Neutral Networks and Applications

Class	Machine Learning
Created	@Jun 14, 2020 11:41 PM
Materials	
Reviewed	☐
Type	Lecture Notes

Why Neural Networks?

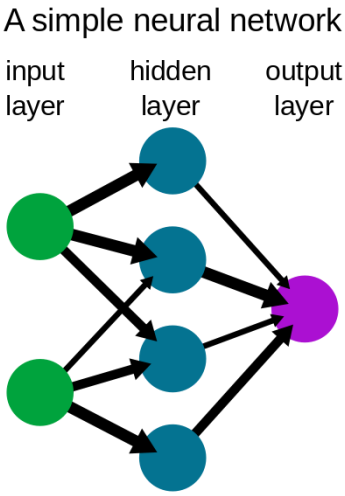
When you were young, you didn't know how things worked. You learnt to walk, talk, sing, love, read all by your own. Because of the neurons in your brain. And that's something that really bothers a computer scientist. Can we mimic the same network in a computer to make it think. This is the motivation for neural networks. One of the most powerful state of the art algorithm of the 21st century.

One of the motivations that we can think of as to why we need to use neural networks is in the case of a supervised classification problem where in it involves a non-linear hypothesis function which includes polynomial terms. When we have about a 100 features say x1, x2... x100. Then just including the product terms of each of these will shoot the feature space to about 5000 features. When we include more polynomial terms, it will increase even further and its computationally expensive.

History of Neural Networks and What are they?

- Origins: Algorithms that try to mimic the brain. Was very widely used in 80s and early 90s; popularity diminished in the late 90s because it was getting computationally expensive to move further in the field.

- A neural network is a network or circuit of neurons, or in a modern sense, an artificial neural network, composed of artificial neurons or nodes.
- More on how neural networks work later in the page.



▼ MODEL REPRESENTATION

- At a very simple level, neurons are basically computational units that take inputs (**dendrites -** referring to the brain) as electrical inputs (called "spikes") that are channeled to outputs (**axons**).
- In our model, our dendrites are like the input features *x1...xn* , 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.
- We use the same logistic function as in classification which is the sigmoid **activation function.** In this situation, our **theta** parameters are sometimes called as **weights**.
- Our input nodes (layer 1), also known as the "input layer", go into another node (layer 2), which finally outputs the hypothesis function, known as the "output layer". We can have intermediate layers of nodes between the input and output layers called the "hidden layers."
- In this example, we label these intermediate or "hidden" layer nodes as a0^2... an^2 and call them activation units.

$a_i^{(j)}$ = "activation" of unit i in layer j

$\Theta^{(j)}$ = matrix of weights controlling function mapping from layer j to layer $j+1$

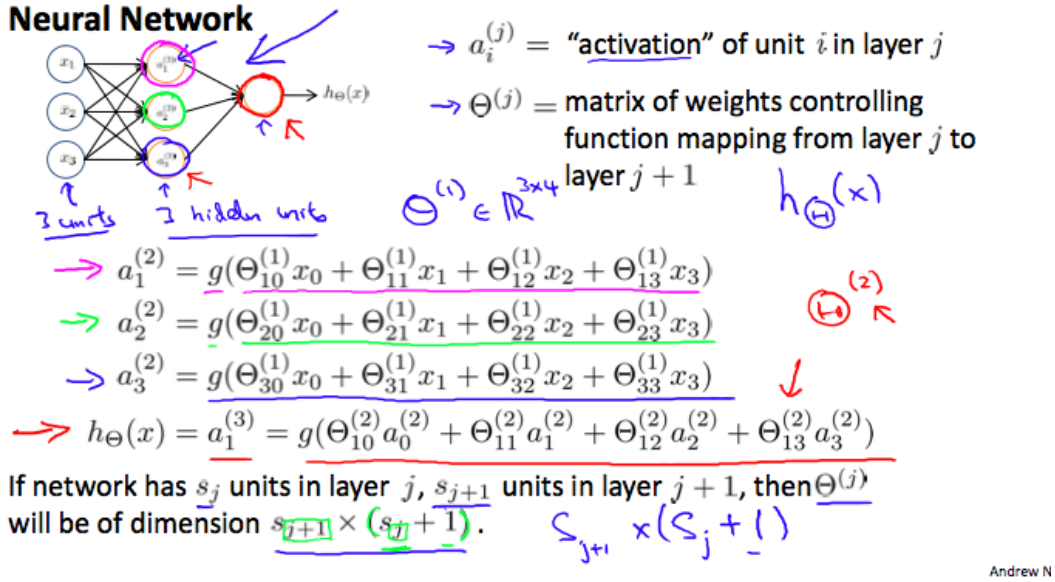
- If we had one hidden layer, it would look like this:

$$\left[x_0 x_1 x_2 x_3\right] \rightarrow \left[a_1^{(2)} a_2^{(2)} a_3^{(2)}\right] \rightarrow h_\theta(x)$$

- The values of each of the **activation nodes** is obtained as follows:

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

  - Here, g(x) is the activation function and (a1)^3 is the activation of the final unit which is the output unit.
- We are computing 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*
- All of this is summarized in this one picture



Intuitive Example

- We shall opt to do the above model representation in a vectorized format. So, as we previously saw:

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

- We shall declare a new variable zk^(j) that encompasses the parameters inside our g function. As in the above example, if we replaced by the variable z for all the parameters we would get:

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

- So, to visualize it:

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

- The vectorized implementation of the input layer: X and the z vector is as follows:

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

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** which is multiplied by one column $a^{(j)}$ 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.

▼ APPLICATIONS ~ INTUITIONS

We can perform the **Logical AND** operation using neural networks!

- The graph of our function would look like this:

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} \rightarrow \left[g(z^{(2)}) \right] \rightarrow h_\Theta(x)$$

- **Remember that the value of x0 is always 1.**

- The first theta matrix be defined as follows:

$$\Theta^{(1)} = \begin{bmatrix} -30 & 20 & 20 \end{bmatrix}$$

- Using this theta matrix, we cause the output of our hypothesis to only be positive if both x1 and x2 are 1. That is:

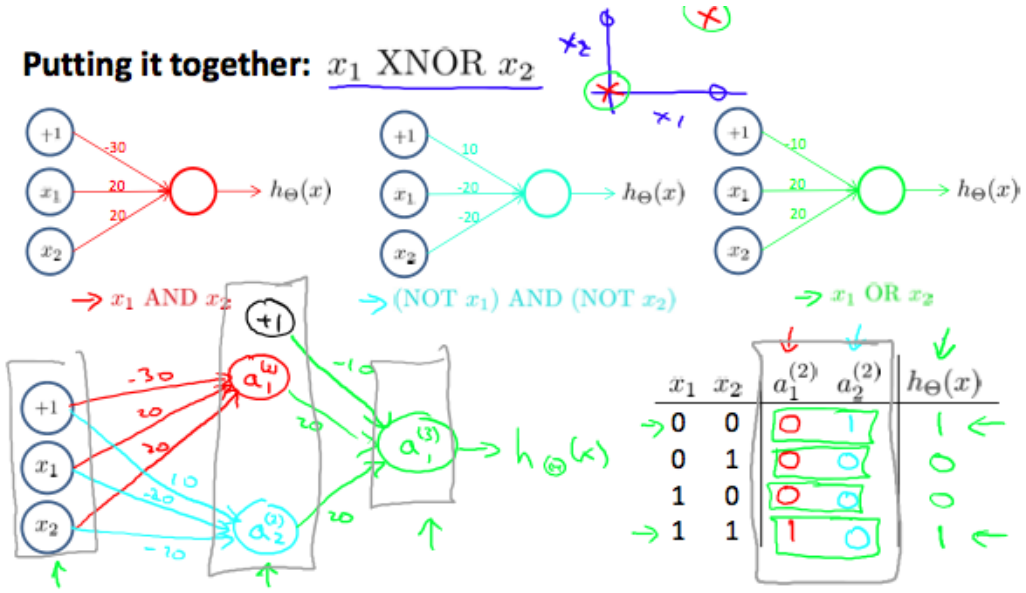
$$h_\Theta(x) = g(-30 + 20x_1 + 20x_2)$$

$$\begin{aligned} x_1 = 0 \ and \ x_2 = 0 \ then \ g(-30) \approx 0 \\ x_1 = 0 \ and \ x_2 = 1 \ then \ g(-10) \approx 0 \\ x_1 = 1 \ and \ x_2 = 0 \ then \ g(-10) \approx 0 \\ x_1 = 1 \ and \ x_2 = 1 \ then \ g(10) \approx 1 \end{aligned}$$

- We can also perform the same for the **Logical OR** gate. Only changing the theta values to be -10, 20, 20.
- So, the theta(1) matrices for AND, NOR, and OR are:

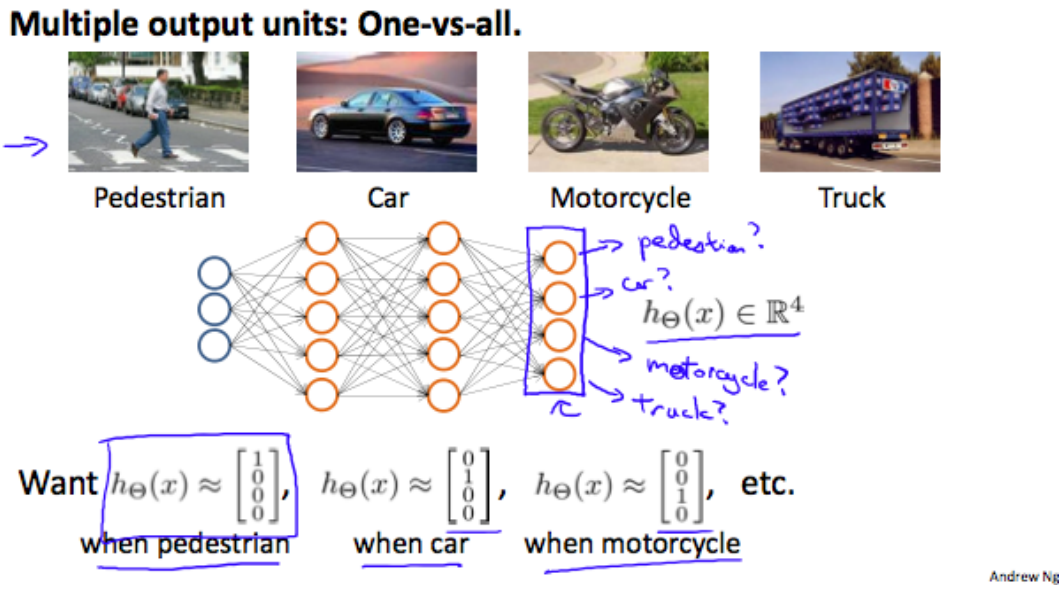
$$\begin{aligned} &AND: \\ &\quad \Theta^{(1)} = \begin{bmatrix} -30 & 20 & 20 \end{bmatrix} \\ &NOR: \\ &\quad \Theta^{(1)} = \begin{bmatrix} 10 & -20 & -20 \end{bmatrix} \\ &OR: \\ &\quad \Theta^{(1)} = \begin{bmatrix} -10 & 20 & 20 \end{bmatrix} \end{aligned}$$

- We can combine these to get the XNOR logical operator which gives 1 if x1 and x2 are both 0 or both 1.



▼ MULTICLASS CLASSIFICATION

- In the previous examples, we saw that the output unit was only one which can be used as binary classification({1,0}). But to have a classification model for more than two classes, we can have as many output nodes as we want to that is the hypothesis function needs to return a vector of values.
- Say, we want to classify our data into one of four categories. We will use the following example to see how this classification is done. This algorithm takes as input an image and classifies it accordingly:



- We can define our set of resulting classes as y

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

- Each y(i) represents a different image corresponding to either a car, pedestrian, truck, or motorcycle. The inner layers, each provide us with some new information which leads to our final hypothesis function. The setup looks like:

$$\begin{bmatrix}x_0\\x_1\\x_2\\\cdots\\x_n\end{bmatrix} \rightarrow \begin{bmatrix}a_0^{(2)}\\a_1^{(2)}\\a_2^{(2)}\\\cdots\end{bmatrix} \rightarrow \begin{bmatrix}a_0^{(3)}\\a_1^{(3)}\\a_2^{(3)}\\\cdots\end{bmatrix} \rightarrow \cdots \rightarrow \begin{bmatrix}h_\Theta(x)_1\\h_\Theta(x)_2\\h_\Theta(x)_3\\h_\Theta(x)_4\end{bmatrix}$$