1 Neural networks (for classification or regression)

Used to estimate unknown functions (complex, non-linear hypothesis) that are based on a large number of inputs, through the back-propagation algorithm.
Generally more complex and computationally expensive than other methods, but powerful for certain problems.
The basis of many deep learning methods.
Today is the state of the art technique for many different machine learning problems.

1.1 Description

Neural networks were developed to vaguely simulate the neurons in the brain.

A neuron is a computational unit that receives a number of inputs through its input wires (*dendrites*), does some computation and then sends signals through its output wire (*axon*) to other neurons in the brain.
A neural network is a group of neurons. The inputs are groupes in an *input layer*. The outputs are grouped in a final *output layer*. The layers in between are called the *hidden layers*.
Adding more layers helps computing even more complex functions on the input data.

1.2 Notation

- training set $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}$ with m training samples
- input variable with n features: $x^{(i)} = \{x_1^{(i)}, x_2^{(i)}, \dots, x_n^{(i)}\}$
- output variable $y^{(i)}$, represented by either a single value (in case of regression or binary classification) or by a probability vector (in case of multi-class classification)
- L: the total number of layers in the network (including the input and the output layer)
- s_l: the number of units (neurons) in layer l
- $\Theta^{(l)}$: matrix of weights (parameters) controlling function mapping from layer $l-1$ to layer l
- $\Theta_{ij}^{(l)}$: weight from unit i in layer $l-1$ to unit j in layer l
- $z_j^{(l)} = \Theta_{0j}^{(l)} a_0^{(l-1)} + \Theta_{1j}^{(l)} a_1^{(l-1)} + \dots + \Theta_{s_{l-1}j}^{(l)} a_{s_{l-1}}^{(l-1)}$: product between weights and inputs
- $a_j^{(l)} = g(z_j^{(l)})$: *activation* of unit j in layer l
- output $\hat{y} = h_\Theta(x)$

1.3 Neural networks for multi-class classification

- Model's architecture
  - fully connected network
  - L layers
  - input layer with m units (a training sample x)
  - output layer with K units ($\hat{y} = h_\Theta(x) \in \mathrm{R}^K$)

- $L - 2$ hidden layers with $s^{(l)}$ units (for $l \in \{2, \dots, L-1\}$)

- Model's parameters

  $\Theta = \{\theta^{(1)}, \theta^{(2)}, \dots, \theta^{(L)}\}$

- Cost function

  $$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{K} \left(-y_k^{(i)} \, log \, \hat{y}_k^{(i)} - (1 - y_k^{(i)}) \, log \, (1 - \hat{y}_k^{(i)}) \right)$$

- Goal

  $\min_{\Theta} J(\Theta)$

- Algorithm

  - start with some initial values for Θ (usually random values in $[-\epsilon, \epsilon]$)
  - set $\Delta_{ij}^{l} = 0$ (accumulate the partial derivatives $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$)
  - for $i = 1$ *to* m
    - set $a^{(1)} = x^{(i)}$
    - perform forward-propagation to compute $a^{(l)}$ for $l = \{2, 3, \dots, L\}$
      - $z^{(l)} = \Theta^{(l)} a^{(l-1)}$
      - $a^{(l)} = g(z^{(l)})$
    - compute the output error $\delta^{(L)} = a^{(L)} - y^{(i)}$
    - perform back-propagation
      - back propagate error trough each layer $\delta^{(L-1)}, \delta^{(L-2)}, \dots, \delta^{(2)}$

        $$\delta^{(l-1)} = (\Theta^{l-1})^T \delta^{(l)} \cdot \left(a^{(l)} * (1 - a^{(l)}) \right)$$

      - compute the gradients: $\Delta_{ij}^{l} = \Delta_{ij}^{l} + a_j^{(l)} \delta_i^{(l+1)}$