Weekly Work Report

Yufeng Jiang

VISION@OUC

July 15, 2018

1 Basics of Neural Network Programming

1.1 Binary Classification

When we have a training set of m training examples, the most method we will use may be training it by having a for loop to step through the m training examples. But, the neural network can help us to process the entire training set without using for loop. Besides, when we organize the computation in our neural network, usually we have a forward propagation step followed by a backward propagation step. In this week, Andrew will use a logistic regression that is an algorithm for binary classification to represent his ideas.

When it comes to binary classification, there is a example. We want to recognize whether is a cat in Figure 1 used at [2] . We input this paper represented by a feature vector x to predict whether the output label y is 1 or 0. In this case, 1 represents that there is a cat in image and 0 represents the contrary answer.

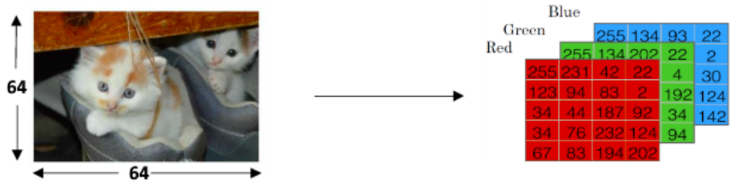


Figure 1: Examples: Cat vs Non-Cat.

The image is stored at a compute by storing three separate matrices corresponding to the red, green and blue color channels of this image. So, if we input an image is 64×64 pixels, the computer will store three 64× matrices corresponding to the red, green and blue pixel intensity values for this image. To put these pixel intensity values into feature vector, we can define a feature vector x to represent this image. If the image is 64 × 64, the total dimension of this vector x is 64×64×3. We use $n_x = 12288$ to represent the dimension of the input features. The purpose of output y is 1 or 0. A single training example is represented by a pair (x,y) where $x \in R^{n_x}$ and $y \in \{0, 1\}$. The training sets comprised by m training examples are [2]

$$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\} \tag{1}$$

We can use $X.shape = (n_x, m)$ to mean that it is an $n_x \times m$ dimension matrice and the label y is used as $Y.shape = (1, m)$ to make our implementation of a neural network easier, just like [2]:

$$Y = [y^{(1)}, y^{(2)}, \dots, y^{(m)}] \tag{2}$$

1.2 Logistic Regression

The input vector x may be an image, and we want to recognize whether is an image with cats. Thus, an algorithm needed to be used to give a prediction $\hat{y}$ like this [2]:

$$\hat{y} = P(y = 1|x) \qquad (0 \leq \hat{y} \leq 1) \tag{3}$$

where $\hat{y}$ is a probaility that the value of y is 1 when the input feature x meet the conditions. For example, $\hat{y}$ gives the probaility that the this is a cat picture, and $x \in \mathbb{R}^{n_x}$ represents that x is an n_x dimensional vector. The parameter of logistic regression w is an n_x dimensional vector, and b is a real number.
Sigmoid function: The output $\hat{y}$ equals to the sigmoid function shhown at Figure 2 cited from [2] applied to this quantity with [2]:

$$\hat{y} = \sigma(w^\top x + b) \tag{4}$$

where $\sigma(z) = \frac{1}{1+e^{-z}}$ whth the character that the $\sigma(z) = 1$ when $z \to +\infty$, the $\sigma(z) = 0$ when $z \to -\infty$ and the $\sigma = 0.5$ when $z = 0$. We usually use the parameter w seperated from b corresponded to an interceptor when we programmed neural network.

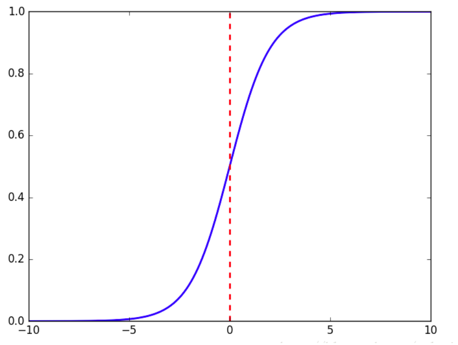


Figure 2: The graphics of sigmoid function.

1.3 Logistic Regression Loss Function and Cost Function

We need to define a cost function to train the parameter w and b of logistic regression model. In the Equation 4, the $\hat{y}$ is defined to a single train example. To learn parameters for out model we are given a training set of m training examples like $\{(x^{(1)}, y^{(1)}), (x^{(2)}), y^{(2)}, \dots, (x^{(m)}, y^{(m)})\}$ and the most we want to do is $\hat{y}(i) \approx y(i)$. For each training example, the prediction on training sample (i) is $\hat{y}^{(i)}$ where the superscript parenthnesses i refers to data.
Loss function: We use Equation 5 used in [2] to difine the loss function in logistic regression [2]

$$\mathcal{L}(\hat{y}, y) = -(y \log(\hat{y}) + (1-y)\log(1-\hat{y})) \tag{5}$$

the most we want to is $\mathcal{L}(\hat{y}, y)$ as little as possible. If $y = 1$, Equation 5 becomes to $\mathcal{L}(\hat{y}, y) = -\log \hat{y}$. So, we need to make $\log(\hat{y})$ large enough, as well as make $\hat{y}$ large enough. However, the $\hat{y}$ is got from sigmoid function, so it is never bigger than 1. In other words, we want to make $\hat{y}$ as big as possible but the most it will get is 1, so we should make $\hat{y}$ close to 1. Similarly, we need to make $\hat{y}$ close to 0 as much as possible when $y = 0$.
Cost function: It is defined the average value among the whole training examples and it measures how well we are doing on a single training example. The cost function is [2]:

$$J(w, b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)} \log(\hat{y}^{(i)}) + (1-y^{(i)})\log(1-\hat{y}^{(i)})] \tag{6}$$

The cost function is the total cost of our parameter compared the loss function is applied the single training sample. Thus, the cost function should little enough only if finding the appropriate parameter w and b while training the logistic regression model.

1.4 Gradient Descent

We use Figure 3 [2] to explain why the gradient descent is a better approach to find the minimization of the cost function.

To find better parameters, the most we need to do is use a initial value to inital the parameter w and b, just like the top red point at Figure 3. The porpose of gradient descent is finding a faster direction to the global optimum. We repeat the Equation 7 used in [2] before the algorithm convergence.

$$
\begin{aligned}
w &:= w - \alpha \frac{dJ(w,b)}{dw} \\
b &:= b - \alpha \frac{dJ(w,b)}{db}
\end{aligned}
\tag{7}
$$

where $:=$ represents updating w and α means learning rate that can control each iteration or the step length in gradient descent algorithm. $\frac{dJ(w,b)}{dw}$ is insteaded by dw and $\frac{dJ(w,b)}{db}$ while programming.

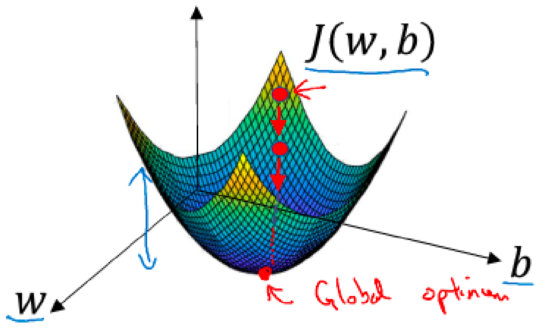
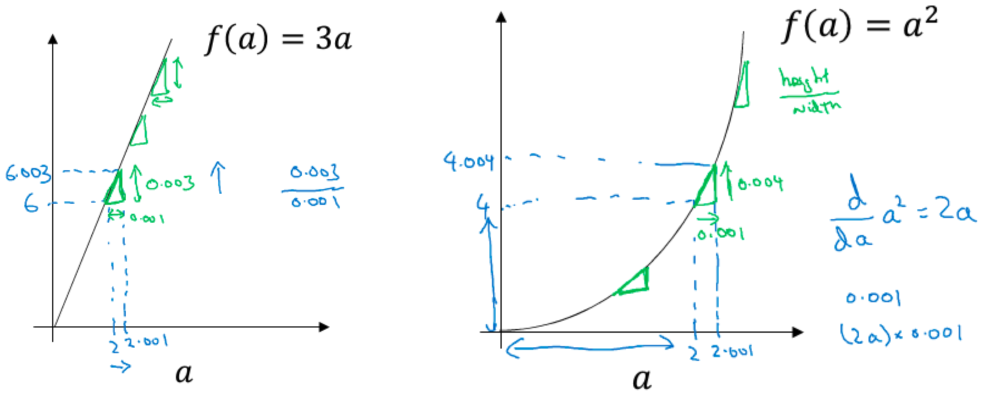


Figure 3: The graphics of cost function.

1.5 Derivatives

In the Figure 4(a) shown at [2], whatever the value of a if we increase it by 0.001 that value of f of a goes up by three times as much. So this function has the same slope everywhere. Different in Figure 4(b), the derivative of the function just means the slope of a funciton and the slope of a function can be different at differnet points on the function.



(a) (b)

Figure 4: Intuition about derivation: (a) the slope is invariable; (b) the slope is variable.

1.6 Computation Graph

The computations of a neural network are all organized in terms of a forward path or a forward propagation step. Firstly, we compute the output of the neural network followed by a backward pass or a back complication step which we use to compute gradients or compute derivatives. The blue arrow in Figure 5.

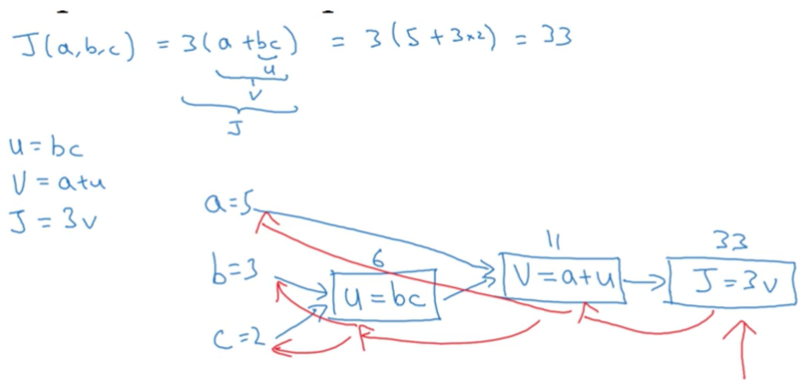


Figure 5: The explination of computation graph.

1.7 Logistic Regression Gradient descent

In this part, we mainly study how to compute derivatives to implement gradient descent for logistic regrsesion. In Figure 6 used at [2], we show the computation graph with only two input x_1 and x_2 to implement the logistic regression derivatives.

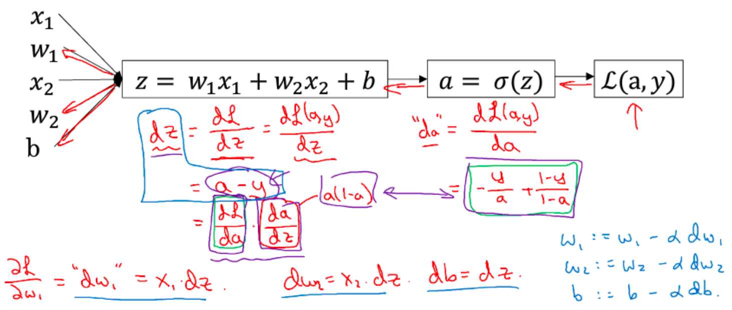


Figure 6: Logistic regression derivatives.

where $\frac{d\mathcal{L}(a,y)}{da}$ can be represented as da, and $\frac{d\mathcal{L}(a,y)}{dz}$ is insteaded by dz in our code. After computing, we can get the relation of $dw_1 = \frac{\partial L}{\partial w_1} = x_1 \cdot dz$, $dw_2 = \frac{\partial L}{\partial w_2} = x_2 dz$ and $db = \frac{\partial L}{\partial b} = dz$.

The steps of using the gradient descent are: getting dz through $dz = a - y$, and then computing dw and db, updating w and b via Equation 7.

1.8 Vectorization

The gradient descent of logistic regression we said before is based on one sample, when we have m samples, functions we proposed before will be translated as:

$$\begin{aligned} J(w,b) &= \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(a^{(i)},y^{(i)}) \\ a^{(i)} &= \hat{y}^{(i)} = \sigma(z^{(i)}) = \sigma(w^\top x^{(i)} + b) \\ \frac{\partial}{\partial w}J(w,b) &= \frac{1}{m}\sum_{i=1}^{m}\underbrace{\frac{\partial}{\partial w}\mathcal{L}(a^{(i)},y^{(i)})}_{dw^{(i)}} \end{aligned} \tag{8}$$

Using for loop to realize this gradient descent, we will spend more time. Instead, applying deep learning algorithms named vectorization, we will get rid of for loops.

What is vectiorzation? We often find that deep learning can perform better when training on relatively large data sets, and our code can run faster than using the traditional for loops. The develop of deep learning make the vectorization become a key skill. For example, we have a parameter $w \in \mathbb{R}^{n_x}$ and the input $x \in \mathbb{R}^{n_x}$. If we use non-vectorized method, we need to put $z = 0$ firstly, and then use a for loop to implement it. However, the propability is very slow. In constract, using the Equation 9 which is vectorized can spend less time.

$$z = \underbrace{np.dot(w, x)}_{w^\top x} + b \tag{9}$$

Vectiorizing logistic regression. Supposing that we have m training samples corresponding to Equation 10

$$z^{(i)} = w^\top x^{(i)} + b \tag{10}$$
$$a^{(i)} = \sigma(z^{(i)})$$

where $x^{(1)}, x^{(2)}, \ldots, x^{(m)}$ composing the matrix $X \in \mathcal{R}^{n_x \times m}$, and the captical Z can be defined by:

$$Z = [z^{(1)} \quad z^{(2)} \quad \ldots \quad z^{(m)}] = W^\top X + [b \quad b \quad \ldots \quad b] \tag{11}$$
$$= \underbrace{[\overbrace{w^\top x^{(1)} + b}^{z^{(1)}} \quad \overbrace{w^\top x^{(2)} + b}^{z^{(2)}} \quad \ldots \quad \overbrace{w^\top x^{(m)} + b}^{z^{(m)}}]}_{1\times m} \tag{12}$$

1.9 Broadcasting in Python

In Equation 9, b is a real number originally, but it can be extended to a row vector by $1 \times m$ automatically named "broadcasting" at Python when adding a vector to this real number. Generally, if a matrix with $m \times n$ has a computation $(+, -, \times, /)$ with one-dimensional matrix like $(1, n)$ or $(m, 1)$, the one-dimensional matrix will be duplicated m or n times to become (m, n).

Notation. (1). We can use *np.linalg.norm* to get the norm through the Equation 13 cited from [1]

$$x\_norm = np.linalg.norm(x, ord = None, axis = None, Keepdims = False) \tag{13}$$

where x represents the matirx, *ord* indicates types of norms, *axis* represents types of dealing matrixs, and *keepdims* shows whether keeping the two-dimensional features of matrix. (2). *np.dot*() performs a matrix-matirx or matrix-vector multiplication. This is different from *np.multiply*() and the operator which performs an element-wise multiplication.

References

[1] Csdn qinghaohuang. https://blog.csdn.net/hqh131360239/article/details/79061535. 5

[2] Neural Networks & Deep Learning. https://mooc.study.163.com/course/2001281002#/info. 1, 2, 3, 4