Neural Networks

May 28, 2012

1 Neural Networks: Representation

1.1 Non-linear Hypotheses

Logistic regression applied to classification, can work for a small number of features, say 2 but for a lot of ML problems we have many more features.
Take a housing prediction, say that a house will be sold within the next 6 months.

$$number\,of\,features = n + n * (n + 1)/2$$

The number of features grows asymptotically in $O(n^2)$.
It's computationally expensive to work with that many features. One thing to do could be work only with quadratic features, but that isn't enough features, you can fit ellipses, but nont complex polynomials.
We need a non-linear hypothesis because when classifying, say images, we have a large number of features lying in different areas.
For third order polynomial features there will be more than $O(n^3)$ such features.

1.2 Neurons and the Brain

Originally developed to have computers mimic the brain.
Widely used in 80s and early 90s with diminishing popularity until recently.
The "one learning algorithm" hypothesis. Auditory cortex learns to see, say if, say, the visual nerves get cut. Neuro rewiring experiments.
Metin & Frost, 1989.
BrainPort; welsh & Blasch, 1997;
Nage et al., 2005;
Constantine-Paton & Law, 2009;

1.3 Model Representation I

Neuron model - Logistic unit:

- input
- output

- bias unit $x_0 = 1$
- sigmoid (logistic) activation function $g(z)$

$$h_\Theta(x) = \frac{1}{1+e^{-\Theta^T x}}$$

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

$$\Theta = \begin{bmatrix} \Theta_0 \\ \Theta_1 \\ \Theta_2 \\ \Theta_3 \end{bmatrix} \longrightarrow weights\,(parameters)$$

$$g(z) = \frac{1}{1+e^{-z}}$$

$$z = \Theta^T x$$

Neural network is a network of the above units.

- Layer 1 - input layer, input features x_j
- Layer 2 - hidden layer (values not in training set) $a_i^{(j)}$
- Layer 3 - output layer - outputs the final value computed by the hypothesis $h_\Theta(x)$.

$a_i^{(j)}$ $\longrightarrow$"activation" of unit i in layer j. "activation" is the value computed and output by a specific ?
$\Theta^{(j)}$ $\longrightarrow$matrix of weights controlling function mapping from layer j to layer $j+1$
Figure at 8:50:

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

$$a_2^{(2)} = g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3)$$

$$a_3^{(2)} = g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3)$$

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

$\Theta^{(1)} \in \mathbb{R}^{3\times 4}$for 3 input, 3 hidden layers, according to:
If a network has $s_j$units in layer j, s_{j+1}unites in layer $j+1$, then $\Theta^{(j)}$will be of dimension $s_{j+1} \times (s_j + 1)$.
So a neural network maps from a space of predictions $h_\Theta(x)$ parameterized by Θ to some space of predicitons y. As we vary Θ we get a different mapping.

1.4 Model Representation II

1.4.1 Forward propagation: Vectorized implementation

Consider the network:

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

$$a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3)$$

$$a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3)$$

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

definitions:

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

so $a_1^{(2)} = g(z_1^{(2)})$ and similarly for $a_2^{(2)}$and $a_3^{(2)}$.
The z values are weighted linear combinations of input values x_0, x_1, x_2, $x_3$that go into the particular neuron.
Let's define:

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

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

now we can vectorize:

$$z^{(2)} = \Theta^{(1)} x$$

$$a^{(2)} = g(z^{(2)}) \quad , z^{(2)} \in \mathbb{R}^3$$

$a^{(2)} \in \mathbb{R}^3$ $\therefore$the activation g applies sigmoid function element wise to each of $z^{(2)}$elements.
We can also think of inputs as activations $a^{(1)}$, so we can rewrite:

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

add bias unit:

$$a_0^{(2)} = 1 \quad , a^{(2)} \in \mathbb{R}^4$$

$$z^{(3)} = \Theta^{(2)}a^{(2)}$$

Finally the activation of the only unit in the output layer,

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

This process is also called **forward propagation**.
Neural Network learning its own features.
The process is similar to logistic regression where

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

but rather than using features x_1, x_2, $\ldots x_n$, we use a_1, a_2, $\ldots x_n$. These features are themselves learned as functions of the input. Concretely, the function mapping from layer 1 to layer 2 is determined by some other set of parameters Θ_1. So instead of the network being constrained by feeding the features x_1, x_2, $\ldots x_n$, it gets to learn its own set of features, a_1, a_2, $\ldots x_n$, to feed into logistic regression. You can learn a better hypothesis if you were constrained.

There can be different network architectures. Each layers can learn more complex features.

Octave implementation
loops:

```
% Theta1 is Theta with superscript "(1)" from lecture
% ie, the matrix of parameters for the mapping from layer 1 (input) to layer 2
% Theta1 has size 3x3
% Assume 'sigmoid' is a built-in function to compute 1 / (1 + exp(-z))

a2 = zeros (3, 1);
for i = 1:3
  for j = 1:3
    a2(i) = a2(i) + x(j) * Theta1(i, j);
  end
  a2(i) = sigmoid (a2(i));
end
```

vectorized:

```
a2 = sigmoid(Theta1 * x);
```

1.5 Examples and Intuitions I

Non-linear classification example: XOR/XNOR.
x_1, $x_2$are binary ($0\, or\, 1$). We want to learn a non-linear boundary that separates positive and negative examples.

$y = x_1\, XOR\, x_2$, however it turns out XNOR is better suited, so

$$y = x_1\, XNOR\, x_2 = NOT(x_1\, XOR\, x_2)$$

Let's start with a simpler network: AND
inputs:

$$x_{1,}\, x_2 \in \{0,\, 1\}$$

target labels:

$$y = x_1\, AND\, x_2$$

you can associate edges of the network with Θvalues, so:

$$\Theta_{10}^{(1)} = -30 \longrightarrow for\, x_0 = 1$$

$$\Theta_{11}^{(1)} = 20$$

$$\Theta_{12}^{(1)} = 20$$

and so

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

You can now plot the sigmoid function and estimate $h_\Theta(x)\ \forall_{x_1,\, x_2 \in \{0,1\}}$.
Look at the four possible input values for $x_1$and x_2 and what the hypothesis will output.

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

if you look at the results of the hypotheses, that's exactly the AND function.

1.6 Examples and Intuitions II

For negations, put a large negative weight infront the variable you want to negate.

$x_1\, XNOR\, x_2$ 04:30
Use 2 activations for layer 2: $x_1\, AND\, x_2$and $(NOT\, x_1)\, AND\, (NOT\, x_2)$. Activation for the output layer is $x_1\, OR\, x_2$, add bias unit.

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

1.7 Multiclass Classification

We have more than one category that we want to distinguish between.
One-vs-all:
say $h_\Theta(x) \in \mathbb{R}^4$and:
training set:

$$(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), ..., (x^{(i)}, y^{(i)}),$$

one training example is one pair of:

$$(x^{(i)}, y^{(i)})$$

where $x^{(i)}$is an image with one of the objects and $y^{(i)}$is one of the below vectors.

The output layer has essentially 4 logistic regression classifiers
$h_\Theta(x) \in \mathbb{R}^4$ one of:

$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

for pedestrian, car, motorcycle, truck respectively.
$y^{(i)}$ one of:

$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

for pedestrian, car, motorcycle, truck respectively.