[Summary]NNs 1

Neural Networks: Representation
Non-linear Hypotheses

Performing linear regression with a complex set of data with many features is very unwieldy. Say you wanted to create a hypothesis from three (3) features that included all the quadratic terms:

$$g(\theta_0 + \theta_1 x_1^2 + \theta_2 x_1 x_2 + \theta_3 x_1 x_3 + \theta_4 x_2^2 + \theta_5 x_2 x_3 + \theta_6 x_3^2$$

That gives us 6 features.

In this case we are taking all two–element combinations of three features:

$$\frac{(3+2-1)!}{(2!\cdot(3-1)!)} = \frac{4!}{4} = 6$$

For 100 features, if we wanted to make them quadratic we would get

$\frac{(100+2-1)}{(2\cdot(100-1)!)} = 5050$ resulting new features.

We can approximate the growth of the number of new features we get with all quadratic terms with $O(\frac{n^2}{2})$. And if you wanted to include all cubic terms in your hypothesis, the features would grow asymptotically at $O(n^3)$. These are very steep growths, so as the number of our features increase, the number of quadratic or cubic features increase very rapidly and becomes quickly impractical.

Neural networks offers an alternate way to perform machine learning when we have complex hypotheses with many features.

Model Representation I

At a very simple level, neurons are basically computational units that take input (**dendrites**) as electrical input (called "spikes") that are channeled to outputs (**axons**).

In our model, our dendrites are like the input features $x_1 \cdots x_n$, and the output is the result of our hypothesis function:

In this model our x_0 input node is sometimes called the "bias unit." It is always equal to 1.

In neural networks, we use the same logistic function as in classification: $\frac{1}{1+e^{-\theta^T x}}$. In neural networks however we sometimes call it a sigmoid (logistic) **activation** function.

Our "theta" parameters are sometimes instead called "weights" in the neural networks model.

Visually, a simplistic representation looks like:

$$[x_0\ x_1\ x_2] \rightarrow [\quad] \rightarrow h_\theta(x)$$

Our input nodes (layer 1) go into another node (layer 2), and are output as the hypothesis function.

The first layer is called the "input layer" and the final layer the "output layer", which gives the final value computed on the hypothesis.

We can have intermediate layers of nodes between the input and output layers called the "hidden layer".

We label these intermediate or "hidden" layer nodes $a_0^2 \cdots a_n^2$ and call them "activation units".

$$\mathrm{a}_i^j = \text{activation} \mathit{of\ unit\ i\ in\ layer\ j}$$
$$\Theta^j = \mathit{matrix\ of\ weights\ controlling\ function\ mapping\ from\ layer\ j\ to\ layer\ j} + 1$$

If we had one hidden layer, it would look visually something like:

$$[x_0\ x_1\ x_2] \rightarrow [a_1^2\ a_2^2\ a_3^2] \rightarrow \mathrm{h}_\theta(x)$$

The values for each of the "activation" nodes is obtained as follows:

$$\begin{aligned} a_1^2 &= g(\Theta_{10}^1 x_0 + \Theta_{11}^1 x_1 + \Theta_{12}^1 x_2 + \Theta_{13}^1 \mathrm{x}_3) \\ a_2^2 &= g(\Theta_{20}^1 x_0 + \Theta_{21}^1 x_1 + \Theta_{22}^1 x_2 + \Theta_{23}^1 \mathrm{x}_3) \\ a_3^2 &= g(\Theta_{30}^1 x_0 + \Theta_{31}^1 x_1 + \Theta_{32}^1 x_2 + \Theta_{33}^1 \mathrm{x}_3) \\ \mathrm{h}_\Theta(x) = a_1^3 &= g(\Theta_{10}^2 a_0^2 + \Theta_{11}^2 a_1^2 + \Theta_{12}^2 a_2^2 + \Theta_{13}^2 a_3^2) \end{aligned}$$

This is saying that we compute our activation nodes by using a 3×4 matrix of parameters. We apply each row of the parameters to our inputs to obtain the value for one activation node. Our hypothesis output is the logistic function applied to the sum of the values of our activation nodes, which have been multiplied by yet another parameter matrix Θ^2 containing the weights for our second layer of nodes.

Each layer gets its own matrix of weights, Θ^j

The dimensions of these matrices of weights is determined as follows:

$$\textbf{\textit{If network has } } \boldsymbol{s_j} \textbf{\textit{ units in layer j and }} \boldsymbol{s_{j+1}} \textbf{\textit{ units in layer j}} \\ \boldsymbol{+1,} \textbf{\textit{ then }} \boldsymbol{\Theta^j} \textbf{\textit{ will be of dimension }} \boldsymbol{s_{j+1} \times (s_j + 1)}$$

The +1 comes from the addition in Θ^j of the "bias nodes," x_0 and Θ_0^j. In other words the output nodes will not include the bias nodes while the inputs will.

Model Representation II

In this section we'll do a vectorized implementation of the above functions. We're going to define a new variable z_k^j that encompasses the parameters inside our g function. In our previous example if we replaced the variable z for all the parameters we would get:

$$a_1^2 = g(z_1^2)$$

$$a_2^2 = g(z_2^2)$$
$$a_3^2 = g(z_3^2)$$

In other words, for layer j=2 and node k, the variable z will be:

$$z_k^2 = \Theta_{k,0}^1 x_0 + \Theta_{k,1}^1 x_1 + \cdots + \Theta_{k,n}^1 x_n$$

The vector representation of x and z^j is:

$$x = \begin{bmatrix} x_0 \\ x_1 \\ \cdots \\ x_n \end{bmatrix} z^j = \begin{bmatrix} z_1^j \\ z_2^j \\ \cdots \\ z_n^j \end{bmatrix}$$

Setting $x = a^1$, we can rewrite the equation as:

$$z^j = \Theta^{j-1} a^{j-1}$$

Now we can get a vector of our activiation nodes for layer j as follows:

$$a^j = g(z^j)$$

Where our function g can be applied element-wise to our vector z^j.

We can then add a bias unit (equal to 1) to layer j after we have computed a^j. This will be element a_0^j and will be equal to 1.

To compute our final hypothesis, let's first compute another z vector:

$$z^{j+1} = \Theta^j a^j$$

We get this final z vector by multiplying the next theta matrix after Θ^{j-1} with the values of all the activation nodes we just got.

This last theta matrix Θ^j will have only **one row** so that our result is a single number.

We then get our final result with:

$$h_\Theta(x) = a^{j+1} = g(z^{j+1})$$

Notice that in this **last step**, between layer j and layer j+1, we are doing **exactly the same thing** as we did in logistic regression.

Adding all these intermediate layers in neural networks allows us to more elegantly produce interesting and more complex non–linear hypotheses.

Multiclass Classification

We can define our set of resulting classes as y:

$$y^i = \begin{bmatrix}1\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\1\end{bmatrix},$$

Our final value of our hypothesis for a set of inputs will be one of the elements in y.