Week 4 Lecture Notes

ML: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. The exact way to calculate how many features for all polynomial terms is the combination function with repetition: http://www.mathsisfun.com/combinatorics/combinations-permutations.html $\frac{(n+r-1)!}{r!(n-1)!}$. In this case we are taking all two-element combinations of three features: $\frac{(3+2-1)!}{(2!\cdot(3-1)!)} = \frac{4!}{4} = 6$. (**Note**: you do not have to know these formulas, I just found it helpful for understanding).

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 $\mathcal{O}(n^2/2)$. And if you wanted to include all cubic terms in your hypothesis, the features would grow asymptotically at $\mathcal{O}(n^3)$. 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.

Example: let our training set be a collection of 50 x 50 pixel black-and-white photographs, and our goal will be to classify which ones are photos of cars. Our feature set size is then n = 2500 if we compare every pair of pixels.

Now let's say we need to make a quadratic hypothesis function. With quadratic features, our growth is $\mathcal{O}(n^2/2)$. So our total features will be about $2500^2/2 = 3125000$, which is very impractical.

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

Neurons and the Brain

Neural networks are limited imitations of how our own brains work. They've had a big recent resurgence because of advances in computer hardware.

There is evidence that the brain uses only one "learning algorithm" for all its different functions. Scientists have tried cutting (in an animal brain) the connection between the ears and the auditory cortex and rewiring the optical nerve with the auditory cortex to find that the auditory cortex literally learns to see.

This principle is called "neuroplasticity" and has many examples and experimental evidence.

Model Representation I

Let's examine how we will represent a hypothesis function using neural networks.

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 x0 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:

$$\begin{bmatrix}x_0\\x_1\\x_2\end{bmatrix} \rightarrow [\quad] \rightarrow h_\theta(x)$$

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."

$$a_i^{(j)} = \text{"activation" of unit } i \text{ in layer } j$$
$$\Theta^{(j)} = \text{matrix of weights controlling function mapping from layer } j \text{ to layer } j+1$$

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

$$\begin{bmatrix}x_0\\x_1\\x_2\\x_3\end{bmatrix} \rightarrow \begin{bmatrix}a_1^{(2)}\\a_2^{(2)}\\a_3^{(2)}\end{bmatrix} \rightarrow h_\theta(x)$$

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

$$\begin{aligned} a_1^{(2)} &= g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3) \\ a_2^{(2)} &= g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3) \\ a_3^{(2)} &= g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3) \\ h_\Theta(x) = a_1^{(3)} &= g(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)}) \end{aligned}$$

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 $\Theta^{(2)}$ containing the weights for our second layer of nodes.

Each layer gets its own matrix of weights, $\Theta^{(j)}$.

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

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

The +1 comes from the addition in $\Theta^{(j)}$ of the "bias nodes," x_0 and $\Theta_0^{(j)}$. In other words the output nodes will not include the bias nodes while the inputs will.

Example: layer 1 has 2 input nodes and layer 2 has 4 activation nodes. Dimension of $\Theta^{(1)}$ is going to be 4×3 where $s_j = 2$ and $s_{j+1} = 4$, so $s_{j+1} \times (s_j + 1) = 4 \times 3$.

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:

$$\begin{aligned} a_1^{(2)} &= g(z_1^{(2)}) \\ a_2^{(2)} &= g(z_2^{(2)}) \\ a_3^{(2)} &= g(z_3^{(2)}) \end{aligned}$$

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:

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

Setting $x = a^{(1)}$, we can rewrite the equation as:

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

We are multiplying our matrix $\Theta^{(j-1)}$ with dimensions $s_j \times (n+1)$ (where s_j is the number of our activation nodes) by our vector $a^{(j-1)}$ with height (n+1). This gives us our vector $z^{(j)}$ with height s_j.

Now we can get a vector of our activation 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 $\Theta^{(j-1)}$ with the values of all the activation nodes we just got.

This last theta matrix $\Theta^{(j)}$ 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.

Examples and Intuitions I

A simple example of applying neural networks is by predicting x_1 AND x_2, which is the logical 'and' operator and is only true if both x_1 and are 1.

The graph of our functions will look like:

Remember that is our bias variable and is always 1.

Let's set our first theta matrix as:

This will cause the output of our hypothesis to only be positive if both and are 1. In other words:

So we have constructed one of the fundamental operations in computers by using a small neural network rather than using an actual AND gate. Neural networks can also be used to simulate all the other logical gates.

Examples and Intuitions II

The matrices for AND, NOR, and OR are:

We can combine these to get the XNOR logical operator (which gives 1 if and are both 0 or both 1).

For the transition between the first and second layer, we'll use a matrix that combines the values for AND and NOR:

For the transition between the second and third layer, we'll use a matrix that uses the value for OR:

Let's write out the values for all our nodes:

And there we have the XNOR operator using two hidden layers!

Multiclass Classification

To classify data into multiple classes, we let our hypothesis function return a vector of values. Say we wanted to classify our data into one of four final resulting classes:

Our final layer of nodes, when multiplied by its theta matrix, will result in another vector, on which we will apply the g() logistic function to get a vector of hypothesis values.

Our resulting hypothesis for one set of inputs may look like:

In which case our resulting class is the third one down, or .

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.