Thiago G. Martins

Data Scientist at Yahoo!

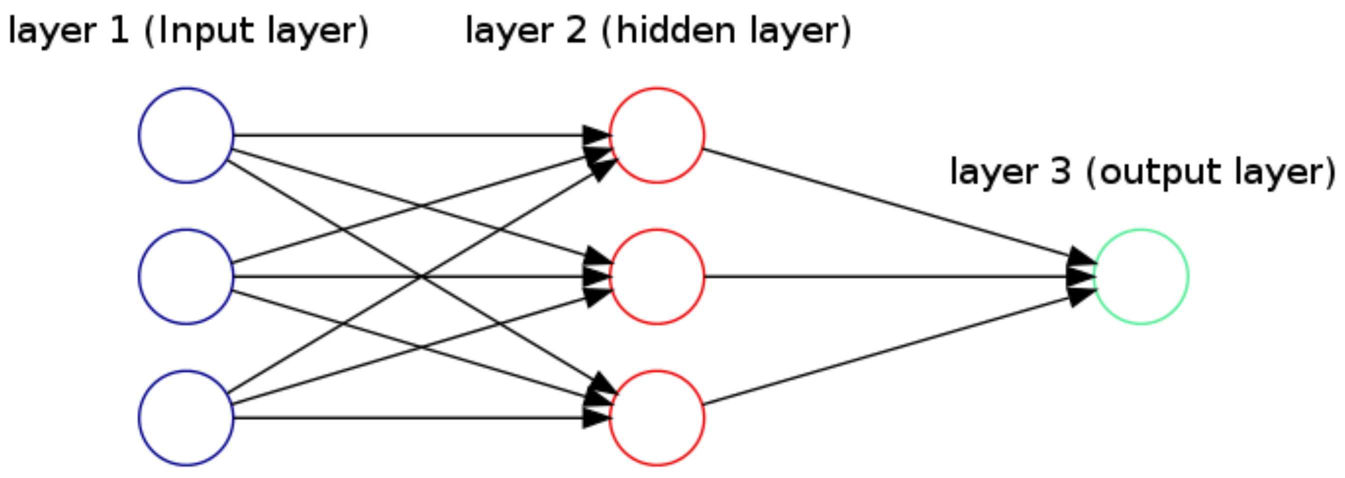
4th and 5th week of Coursera's Machine Learning (neural networks)

Posted on June 5, 2013 by thiagogm

The fourth and fifth weeks of the Andrew Ng's Machine Learning course at Coursera (https://www.coursera.org/course/ml) were about Neural Networks. From picking a neural network architecture to how to fit them to data at hand, as well as some practical advice. Following are my notes about it.

A simple Neural Network diagram

Figure 1 represents a neural network with three layers. The first (blue) layer, denoted as $a^{(1)} = (a_1^{(1)}, a_2^{(1)}, a_3^{(1)})^T$, has three nodes and is called the input layer because its nodes are formed by the covariate/features $x = (x_1, x_2, x_3)$, so that $a^{(1)} = x$.



(https://tgmstat.files.wordpress.com/2013/05/neural_network_example1.png)

Figure 1: An example of a neural network diagram with one output unit for binary classification problems.

The second (red) layer, denoted as $a^{(2)} = (a_1^{(2)}, a_2^{(2)}, a_3^{(2)})^T$, is called a hidden layer because we don't observe but rather compute the value of its nodes. The components of $a^{(2)}$ are given by a non-linear function applied to a linear combination of the nodes of the previous layer, so that

$$\begin{aligned} a_1^{(2)} &= g(\Theta_{11}^{(1)}a_1^{(1)} + \Theta_{12}^{(1)}a_2^{(1)} + \Theta_{13}^{(1)}a_3^{(1)}) \\ a_2^{(2)} &= g(\Theta_{21}^{(1)}a_1^{(1)} + \Theta_{22}^{(1)}a_2^{(1)} + \Theta_{23}^{(1)}a_3^{(1)}) \\ a_3^{(2)} &= g(\Theta_{31}^{(1)}a_1^{(1)} + \Theta_{32}^{(1)}a_2^{(1)} + \Theta_{33}^{(1)}a_3^{(1)}), \end{aligned}$$

where $g(x) = 1/(1 + e^{-z})$ is the sigmoid/logistic function.

The third (green) layer, denoted as $a^{(3)} = (a_1^{(3)})$, is called the output layer (later we see that in a multi-class classification the output layer can have more than one element) because it returns the hypothesis function $h_\Theta(x)$, which is again a non-linear function applied to a linear combination of the nodes of the previous layer,

$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)}). \qquad (1)$$

So, $a^{(j)}$ denotes the elements on layer j, $a_i^{(j)}$ denotes the i-th unit in layer j and $\Theta^{(j)}$ denotes thematrix of parameters controlling the mapping from layer j to layer $j+1$.

What is going on?

Note that Eq. (1) is similar to the formula used in logistic regression (https://tgmstat.wordpress.com/2013/05/15/third-week-of-courseras-machine-learning/) with the exception that now $h_\theta(x)$ equals $g(\theta^T a^{(2)})$ instead of $g(\theta^T x)$. That is, the original features x are now replaced by the second layer of the neural network.

Although is hard to see exactly what is going on, Eq. (1) uses the nodes in layer 2, $a^{(2)}$, as input to produce the final output of the neural network. And $a^{(2)}$ has each element formed by a non-linear combination of the original features. Intuitively, what the neural network does is to create complex non-linear boundaries that will be used in the classification of future features.

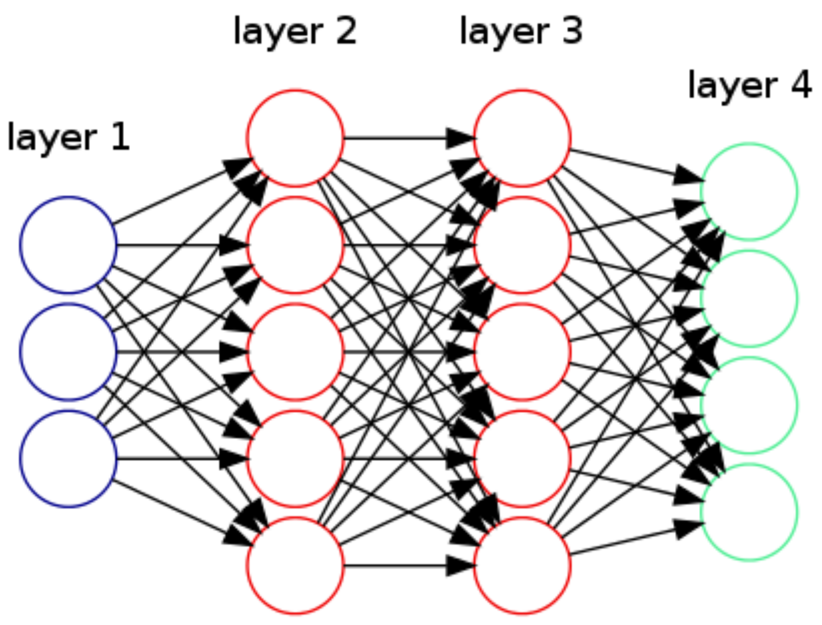
In the third week of the course (https://tgmstat.wordpress.com/2013/05/15/third-week-of-courseras-machine-learning/) it was mentioned that non-linear classification boundaries could be obtained by using non-linear transformation of the original features, like x^2 or $\sqrt(x)$. However, the type of non-linear transformation had to be hand-picked and varies on a case-by-case basis, getting hard to do in cases with large number of features. In a sense, neural network is automating this process of creating non-linear functions of the features to produce non-linear classification boundaries.

Multi-class classification

In a binary classification problem, the target variable can be represented by $y \in \{0, 1\}$ and the neural network has one output unit, as represented by Figure 1. In a neural network context, for a multi-class classification problem with K classes, the target variable is represented by a vector of length K instead of $y \in \{1, ..., K\}$. For example, for $K = 4$, $y \in \mathbb{R}^4$ and can take one of the following vectors:

$$\underset{y=1}{\begin{bmatrix}1\\0\\0\\0\end{bmatrix}} \quad \underset{y=2}{\begin{bmatrix}0\\1\\0\\0\end{bmatrix}} \quad \underset{y=3}{\begin{bmatrix}0\\0\\1\\0\end{bmatrix}} \quad \underset{y=4}{\begin{bmatrix}0\\0\\0\\1\end{bmatrix}}$$

and in this case the output layer will have K output units, as represented in Figure 2 for $K = 4$.



(https://tgmstat.files.wordpress.com/2013/05/multiclass_neural_network_example.png)
Figure 2: An example of a neural network diagram with K=4 output units for a multi-class classification problem.

The neural network represented in Figure 2 is similar to the one in Figure 1. The only difference is that it has one extra hidden layer and that the dimensions of the layers $a^{(j)}$, $j = 1, \ldots, 4$ are different. The math behind it stays exactly the same, as can be seen with the forward propagation algorithm.

Forward propagation: vectorized implementation

Forward propagation shows how to compute $h_\Theta(x)$, for a given x. Assume your neural network has L layers, then the pseudo-code for forward propagation is given by:

Algorithm 1 (forward propagation)

```
1. $a^{(1)} = x$
2. for $l = 1, \ldots, L-1$ do
3.     $z^{(l+1)} = \Theta^{(l)} a^{(l)}$
4.     $a^{(l+1)} = g(z^{(l+1)})$
5. end
6. $h_\Theta(x) = a^{(L)}$
```

The only thing that changes for different neural networks are the number of layers L and the dimensions of the vectors $a^{(j)}$, $z^{(j)}$, $j = 1, \ldots, L$ and matrices $\Theta^{(i)}$, $i = 1, \ldots, L-1$.

Cost function

From now on, assume we have a training set with m data-points, $\{(y^{(i)}, x^{(i)})\}_{i=1}^{m}$. The cost function for a neural network with K output units is very similar to the logistic regression one (https://tgmstat.wordpress.com/2013/05/15/third-week-of-courseras-machine-learning):

$$\begin{aligned} J(\Theta) &= -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log h_\theta(x^{(i)})_k + (1-y_k^{(i)})\log(1-h_\theta(x^{(i)})_k)\right] \\ &+ \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2, \end{aligned}$$

where $h_\theta(x^{(i)})_k$ is the k-th unit of the output layer. The main difference is that now $h_\theta(x^{(i)})$ is computed with the forward propagation algorithm. Technically, everything we have so far is enough for optimization of the cost function above. Many of the modern optimization algorithms allow you to provide just the function to be optimized while derivatives are computed numerically within the optimization routine. However, this can be very inefficient in some cases. The backpropagation algorithm provides an efficient way to compute the gradient of the cost function $J(\Theta)$ of a neural network.

Backpropagation algorithm

For the cost function given above, we have that $\frac{\partial}{\partial\Theta^{(l)}}J(\Theta) = \delta^{(l+1)}(a^{(l)})^T + \lambda\Theta$, where $\delta_j^{(l)}$ can be interpreted as the "error" of node j in layer l and is computed by

Algorithm 2 (backpropagation)

```
1. $\delta^{(L)} = a^{(L)} - y$
2. for $l = L-1, ..., 2$ do
3.   $\delta^{(l)} = (\Theta^{(l)})^T \delta^{(l+1)} .* g'(z^{(l)})$
4. end
```

Knowing how to compute the δ's, the complete pseudo-algorithm to compute the gradient of $J(\Theta)$ is given by

Algorithm 3 (gradient of $J(\Theta)$)

```
1. Set $\Delta^{(l)} = 0,$ for $l = 1, \dots, L-1$
2. for $i = 1, ..., m$ do
3.   Set $a^{(1)} = x^{(i)}$
4.   Compute $a^{(l)}$ for $l = 2, \dots, L$ using forward propagation (Algorithm 1)
5.   Using $y^{(i)}$, compute $\delta^{(l)}$ for $l = L, \dots, 2$ using the backpropagation algorithm (Algorithm 2)
6.   $\Delta^{(l)} = \Delta^{(l)} + \delta^{(l+1)}(a^{(l)})^T$ for $l = L-1, \dots, 1$
7. end
```

Then $\frac{\partial}{\partial\Theta^{(l)}}J(\Theta) = \Delta^{(l)} + \lambda\Theta$

Some practical advice

- Pick a network architecture. The number of input units is given by the dimension of the features $x^{(i)}$. The number of output units is given by the number of classes. So basically we need to decide the number of hidden layers and how many units in each hidden layer. A reasonable default is to have one hidden layer with the number of units equal to 2 times the number of input units. If more than one hidden layer, use the same number of hidden units in every layer. The more hidden units the better, the constrain here is the burden in computational time that increases with the number of hidden units.
- When using numerical optimization you need initial values for Θ. Randomly initialize the parameters Θ so that each $\theta_{ij} \in [-\epsilon, \epsilon]$. Initializing $\theta_{ij} = 0$ for all i, j will result in problems. Basically all hidden units will have the same value, which is clearly undesirable.
- During the building of your neural network algorithm, use numerical derivatives to make sure that your implementation of the backpropagation is correct.

Reference:

– Andrew Ng's Machine Learning course at Coursera (https://www.coursera.org/course/ml)

Related posts:

– First two weeks of Coursera's Machine Learning (linear regression) (https://tgmstat.wordpress.com/2013/05/08/first-two-weeks-of-courseras-machine-learning/)
– Third week of Coursera's Machine Learning (logistic regression) (https://tgmstat.wordpress.com/2013/05/15/third-week-of-courseras-machine-learning/)

Advertisements

This entry was posted in Data Analysis & Statistics, Models and tagged classification, coursera, data analysis, machine learning, multiclass classification, neural network, pattern recognition, statistics. Bookmark the permalink.

Create a free website or blog at WordPress.com.