Gradient Descent Example Code

In the paper, we suppose you have a basic understanding of gradient descent, however it's not a must. If you know how to code in Python, you can figure out what is gradient descent and how it works by reading the following codes and explanations.

Gradient descent[1] is a first-order optimization algorithm. To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient (or of the approximate gradient) of the function at the current point. If instead one takes steps proportional to the positive of the gradient, one approaches a local maximum of that function; the procedure is then known as gradient ascent.

1 Linear Regression with Gradient Descent

1.1 Preliminary

1.2 Show Me the Code

```python
import sys
import numpy as np

def linear_regression(x):
    true_theta = np.array([[-11.3,99.9,3]]).T
    return np.dot(x, true_theta)

x = np.array([[31,24,311], [1,103,113], [1,200,3], [11,16,2], [1,310,4], [111,3,7], [211,2,2]])
y = linear_regression(x)

alpha = 0.00001 # not too small since graident is aways a large number
max_iterator = 40
cnt = 0

theta = np.random.random((3,1)) - 0.5
xmatrix = np.full((len(y), 1), 1, dtype=int).T

while cnt < max_iterator:
    cnt = cnt + 1

    # target function is L(x, theta) = 1.0/2 * (y - np.dot(x, theta))**2
    # Here we take np.dot(x, theta) as a variable, rather than theta, which is much easier.
    neg_grad = y - np.dot(x, theta)
    theta += alpha*np.dot(x.T, neg_grad)

    #calculate the cost function
    L = 1.0/2 * (y - np.dot(x, theta))**2 # not 1, but 1.0
```

```
print 'error : %f\n'%(np.dot(xmatrix, L))
print theta

test_x1 = np.array([[1,3,7]])
print 'prediction : %f, result : %f\n'%(np.dot(test_x1, theta), linear_regression(test_x1))

test_x2 = np.array([[4,13,7]])
print 'prediction : %f, result : %f\n'%(np.dot(test_x2, theta), linear_regression(test_x2))
```

1.3 Question 1: How this code works?

The above code, we make $\delta = np.dot(x, theta)$ as a whole variable, then we get the derivative:

$$der = \frac{d(L(x, theta)}{d(\delta)} = -1 * (y - \delta) \tag{1.3.1}$$

First we should take a look at the problem why we have to calculate the derivative der. The target function is the loss function of the model, so we need to minimize it as possible as we can. It's a good way to reduce the value $abs(y - \delta)$ with the calculated derivative. We can decrease δ if $y < \delta$ and increase δ if $y > \delta$ to make the loss function L smaller.

Here we derive the negative of gradient with der in equation 1.3.1:

$$neg\_grad = -1 * der = y - \delta = \begin{cases} > 0, & y > \delta; \quad // \ \delta \ needs \ to \ increase \\ < 0, & y < \delta; \quad // \ \delta \ needs \ to \ decrease \\ = 0, & y = \delta; \quad // \ \delta \ is \ ok \end{cases} \tag{1.3.2}$$

With equation 1.3.2, we can update

$$\delta' = \delta + \lambda * neg\_grad \tag{1.3.3}$$

to reduce cost function, λ is the step size. You may ask why we have use the negative of gradient and why is gradient the direction of steepest ascent? The textbook said target function decreases fastest if one goes from the point α in the direction of the negative gradient of F at α. How can you demonstrate it?

So firstly let's recall the concept of gradient. Let $f(x_1, x_2, \ldots x_n) : \mathbb{R}^n \to \mathbb{R}$, the partial derivatives of f are the rates of change along the basis vectors of x. The partial derivative, or rate of change along e_i can be expressed like this:

$$der(f(x))_i = \frac{\partial f}{\partial x_i} = \lim_{h \to 0} \frac{f(x + h * e_i) - f(x)}{h} \qquad // \ it's \ a \ number. \tag{1.3.4}$$

Actually the gradient is the directional derivative, the gradient along e_i is:

$$grad(f(x))_i = der(f(x))_i * e_i \qquad // \ it's \ a \ vector. \tag{1.3.5}$$

So if we change x along e_i direction, we can increase or decrease target function f.

- $if \ der(f(x))_i > 0$, We can increase x along e_i direction to increase f.

- $if\ der(f(x))_i < 0$, We can decrease x along e_i direction to increase f

Finally we can draw the conclusion like this:

$$diff = \lim_{h\to 0} f(a + h * der(f(a))_i * e_i) - f(a) = \lim_{h\to 0} f(a + h * grad(f(a))_i > 0 \tag{1.3.6}$$

We will increase target function by changing x along positive of gradient, otherwise changing x along negative of gradient to reduce target function.

1.4 Question 2: How to explain equation 1.4.1?

However in the code we use equation 1.4.1 instead of 1.3.3. Here we got the problem that the gradient is of δ, how can we use for *theta* directly in the example?

To figure out this question, we suppose

$$theta' = theta + alpha * np.dot(x.T, neg\_grad) \tag{1.4.1}$$

then we calculate the new δ':

$$\begin{aligned}\delta' &= np.dot(x, theta') = np.dot(x, theta) + np.dot(x, alpha * np.dot(x.T, neg_grad))\\ &= \delta + alpha * np.dot(np.dot(x, x.T), neg\_grad)\end{aligned} \tag{1.4.2}$$

With sample x_i, it's is obvious in equation 1.4.2 that matrix $np.dot(x_i, x_i.T) > 0$, so if $neg\_grad_i > 0$, then $\delta'_i > \delta_i$, which means δ'_i increased, in other words, we can increase δ by making $neg\_grad_i > 0$ and vice versa. At this point we will find that equation 1.4.1 have the same effect with equation 1.3.3. Bingo, the gradient for δ also can be used for θ and the hypothesis proposed above is correct.

1.5 Question 3: How about calculate the derivative of *theta*?

Another question is how about calculate the derivative of *theta*, rather than $\delta = np.dot(x, theta)$. You can give it a try yourself. However you will find it's quite complicated and you have to calculate the derivative of a matrix, it's really a disaster. So this a trick you have to take in mind when dealing with gradient descent problems.

1.6 Question 4: Will the target function converge?

Then you may ask how could the target function converges to the desired minimum at the end. When you update *theta* for sample x_i, sample x_k will be affected. How can you make sure that target function or cost function will converge to a tiny number.

1.7 Question 5: Is it a global minimum?

The following question is whether it's a global minimum or just a local one.

2 Sigmod Regression with Gradient Descent

```
import sys
import numpy as np

def sigmod(x, theta):
    return 1.0/(1 + np.exp(-np.dot(x, theta)))

true_theta = np.array([[1,1,-1]]).T
x = np.array([[0,0,1], [0,1,0], [0,1,1], [1,0,0], [1,0,1], [1,1,0]])
y = sigmod(x, true_theta)

alpha = 0.01 # not too small since graident is aways less than 1
max_iterator = 4000
cnt = 0

theta = np.random.random((3,1)) - 0.5
xmatrix = np.full((len(y), 1), 1, dtype=int).T

while cnt < max_iterator:
    cnt = cnt + 1

    # target function is L(x, theta) = 1.0/2 * (y - sigmod(x, theta))**2
    # Here we take sigmod(x, theta) as a whole variable, not theta
    # Therefore d(L(x, theta))/d(sigmod(x, theta)) = -1 * (y - sigmod(x, theta))
    neg_grad = y - sigmod(x, theta)
    theta += alpha*np.dot(x.T, neg_grad)

    #calculate the cost function
    L = 1.0/2 * (y - sigmod(x, theta))**2 # not 1, but 1.0

print 'error : %f\n'%(np.dot(xmatrix, L))
print theta

test_x1 = np.array([[1,3,7]])
print 'prediction : %f, result : %f\n'%(sigmod(test_x1, theta), sigmod(test_x1, true_theta))

test_x2 = np.array([[4,13,7]])
print 'prediction : %f, result : %f\n'%(sigmod(test_x2, theta), sigmod(test_x2, true_theta))
```

3 Conclusion and Future Work

3.1 Conclusion and Recommendation

The idea of gradient descent we get in textbook is always abstract, when it comes to the real world problems, there seems to be much confusion and you may get no clues what to do. It greatly dependents on the train model and target function of task you are tackling when you are implementing gradient descent algorithm, also there are some tricks you have to keep in mind. We strongly recommend you to write your own code after reading this article. If you want to be able to create arbitrary architectures based on new academic papers or read and understand sample code for these different architectures, I think that it's a killer exercise.

3.2 Future Work

References

[1] WikiPedia. *Gradient descent*. 7 February 2016. URL: `https://en.wikipedia.org/wiki/Gradient_descent`.