calculation as above:

$$\frac{\partial z^{(K)}}{\partial \alpha} = \frac{\partial}{\partial \alpha} g(z^{(K-1)}) = g(z^{(K-1)})(1 - g(z^{(K-1)})) \frac{\partial z^{(K-1)}}{\partial \alpha} \tag{5.19}$$

Again, the two first factors can be expressed as activiations, and because α is associated with layer $K-1$, the derivative is analogous to the results from equations 5.13 and 5.14. Inserting into equation 5.5, once again, there's some cancelling out, but this time we get some extra factors:

$$\frac{\partial L}{\partial \alpha} = \sum_i \left\{ \delta_i^{(K)} a_i^{(K-1)} \left(1 - a_i^{(K-1)}\right) \frac{\partial z_i^{(K-1)}}{\partial \alpha} \right\} \tag{5.20}$$

Now, set the error of layer $K-1$ equal to:

$$\delta^{(K-1)} = \delta^{(K)} \odot a_i^{(K-1)} \odot \left(1 - a^{(K-1)}\right) \tag{5.21}$$

The calculations are now essentially the same as in the last section, and we get:

$$\frac{\partial L}{\partial \theta_{ij}^{(K-1)}} = \delta_i^{(K-1)} a_j^{(K-2)} \tag{5.22}$$

$$\frac{\partial L}{\partial b_i^{(K-1)}} = \delta_i^{(K-1)} \tag{5.23}$$

If α had been associated with an earlier hidden layer, we could have followed the same steps iteratively, defining the error of layer N recursively:

$$\delta^{(N)} = \delta^{(N+1)} \odot a_i^{(N)} \odot \left(1 - a^{(N)}\right), \quad N \geq 2 \tag{5.24}$$

The general formulas for the derivatives is then:

$$\frac{\partial L}{\partial \theta_{ij}^{(N)}} = \delta_i^{(N)} a_j^{(N-1)} \tag{5.25}$$

$$\frac{\partial L}{\partial b_i^{(N)}} = \delta_i^{(N)} \tag{5.26}$$

Again, N should be at least 2, reflecting the fact that there's no error, weights or biases for the input layer.

5.3 Backpropagation algorithm

As we have seen above, calculation of derivatives is a recursive process, starting from the output layer and working backwars. Hence the name *backpropagation algorithm*. Here are the steps in bullet form:

- First calculate all the activations of the neurons corresponding to the input x and the current weights and biases.
- Using the y's, calculate the error of the output layer.
- Repeat the following until all derivatives are calculated:
  - Calculate derivatives for the current layer using the layer's error and the previousy calculated activations.
  - Iteratively calculate the error of the previous layer.

This is merely one step in the gradient descent process. The derivatives are used to generate a new set of weights/biases and the process is repeated until all derivatives are sufficiently small.

6 Full training set

So far, we've been dealing with the situation for a training set consisting of only one item. In practice, we ideally want to train our ANN on a large dataset.

However, since each item in the training set is independent of each other, the log-likelihood of multiple items is simply equal to the sum of all of them. By additivity of the derivative, the backpropagation process essentially remains the same, except we need to sum over the entire training set in each step of the gradient descent.