Neural Network and Deep Learning

Qingyun Li

VISION@OUC

July 15, 2018

1 First Week

2 Second Week

2.1 Binary Classification

2.2 Logistic Regression

2.3 Logistic Regression cost function

2.4 Gradient Descent

2.5 Derivatives

In the last segment, in order to aquire beautiful w and b, we use some derivatives in calculus, so Andrew NG give us better intuition about derivatives and even without deep mathematical understanding of calculus with just an intuitive understanding of calculus, we'll be able to make neural networks work effectively.

2.6 Computationn Graph

The computations of a neural network are all organized in terms of a forward path or a forward propagation step in which we compute the output of the neural network followed by a backward pass or a back complication step which we used to compute gradients or compute derivatives. And the computation graph explains why it is organized this way.

2.7 Logistic Regression Gradient descent

In this segment, we learned how to compute derivatives for us to implement gradient descent for logistic regression. And we'll do this computation using computation graph which get us familiar with the gradient descent. If we want to compute derivatives to the loss, the first thing we need to do is going backwards and computing the derivative of this loss. And the final step in back propagation is to go back to compute how much we need to change w and b.

2.8 Gradient descent on m example

To train logistic regression model we have not just one training example. Given entire training set of m training examples, we should take all of them and apply them to learn not just from one example but from an entire training set.

2.9 Vectorization

In order to get rid of explicit for-loops, it is a good idea to use the vectorization, especially when the data sets are large, at that moment, it's important that our code very quickly because otherwise, if it's running on a big data set, we'll take long time to run. So the ability to perform vectorization has become a key skill. This method enables Python numpy to take much better advantage of parallelism to make our computations much faster. And this is true both computations on CPUs and computations on GPUs.

2.10 Vectorizing Logistic Regression

This is how we implement a vectorize implementation of the forward propagation for all M training examples at the same time.

2.11 Broadcasting in Python

There is some general principle of broadcasting in Python. If we have an m by n matrix and we add or subtract or multiply or divide with a 1 by n matrix, then this will copy in n times into a m by n matrix, and then apply the addition, subtraction and multiplication of division element wise.

2.12 A note on python/numpy vectors

When we use broadcasting operations in python, they give us great flexibility which is a strength as well as a weakness of the programmimg language, they creat expressivity of the language, but it's possible we can introduce very subtle bugs or very strange looking bugs, if we're not familiar with all of the intricacies of how broadcasting and how features like broadcasting work. And there is an internal logic to these strange effects of Python. If we are not familiar with the Python, some strange bugs will appear. So Andrew Ng share some couple tips and tricks with us that could help us to eliminate or simplify the strange looking bugs in the code. such as, when we're coding neural networks that we just not use data structures where the shape is 5 or rank 1 array.

3 Third weeks

3.1 Neural network representation

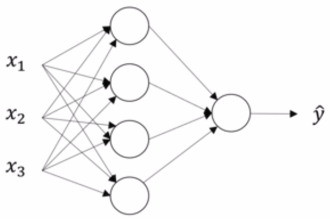

Figure 1: A Neural Network

As is shown at Figure. 1, this is a picture of neural network, and we give different parts of these pictures some names, the first layer is the input layer of the neural network, this contains the inputs to the neural network. The medium layer is the hidden layer of the neural network. The final layer is formed by in this case just one node, and this single note is called the output layer, it's responsible for generating the predicted value. The term hidden layer refers to the fact that in a training set the true values for these nodes in the middle are not observed that is we don't see what they should be in the training set. And in fact, we're always calling the input layer 0, so the hidden layer is layer 1, and the output layer is layer 2. And people always refer to particular neural network as a two layer neural network because we don't count input layer as an official layer.

3.2 Activation functions

When we build a neural network, one of the choices we get to make is what activation functions use in the hidden layers as well as at the output unit of our neural network. So far we've just been using the sigmoid activation function, but sometimes other chioces can work much better. Such as tanh function, ReLU function, or leaky ReLU function. As shown at the Figure. 2.

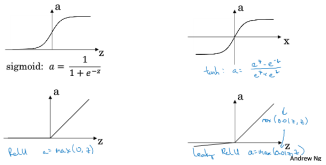

Figure 2: Activation Function

In fact, these functions always work better than the sigmoid function. Such as the tanh function, its values are always between plus one and minus one, the mean of the activations that come out of our head in there are closer to having a zero mean, and using tanh function also has the effect of centering our data, so that the mean of the data is close to the zero. And this actually makes learning for the next layer a little bit easier.