notes_course_1

March 11, 2018

1 Chapter 1. Neural Networks and Deep Learning

1.1 Introduction to Deep Learning

We are introducing *ReLU* (Rectified Linear Unit) function which is used as a activation function.

$$f(x) = \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases} \tag{1}$$

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 Geoffrey Hinton where a lot of interesting ideas were mentioned so it would be really beneficial to watch this video again once I gain more knowledge about different models like Boltzman Machines etc.

```
In [1]: import matplotlib.pyplot as plt
        import numpy as np


        def relu(x):
            return np.maximum(x, 0)


        x = np.array([-2, -1, 0, 1, 2, 3])
        y = relu(x)
        plt.plot(x, y)

        plt.xlabel('x')
        plt.ylabel('y')
        plt.title('Rectified Linear Unit')
        # plt.grid(True)
        plt.savefig("relu.png")
        plt.show()
```

```
<matplotlib.figure.Figure at 0x7fe1884e55c0>
```

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:

$$L(y, \hat{y}) = -(y \log \hat{y} + (1 - y) \log (1 - \hat{y})) \tag{2}$$

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:

$$J(W, b) = \frac{1}{m} \sum_{i=1}^{m} L(y^{(i)}, \hat{y}^{(i)}) = -\frac{1}{m} \sum_{i=1}^{m} (y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log (1 - \hat{y}^{(i)})) \tag{3}$$

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 Broadcasting in Python

General principle for the python broadcasting is that if you have (m, n) matrix and $(m, 1)$ matrix you would be able to apply any of operations $+, -, *, /$ because second matrix will be automatically extended up to (m, n) by repeating the column n times. Similarly, if you have (m, n) matrix and $(1, n)$ matrix you would be able to apply arithmetic operations because second matrix will be automatically extended up to (m, n) by repeating the raw m times.

1.2.5 Tips on numpy

1) Avoid using rank 1 arrays, they are not column vectors nor row vectors

```
In [3]: import numpy as np

        a = np.random.rand(5)

        print(a.shape)
        print(a)

        a = a.reshape(1, 5)
        print(a.shape)
        print(a)
```

```
(5,)
[ 0.50396686  0.68641748  0.99727146  0.78588024  0.1727076 ]
(1, 5)
[[ 0.50396686  0.68641748  0.99727146  0.78588024  0.1727076 ]]
```

2) Use assertions

```
In [4]: assert(a.shape == (1, 5))
```

1.2.6 Explanation of Logistic Regression cost function

Our task is to model the probability $p(y|x)$, so we need to come up with the function, lets name it $\hat{y}$, that will satisfy the requirements: when $y = 1$, $p(y|x) = \hat{y}$ and when $y = 0$, $p(y|x) = 1 - \hat{y}$. These 2 cases described $p(y|x)$ so we are now going to try to write $p(y|x)$ in a single equation:

$$p(y|x) = \hat{y}^y (1 - \hat{y})^{1-y} \tag{4}$$

Since log is strictly monotonically increasing function, instead of modeling $p(y|x)$ we can model $\log(p(y|x)$ and that would give us the same results. Introducing the log function will give us something like this:

$$\log p(y|x) = y \log \hat{y} + (1 - y) \log (1 - \hat{y}) \tag{5}$$

In machine learning we usually want to minimize the loss and that means maximizing the $p(y|x)$ and thus we define $L(y, \hat{y}) = -p(y|x)$. We can now define $\hat{y}$ as a simple polinomial function and add sigmoid to make it satisify probability requirements to be between 0 and 1. Thus $\hat{y} = \sigma(\theta^T x + b)$. Also, when caluclating gradients sigmoid function has a convinient property $\sigma'(x) = \sigma(x)(1 - \sigma(x))$ so that is something that we should keep in mind.

1.2.7 Normalization

Another common technique we use in Machine Learning and Deep Learning is to normalize our data. It often leads to a better performance because gradient descent converges faster after normalization. Here, by normalization we mean changing x to $\frac{x}{\|x\|}$ (dividing each row vector of x by its norm).