Stanford Machine Learning - Week IV

Eric N Johnson

July 31, 2016

1 Neural Networks: Representation

If we have a nonlinear decision boundary sometimes we can use Logistic Regression - but will have a whole bunch of terms that are high-order polynomial. Second-order polynomial terms will grow at approximately $O(n^2)$ where n denotes the number of features. For cubic features, this is even worse at approximately $O(n^3)$.

Here is an example in *Computer Vision*. A digital car picture, for instance, is made up of a giant grid containing colors and darkness values. When a computer looks at a second image, perhaps one that is a different color or is turned a different directions, somehow the computer has to differentiate this image from something that is not a car. The processing required to do this grows in magnitude very quickly. A 50×50 image will end up with 3 million quadratic features that have to be used for classification.

1.1 Neurons in the Brain

Here's some background.

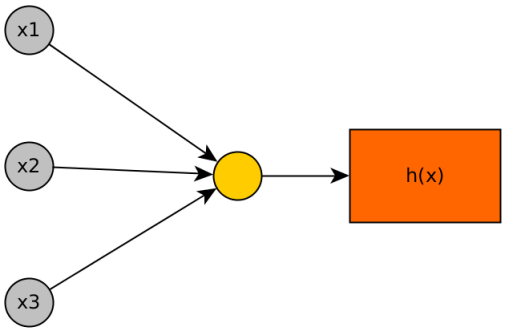
- Algorithms were written to try to mimic the brain.
- Neural Network algorithms were widely used in the 80s and 90s. But in the late 90s, they became less popular.
- There's an idea called the "one learning algorithm" hypothesis. The idea is that even specialized parts of the brain use the same basic learning algorithm. In other words, any sensor will work in any part of the brain.
- There has been a resurgence in the use of Neural Networking algorithms.

1.2 Model Representation I

From biology, neurons have specific structures to perform their tasks.

- *Dendrites* are the 'input wires'
- The *nucleus* form the computational unit in a neuron
- *Axons* are the 'output wires'

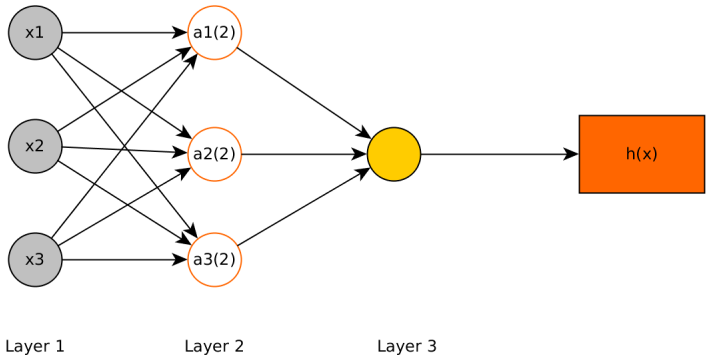
We will represent neurons as *Logistic Units*.



The nodes x_1, x_2, x_3 represent the "input wires." The output is $h_\theta(x)$, where

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}},$$
$$x = [x_0\ x_1\ x_2\ x_3]^T$$
$$\theta = [\theta_0\ \theta_1\ \theta_2\ \theta_3]^T$$

Sometimes the parameter values θ_i are called *weights*.



Layer 1 is also called the 'input layer.' The last layer is called the 'output layer.' Any layers inside input / output layers are called 'hidden layers.' We may also have additional nodes x_0 or $a_0^{(2)}$ called 'bias units.'

Notation:

- $a_i^{(j)}$: activation of unit i in layer j.
- $\Theta^{(j)}$: matrix of weights controlling function mapping from layer j to layer $j+1$. These are the parameters or weights.

Here is an example:

The neural network above has the following computations occuring:

$$a_1^{(2)} = g\left(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3\right)$$
$$a_2^{(2)} = g\left(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3\right)$$
$$a_3^{(2)} = g\left(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3\right)$$

$$h_\Theta(x) = a_1^{(3)} = g\left(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)}\right)$$

We have three input units and three hidden units. So

$$\Theta^{(1)} \in \mathbb{R}^{3\times 4}$$

In general, if a network has s_j units in layer j, s_{j+1} units in layer $j+1$, then $\Theta^{(j)}$ will be of dimension $s_{j+1} \times (s_j + 1)$.

1.3 Model Representation II

Here is a vectorized representation. We define

$$a_1^{(2)} = g\left(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3\right)$$
$$= g\left(z_1^{(2)}\right)$$

where the superscript denotes that we are referring to layer 2. Thus

$$a_1^{(2)} = g\left(z_1^{2)}\right)$$
$$a_2^{(2)} = g\left(z_2^{(2)}\right)$$
$$a_3^{(2)} = g\left(z_3^{(2)}\right)$$

The z values are just a linear combination of the Θ matrices. We also define

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

$$z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix} \in \mathbb{R}^3$$

The calculation may be vectorized as follows:

$$z^{(2)} = \Theta^{(1)} x$$
$$a^{(2)} = g\left(z^{(2)}\right)$$

Recall that this function g is the sigmoid function calculated element-wise. We may add a bias unit x_0 by defining $a_0^{(2)} = 1$. If we define

$$a^{(1)} = x$$

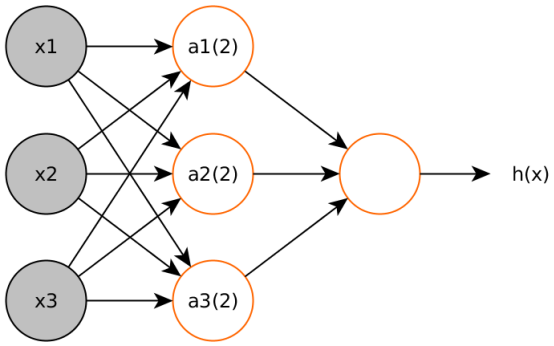
we have a representation $z^{(2)} = \Theta^{(1)} a^{(1)}$ that has the activations on the input layer. Then add $a_0^{(2)}$ to have $a^{(2)} \in \mathbb{R}^4$.

$$z^{(3)} = \Theta^{(2)} a^{(2)}$$
$$h_\Theta(x) = a^{(3)} = g\left(z^{(3)}\right)$$

The process of calculating $h_\Theta(x)$ is called *forward propogation.*

1.4 Neural Network Learning its own Features

Here is a neural network



represented by

$$h_\Theta(x) = g\left(\Theta_{10}^{(1)} a_0^{(2)} + \Theta_{11}^{(1)} a_1^{(2)} + \Theta_{12}^{(1)} a_2^{(2)} + \Theta_{13}^{(1)} a_3^{(2)}\right)$$

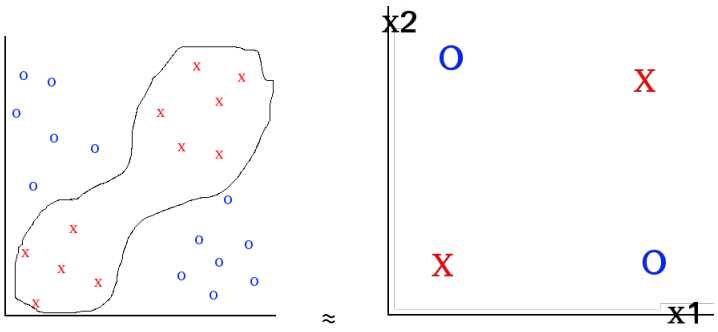
Layer two will have the following representation

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$
$$a^{(2)} = g\left(z^{(2)}\right)$$

This means that the model learns its own features. The next two lectures will have a concrete example.

1.5 A Neural Network Example - Part I

Consider a simplified model that has nonlinear features



The points are

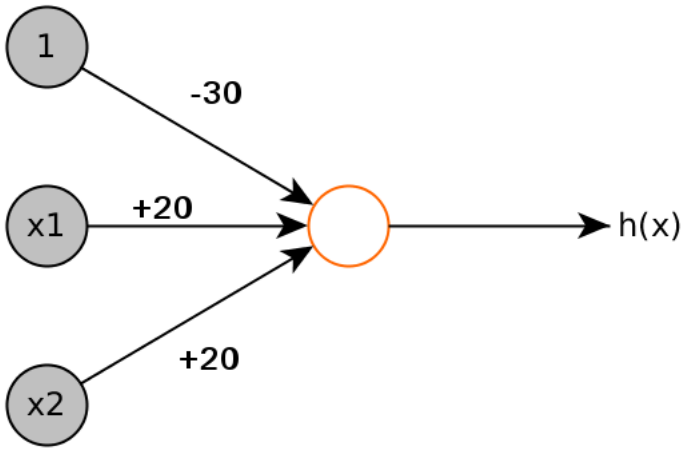
$$(x_1, x_2) = (0,0), (0,1), (1,0), (1,1)$$

where $(0,0)$ and $(1,1)$ correspond to $y = 1$ and the points $(0,1)$ and $(1,0)$ correspond to $y = 0$. This is the same as

$$y = x_1 \text{ XOR } x_2$$

To represent this as a neural network, we construct

$$x_1, x_2 \in \{0,1\}$$
$$y = x_1 \text{ AND } x_2$$



which includes the bias unit +1 and weights. This may be written as

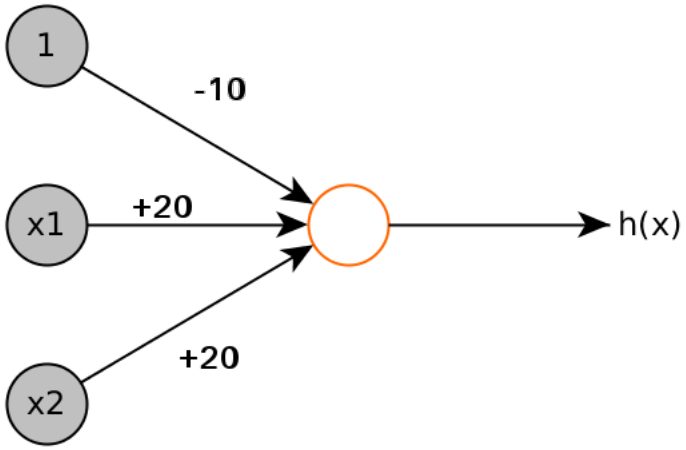
$$h_\Theta(x) = g\left(-30 + 20x_1 + 20x_2\right)$$

where $\Theta_{10}^{(1)} = -30$, $\Theta_{11}^{(1)} = +20$, and $\Theta_{12}^{(1)} = +20$. This neural network will compute the following table:

x_1	x_2	$h_\Theta(x)$
0	0	$g(-30) = (1 + e^{30})^{-1} \approx 0$
0	1	$g(-30 + 20) = (1 + e^{10})^{-1} \approx 0$
1	0	$g(-30 + 20) = (1 + e^{10})^{-1} \approx 0$
1	1	$g(-30 + 20 + 20) = (1 + e^{-10})^{-1} \approx 1$

1.6 Another example

Consider



$$h_\Theta(x) = g(-10 + 20x_1 + 20x_2)$$

The hypothesis $h_\Theta(x)$ produces the following table:

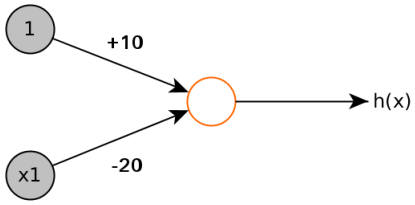
x_1	x_2	$h_\Theta(x)$
0	0	$g(-10) = (1 + e^{10})^{-1} \approx 0$
0	1	$g(-10 + 20) = (1 + e^{-10})^{-1} \approx 1$
1	0	$g(-10 + 20) = (1 + e^{-10})^{-1} \approx 1$
1	1	$g(-10 + 20 + 20) = (1 + e^{-30})^{-1} \approx 1$

which represents

$$x_1 \text{ OR } x_2$$

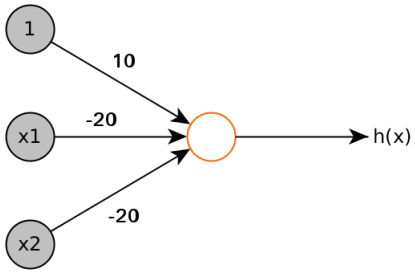
1.7 A Neural Network Example - Part II

Here is a neural network to produce NOT x_1.

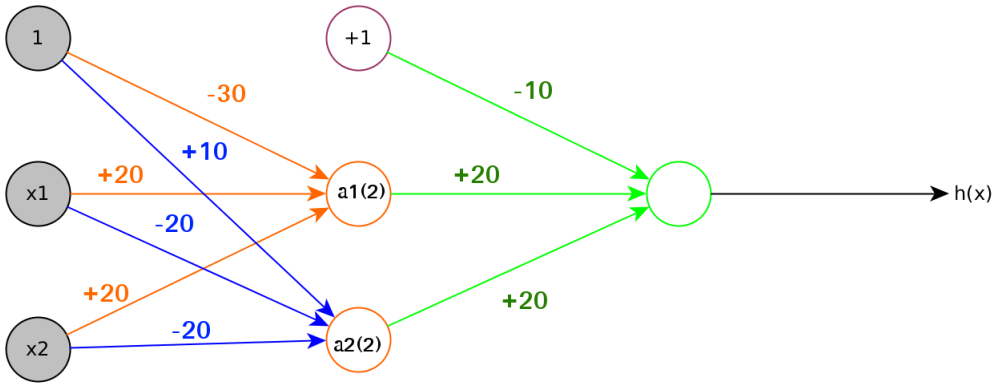


x_1	$h_\Theta(x)$
0	$g(10) \approx 1$
1	$g(-10) \approx 0$

Similarly, this neural network produces (NOT x_1) AND (NOT x_2) :



Here is x_1 XNOR x_2:



which produces this table:

x_1	x_2	$a_1^{(2)}$	$a_2^{(2)}$	$h_\Theta(x)$
0	0	0	1	1
0	1	0	0	0
1	0	0	0	0
1	1	1	0	1

1.8 Multiclass Classification with Neural Networks

We use the "One-Vs.-All' method using Logistic classifiers. For example, if we have {pedestrian, car, motorcycle,truck}, we want

$$h_\Theta(x) \approx \begin{bmatrix}1\\0\\0\\0\end{bmatrix} \text{ when 'pedestrian,' } h_\Theta(x) \approx \begin{bmatrix}0\\1\\0\\0\end{bmatrix} \text{ when 'car,' } h_\Theta(x) \approx \begin{bmatrix}0\\0\\1\\0\end{bmatrix} \text{ when 'motorcycle,'} etc.$$