Study Report

Shuo Xu

August 5, 2018

Abstract

This week I spent mainly time in finishing the CNN program without using tensorflow.

CNN Convolutional Neural Network

About CNN

Forward propagation

The forward propagation of CNN is easier to understand. There are two new things about it: convolutional layer and pooling layer. To realize the convolutional layer, we need use many filters and convolve them on the input. Each 'convolution' will give us a 2D matrix output, then we will stack these outputs to get a 3D volume.

In my view, the pooling layer also used something like filter, but we don't really need create a filter to calculate the outputs. In other words, we only used the f, the filter's size, and *stride* of a filter, so the pooling layer doesn't have parameters for backpropagation to train. Now, I have known two types of pooling layers: Max-pooling and Average-pooling.

There are two useful function:

$$n_H = \lfloor \frac{n_{H_{prev}} - f + 2 \ \times pad}{stride} \rfloor + 1$$

$$n_W = \lfloor \frac{n_{W_{prev}} - f + 2 \ \times pad}{stride} \rfloor + 1$$

In cousera video, the filter and inputs are always square. But after I used the tensorflow, I realized the reality isn't always so good. So it's necessary to calculate the n_H and n_W separately. Another important parameter is n_C, the channel number. The way to calculate it is different in these two layers. In the pooling layer $n_C = n_{C_{prev}}$, but in the convolutional layer n_C is decided by the number of filter.

Backpropagation

The backpropagation of convolutional layer and pooling Layer is easy to understand but difficult to say. The followings are python code of the backpropagations.

Convolutional layer:

```
def conv_back(dZ, cache):
    """
    (A_prev, W, b, hparameters) = cache
    """
    (A_prev, W, b, hparameters) = cache
    (f, f, n_c_prev, n_c) = np.shape(W)
    (m, n_h_prev, n_w_prev, n_c_prev) = A_prev.shape
```

```
    (m, n_h, n_w, n_c) = np.shape(dZ)
    pad = hparameters['pad']
    strides = hparameters['strides']

    dA_prev = np.zeros((m, n_h_prev, n_w_prev, n_c_prev))
    dW = np.zeros((f, f, n_c_prev, n_c))
    db = np.zeros((1, 1, 1, n_c))

    A_prev_pad = zero_pad(A_prev, pad)
    dA_prev_pad = zero_pad(dA_prev, pad)

    for i in range(m):
    a_prev_pad = A_prev_pad[i]
    da_prev_pad = dA_prev_pad[i]
    for h in range(n_h):
    for w in range(n_w):
    for c in range(n_c):

    vert_start = h * strides
    vert_end = vert_start + f
    horiz_start = w * strides
    horiz_end = horiz_start + f

    a_slice = a_prev_pad[vert_start: vert_end,
    horiz_start: horiz_end, :]

    da_prev_pad[vert_start:vert_end, horiz_start:horiz_end,:] += W[:, :, :, c] * dZ[i, h, w, c]
    dW[:, :, :, c] += a_slice * dZ[i, h, w, c]
    db[:, :, :, c] += dZ[i, h, w, c]

    dA_prev[i, :, :, :] = dA_prev_pad[i, pad:-pad, pad: -pad, :]
    return dA_prev, dW, db
```

Pooling layer:

```
def pool_back(dA, cache, mode="max"):
    """
    (A_prev, hparameters) = cache
    """
    (A_prev, hparameters) = cache
    f = hparameters["f"]
    strides = hparameters['strides']
    (m, n_h_prev, n_w_prev, n_c_prev) = np.shape(A_prev)
    (m, n_h, n_w, n_c) = np.shape(dA)

    dA_prev = np.zeros((m, n_h_prev, n_w_prev, n_c_prev))

    for i in range(m):
    a_prev = A_prev[i]
    for h in range(n_h):
    for w in range(n_w):
    for c in range(n_c):

    vert_start = h* strides
    vert_end = vert_start + f
    horiz_start = w * strides
    horiz_end = horiz_start + f

    if mode == 'max':
    a_prev_slice = a_prev[vert_start: vert_end,
    horiz_start:horiz_end, c]
    mask = a_prev_slice == np.max(a_prev_slice)
    dA_prev[i, vert_start: vert_end,
    horiz_start: horiz_end, c] += mask * dA[i, h, w, c]
    return dA_prev
```

Problems

After I finished the videos of CNN, I implement a CNN program with tensorflow. There is a tutorial of CNN program about MNIST dataset in the TensorFlow Chinese community, the parameters of the program that I wrote are from there. After I run this tensorflow program several times, I started to implement a same program without tensorflow.

In this attempt, I used the parameters and structure of the previous program. When I started writing the back-propagation part, I found that the program would run very slowly without using tensorflow. So I simplified the network structure and wrote a program with tensorflow again. I think this is simple, but when I finish the program that doesn't use TunSoFrand, I find that when the iteration is 500 times, it takes 40 minutes to run the program, but only eight pictures are calculated for each iteration. I need a better network structure, otherwise it will be difficult to compare.

The first structure is: $Conv-layer->relu->max-pooling->Conv-layer->relu->max-pooling->Full-connect->Full-connect->softmax$

Shape change:$(m,28,28,1)->(m,14,14,32)->(m,7,7,64)->(m,7*7*64)->(m,10)$

The second is: $Conv-layer->relu->max-pooling->Full-connect->softmax$

Shape change:$(m,28,28,1)->(m,14,14,32)->(m,14*14*32)->(m,10)$

Now it seems that after a few layers, the computation is still very large.And I still don't know how tensorflow's "SAME" and "VALID" algorithm are implemented.

https://github.com/Wanakiki/bug-free-broccoli/tree/master/deepin/cnn

Leetcode

Description

There are N children standing in a line. Each child is assigned a rating value. You are giving candies to these children subjected to the following requirements:

- Each child must have at least one candy.
- Children with a higher rating get more candies than their neighbors.

What is the minimum candies you must give.

Solution

According to the description of the problem, there are two key points: the number of candy for each child can't less than 1, and if the child have a higher rating than his neighbors, he should have more candies.

Now suppose we use an array to record the number of candy of everyone. We traversed the entire array two times and corrected the elements of the array according to the rules we have. The two traversal takes different orders, and we count the total number in the second traverses. In the end return the total number.

Code

C++

```
class Solution {
    public:
        int candy(vector<int>& ratings) {
            int len = ratings.size();
            vector<int> res(len,0);
            res[0] = 1;
            for(int i = 1; i < len; i++){
                    if(ratings[i] > ratings[i-1])
                    res[i] = res[i-1] + 1;
                    else
                    res[i] = 1;
                }
            int sum = res[len-1];
            for(int j = len-2; j >= 0; j--){
                    if(ratings[j] > ratings[j+1])
                    res[j] = max(res[j+1]+1, res[j]);
                    sum += res[j];
                }
            return sum;
        }
    };
```