Coursera - Deep Learning Specialization

andrijazz

February, 2018

Chapter 1

Neural Networks and Deep Learning

1.1 Introduction to Deep Learning

Andrew Ng is introducing *ReLU* (Rectified Linear Unit) function which is often seen in deep learning literature as an example of the simplest choice for activation function.

$$f(x) = \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases} \tag{1.1}$$

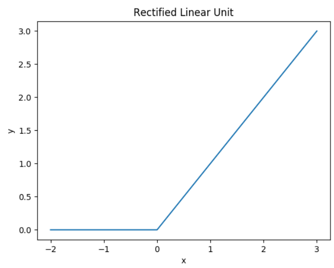

Figure 1.1: ReLU function

One of the major breakthroughs in NN computations was introducing *ReLU* function instead of *Sigmoid* function $f(x) = \frac{1}{1+e^{-x}}$. The advantage of *ReLU* is that has value 0 when $x < 0$, where *Sigmoid* function has values that are close to 0 and thus making the computation harder. Additionally, gradient of *ReLU* function is equal to 1 for all positive x's and thus making the gradient descent runs much faster.

Especially interesting in Week 1 of the course was interview with Geoffry Hinton where a lot of interesting ideas were mentioned so it would be really beneficial to watch this video again once student gain more knowledge about different models like Boltzman Machines etc.

1.2 Neural Network Basics

1.2.1 Image representation

Image is represented with the 3-dimensional matrix where dimensions represent values of Red Green and Blue respectively. 3-dimensional matrix is usually squeezed into 1-dimensional vector and this is what we are considering to be our x in terms of image classification models.

1.2.2 Binary classification and Neural Network Notation

In this section Andrew Ng defines binary classification problem, introduces Logistic Regression and *Sigmoid* function but more importantly it introduces neural network notation that will be used throughout the course. He emphasizes that neural network notation will be different from the one used in his first Machine Learning course in a way that bias will be kept separately of parameters vector because it is more natural notation when implementing neural net.

For detail notation take a look at the file *neuralnetworknotation.pdf*.

1.2.3 Logistic regression cost function

We define a **loss function** as a measurement of how well we are doing on a single training example and **cost function** of how well we are doing on the entire training set.

Loss function for Logistic regression is:

$$L(y, \hat{y}) = -(y \log \hat{y} + (1-y) \log (1-\hat{y})) \tag{1.2}$$

where $\hat{y} = \sigma(w^T x + b)$ and σ is *Sigmoid* function (in literature when we write log it usually stands for logarithm with the base of e). The questions that pops up is why we simply can't use $L(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2$ as a loss function? The answer is that optimization problem that we will encounter will become non-convex and thus gradient descent may converge to local optimum instead of global optimum.

Cost function for Logistic regression is:

$$J(W, b) = \frac{1}{m} \sum_{i=1}^{m} L(y^{(i)}, \hat{y}^{(i)}) = -\frac{1}{m} \sum_{i=1}^{m} (y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log (1 - \hat{y}^{(i)})) \tag{1.3}$$

People usually don't do random initialization for Logistic Regression because cost function is convex and thus the usual way is to simply initialize all the parameters to 0.

1.2.4 Logistic regression derivatives