Training a Neural Network - A Numerical Example

Raphael B. Alampay

December 3, 2017

Abstract

Neural networks are models used to approximate a discriminative function for classification in a supervised learning fashion. You have a bunch of input in the form of n-dimensional numerical vectors that represent features of certain entities you'd like to teach the network to classify (example 1000 emails classified as spam and 1000 emails classified non-spam). This paper takes a look at a quantitative approach in training a neural network given actual numerical examples. This will allow the student/practitioner to understand the fundamentals of the training process namely feed forward and backpropagation. The paper is intended to be light in concept with specific examples for people getting into machine learning with neural networks.

1 Introduction

Neural networks are biologically inspired computational models that attempts to solve classification problems. It is composed on **neurons** which holds and processes values in the network. You can think of these values as signal strengths that aim to mimic how chemical reactions occur in the brain. The higher the value, the stronger the signal. In biology, neurons transmit and receive signals to and from other neurons by means of dendrites. These propagations are modelled in the neural network by means of weight values. Since a neuron may receive values from more than one neuron, it accounts all the weights connected to it before attempting to fire a signal thus simulating how we "react" to certain stimuli. Training the neural network roughly means looking for the optimal values for these weights based on what we already know in order for the model to properly "react" to a certain input.

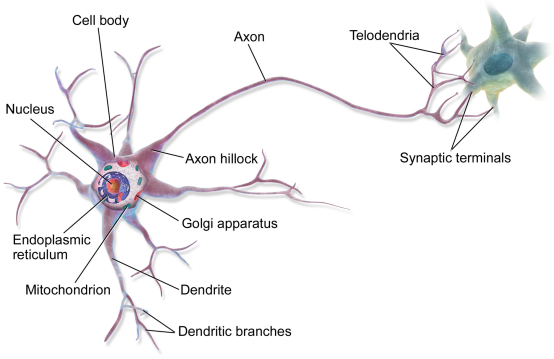

Figure 1: An example of a neuron taken from wikipedia.

There are typically 3 layers in a classical neural network - **input layer**, **hidden layer/s** and **output layer**. These layers are connected to one another via neurons' weights in an adjacent manner. This means that a layer can only be connected to at most two adjacent layers. The **input layer** will always be the leftmost layer, the **output layer** will always be the rightmost layer while everything in between will be **hidden layers**. The following example is an illustration of the classical neural network:

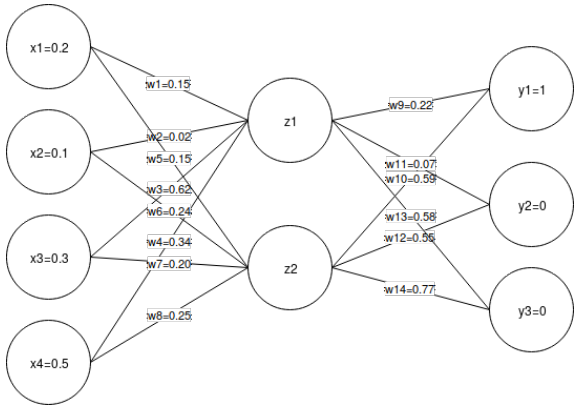

Figure 2: A neural network with 4 neurons in the input layer, 2 for hidden and 3 for output

1.1 Input Layer

The input layer contains neurons that represent the input values or what the network initially receives from the real world. These values are features that represent a classification/label that we'd like to recognize. In the mathematical model $f(x) = y$, this would be the x as an n-dimensional vector. For example, if we'd like to tell the neural network that we're looking at an image of a face represented by a matrix of pixel values, and suppose the size of the image is 32 x 32 pixels, the input layer would have a total of 1024 neurons each one corresponding to a pixel value. We refer to this vector as a **feature vector**.

1.2 Hidden Layer/s

From the input layer, information is passed to a hidden layer which contains neurons that processes signals it receives from the its adjacent layer/s (either from the left or from the right). A neural network can have more than one hidden layer. Neurons in the hidden layer are often denoted as z_i which we refer to as **latent variables**.

1.3 Output Layer

The output layer contains neurons that represent the output of the network or the result/reaction of the network after receiving and processing the input (from input layer to hidden layer then finally to the output layer). The easiest way to model neurons in the output layer is to treat each one (neuron) as a classification/label that the network is trying to recognize with a value ranging from 0 to 1. The closer the value to 1 for an output neuron, the closer it is to thinking that it is that classification or label for a given input x. For example, let's say we're trying to learn how to diffirentiate cats from dogs from any other animal. The set of possible outputs (cat, dog or others) can be represented by:

$$Y = \begin{bmatrix} y_1 & y_2 & y_3 \end{bmatrix} \tag{1}$$

where y_1 represents the label cat, y_2 dog and y_3 others. A cat then for a neural network would look like $f(x) = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$, a dog would be $f(x) = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}$ and finally any other animal $f(x) = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$.

2 Training

To train a neural network means to optimize the set of weights (connections between neurons) in such a way that when we give it a feature vector representing a cat, the output should be close to $f(x) = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$. In order to do this, at the beginning of training, the weights are randomly initialized. We then feed it a feature vector relating to cat and see what the Y value is — how close did the network get to the actual answer. This closeness value can be measured by a **loss function**. A higher value for this function means that the network yielded a higher error while smaller value means that the network is getting pretty close to recognizing what the input is. We

will use this error/closeness value to adjust the weights accordingly. The first step in this entire training process is called **feed forward**.

For the rest of the discussion, we will be referring to figure 2 as reference. Initially, the weights of the network will be as follows:

$$W_1 = \begin{bmatrix} 0.15 & 0.14 \\ 0.02 & 0.24 \\ 0.62 & 0.20 \\ 0.34 & 0.25 \end{bmatrix} \tag{2}$$

$$W_2 = \begin{bmatrix} 0.22 & 0.07 & 0.58 \\ 0.59 & 0.55 & 0.77 \end{bmatrix} \tag{3}$$

2.1 Feed Forward

The feed forward process can be thought of the movement of information from the input layer in the form of the feature vector's values to the first hidden layer and finally making the guess (classification) in the output layer. Given two consecutive layers, information is passed from the left layer to the right layer by performing a matrix multiplication operation between the left layer's neurons and the weight matrix between the left and right layer. The resulting matrix should have the same size as the neurons in the right layer (1 row with n columns where n is the size of the neurons in the right layer). Let's take the following illustration:

We're trying to solve for the values of Z by matrix multiplying X with W_1. The result will then be passed to an activation function such as the **sigmoid function** given by:

$$a(x) = 1/(1 + e^{-x}) \tag{4}$$

Mathematically, we can then represent information flow from X to Z:

$$Z = a((X)(W_1)) \tag{5}$$

Numerically we have the following:

$$Z = \begin{bmatrix} 0.2 & 0.1 & 0.3 & 0.5 \end{bmatrix} \begin{bmatrix} 0.15 & 0.14 \\ 0.02 & 0.24 \\ 0.62 & 0.20 \\ 0.34 & 0.25 \end{bmatrix} \tag{6}$$

$$Z = \begin{bmatrix} 0.388 & 0.237 \end{bmatrix} \tag{7}$$

$$Z = a(\begin{bmatrix} 0.388 & 0.237 \end{bmatrix}) \tag{8}$$

$$Z = \begin{bmatrix} 0.5958 & 0.5590 \end{bmatrix} \tag{9}$$

Using the same approach, we feed forward Z's values towards Y by matrix multiplying it with W_2.

Mathematically:

$$Y = a((Z_1)(W_2)) \tag{10}$$

Numerically:

$$Y = \begin{bmatrix} 0.5958 & 0.5590 \end{bmatrix} \begin{bmatrix} 0.22 & 0.07 & 0.58 \\ 0.59 & 0.55 & 0.77 \end{bmatrix} \tag{11}$$

$$Y = \begin{bmatrix} 0.4609 & 0.3491 & 0.7760 \end{bmatrix} \tag{12}$$

$$Y = a(\begin{bmatrix} 0.4609 & 0.3491 & 0.7760 \end{bmatrix}) \tag{13}$$

$$Y = \begin{bmatrix} 0.6132 & 0.5864 & 0.6848 \end{bmatrix} \tag{14}$$

2.2 Loss Function

The loss function determines how far (or close) the guess of the network (Y) to the actual classification value ($\widehat{Y}$). Remember that we want to teach the network how to recognize a certain classification by adjusting the its weights. The amount of adjustment we do will largely depend on the value given by the **loss function**. For this paper, we will be using a very simple loss function:

$$E(Y, \widehat{Y}) = \sum_{i}^{n} (Y_i - \widehat{Y}_i)^2 \tag{15}$$

In the case of our example, $\widehat{Y} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$, the value of our loss function will be computed as:

$$E(Y, \widehat{Y}) = \sum_{i}^{n} (\begin{bmatrix} 0.6132 & 0.5864 & 0.6848 \end{bmatrix} - \begin{bmatrix} 1 & 0 & 0 \end{bmatrix})^2 \tag{16}$$

Decomposing it we have:

$$e_1 = (0.6132 - 1)^2 \tag{17}$$

$$e_2 = (0.5864 - 0)^2 \tag{18}$$

$$e_3 = (0.6848 - 0)^2 \tag{19}$$

$$E(Y, \widehat{Y}) = e_1 + e_2 + e_3 \tag{20}$$

$$E(Y, \widehat{Y}) = 0.1496 + 0.3439 + 0.4690 \tag{21}$$

$$E(Y, \widehat{Y}) = 0.9624 \tag{22}$$

2.3 Back Propagation

Once we get the error value from the loss function, we can use this value to determine how much change we would apply to the weights to minimize this error value. This is done by a process called back propagation which takes the individual error values and cascades it from the output layer to the input layer seeing how much damage the weight values contributed to the overall error. Integral to this process is to solve for **gradients**. These gradients are approximation of derivatives. As with derivatives, gradient values dictate the direction of the error function of the network. If we know these values, we can roughly determine how to adjust our weights to minimize the error. To simplify things, since weight values drive the error value, we'd like to determine the gradient values for each neuron (since weights are attached to neurons) starting from the output layer (thus backward propagation) to give us the **delta weights** or how much magnitude should we adjust the original weights to lower the error. Numerically, this means that the size of our gradient vector will be the same size of neurons in a given layer.

For a more specific example, we'll break down this process into two major operations. The first part will perform back propagation starting from the output to the last hidden layer. And the second part will be from the last hidden layer down to the input layer. Similar to feed forward, we will be computing values with 3 inputs – 2 layers and the weight matrix in between them. For each pair of layers, the gradient values to be computed for the right layer.

2.3.1 BP from Output to Last Hidden Layer

To start off, we compute the gradients for the right layer in this part of BP which in this case is the output layer (Y).

2.3.2 BP from Last Hidden to Input Layer

3 Some examples to get started

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}^T = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

3.1 How to add Comments

Comments can be added to your project by clicking on the comment icon in the toolbar above. To reply to a comment, simply click the reply button in the lower right corner of the comment, and you can close them when you're done.

Comments can also be added to the margins of the compiled PDF using the todo command, as shown in the example on the right. You can also add inline comments:

This is an inline comment.

Here's a comment in the margin!

3.2 How to write Mathematics

LaTeX is great at typesetting mathematics. Let $X_1, X_2, \ldots, X_n$ be a sequence of independent and identically distributed random variables with $\mathrm{E}[X_i] = \mu$ and $\mathrm{Var}[X_i] = \sigma^2 < \infty$, and let

$$S_n = \frac{X_1 + X_2 + \cdots + X_n}{n} = \frac{1}{n}\sum_{i}^{n} X_i$$

denote their mean. Then as n approaches infinity, the random variables $\sqrt{n}(S_n - \mu)$ converge in distribution to a normal $\mathcal{N}(0, \sigma^2)$.

3.3 How to create Sections and Subsections

Use section and subsections to organize your document. Simply use the section and subsection buttons in the toolbar to create them, and we'll handle all the formatting and numbering automatically.

3.4 How to add Lists

You can make lists with automatic numbering ...

1. Like this,
2. and like this.

... or bullet points ...

- Like this,
- and like this.

3.5 How to add Citations and a References List

You can upload a `.bib` file containing your BibTeX entries, created with JabRef; or import your Mendeley, CiteULike or Zotero library as a `.bib` file. You can then cite entries from it, like this: [?]. Just remember to specify a bibliography style, as well as the filename of the `.bib`.

You can find a video tutorial here to learn more about BibTeX.

We hope you find Overleaf useful, and please let us know if you have any feedback using the help menu above — or use the contact form at `https://www.overleaf.com/contact`!