ML/DL fundamentals

Neural nets: part 1

0.1 Neural Nets

On a very fundamental level, a neural network is nothing a network of small computing units, each of which takes a vector of inputs values and produces an output value. First we will look at a **feedforward network** since in this case the computation proceeds iteratively from one layer to the next. Also, we would be looking at neural nets from a classification point of view, especially in connection with the logistic regression concepts we developed earlier.

0.1.1 Unit

A single computation unit is the most basic building block of a neural net. It takes as input, a vector, does some computation on it and produces an output. Basically, it takes in the weighted sum of the components of the input vector, with a **bias term** also added. With a set of input values $x_1, \cdots, x_n$, its associated set of weights $w_1, \cdots, w_n$ and the bias term b we write the weighted sum z as follows:

$$z = b + \sum_{i}^{n} w_i x_i \tag{1}$$

We can write this conveniently in vector notation as follows:

$$z = \boldsymbol{w} \cdot \boldsymbol{x} + b \tag{2}$$

Now we note that instead of simply using this computed z as the output, the neural unit applies a non-linear function f to z. This is called the **activation function** and the value output is called the **activation value** for the unit, a. So for a single unit, the value y would be given as:

$$y = a = f(z) \tag{3}$$

Possibly the most popular activation function would be the **sigmoid function** given by:

$$y = \sigma(z) = \frac{1}{1 + e^{-z}} \tag{4}$$

Recall that the sigmoid is extremely handy since it maps our z output to the $(0, 1)$ range essentially. Finally, the final output of a single neural unit would be:

$$y = \sigma(\boldsymbol{w} \cdot \boldsymbol{x} + b) = \frac{1}{1 + e^{-(\boldsymbol{w} \cdot \boldsymbol{x} + b)}} \tag{5}$$

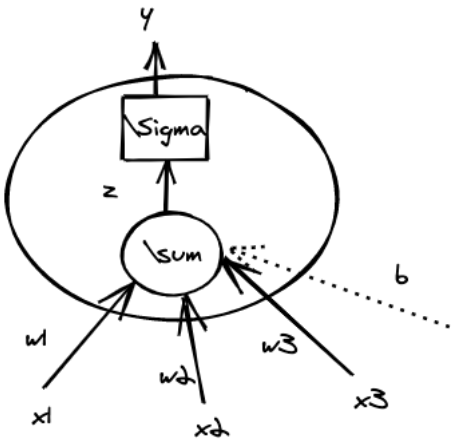

Figure 1: single unit neural net: takes in wieghted values of x and a bias term, computes weighted sum z and outputs the sigmoid output y.

Now we note another variant of the activation function that is often used instead of the sigmoid is the **tanh** function. Its output ranges from -1 to $+1$ and is given by:

$$y = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{6}$$

Yet another common function we use as an activation is the **rectified linear unit (ReLU)**. It is simply given by:

$$y = \max(x, 0) \tag{7}$$

0.2 Feed forward neural net

A feedforward network is a multilayer network in which the units are connected with no **cycles**. That is, outputs of units in each layer are passed on to units of the next layers and not to be previous layer. These are also known as **Multilayer Perceptrons (MLP)**. On a very basic level, they have three kinds of nodes - input units, hidden units and output units. In the figure below, we have a 2 layer feed-forward network. Note that the dashed lines indicate the **bias weight** b attached to a node of default value 1 - it would give different bias wieghts for each layer basically.

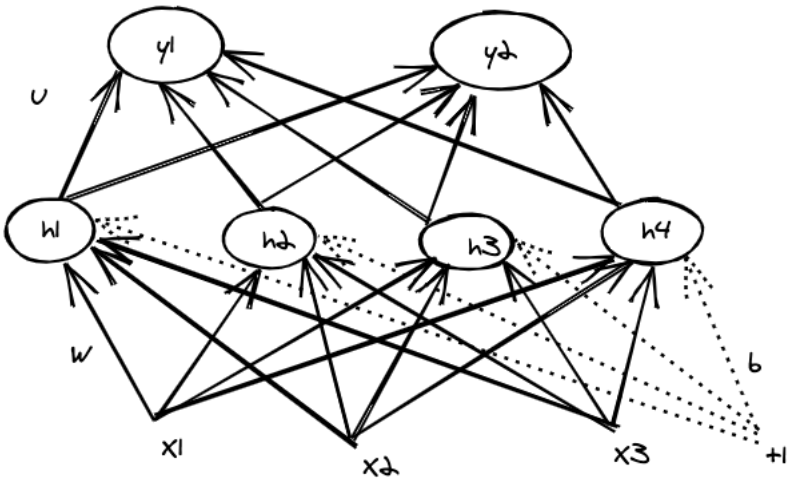

Figure 2: feed forward neural net

The main action in this framework happens in the **hidden layer** where the hidden units take a weighted sum of inputs and then apply a non-linear function on it. We further note that each layers is **fully connected** - which essentially means that each unit in one layer receives inputs from all units of the previous layer. Each hidden unit get as input the parameters for weights w and the bias b. Now we can basically represent the weights of all the hidden units together, that is the weights of the **entire hidden layer** in the form of the weight matrix W and a similarly, a single **bias vector** for the whole layer $\boldsymbol{b}$. **NOTE**: Each element of weight matrix W_{ij} represents the weight of the connection from input unit x_i to the hidden unit h_j. The advantage of putting everything into a matrix is that we can easily apply matrix computations so as to compute values for the entire hidden layer. Now the output of the hidden layer using sigmoid would be:

$$h = \sigma(W\boldsymbol{x} + \boldsymbol{b}) \tag{8}$$

Further we note that the output of this sigmoid transformation would be a vector rather than a single number, as was the case earlier.

0.2.1 Further details

Now we refer to the input layer as layer 0 and let n_0 be the number of inputs. Thefore $\boldsymbol{x}$ is a vector of n_0 dimension. The hidden layer is called 1 and it has a dimensionality of n_1 which in turn implies $h \in R^{n_1}$ and $b \in R^{n_1}$. Finally, the weight matrix has dimensionality $W \in R^{n_0 \times n_1}$. Note additionally that under the hood, the matrix multiplication of W and x will compute the value of each h_j as:

$$h_j = \sigma(\sum_{i=1}^{n_0} w_{ij}x_i + b_j) \tag{9}$$

We further note that each h_j actually forms a different representation of the input. Now finally, the role of the output layer is to take this new representation of the

input and compute an output - the output is typically a number that results in a classification decision. Now like the hidden layer, the output layer also has a weight matrix named U. Now very simply, the weight matrix is multiplied by the input vector h to get the indermediate output z as follows:

$$z = U\boldsymbol{h} \tag{10}$$

Each element U_{ij} is the weight from hidden unit j to the output unit i. Now note that this z is actually a vector, with each component representing the output of each hidden unit. So we want to now transform this vector of real values outputs to a **vector of probabilities**. We typically apply a **softmax** for this purpose. So for any output component z_i we have:

$$softmax(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{d} e^{z_j}} \tag{11}$$

So the final equations to compute the neural net are:

$$\boldsymbol{h} = \sigma(W\boldsymbol{x} + \boldsymbol{b}) \tag{12}$$

$$\boldsymbol{z} = U\boldsymbol{h} \tag{13}$$

$$\boldsymbol{y} = softmax(\boldsymbol{z}) \tag{14}$$

References

[1] Daniel Jurafsky, James H Martin - Speech and Language Processing