descirption_hw2

September 13, 2018

Yutong Dai, yutongd3@illinois.edu

1 A few notes on tensor operations in numpy

The implementation is straightforward. For forward steps, first conduct convolution operations on input image, and then do element-wise non-linear transformation. After doing a linear transformation, we apply the softmax function to get the final results. For backward steps, using the chain rule to calculate the partial derivatives with respect to W, b, K and then update them.

Only thing needs to be specifically addressed here is tensor operations, which consume must of the time. To calculate the "convolution" operation defined by the lecture note, which is indeed more often referred as cross-correlation, can be done with the function `signal.correlate2d` in `scipy`. However we can not use it. So I write my own `Convolution` function.

```
In [1]: import numpy as np
        a = np.arange(9).reshape((3,3))
        b = np.array([[1,2], [3,4]])
        def Convolution(image, myfilter):
            d = image.shape[-1]
            ky, kx = myfilter.shape
            conv = np.zeros((d - ky + 1, d - kx + 1))
            for i in range(d - ky + 1):
                for j in range(d - kx + 1):
                    conv[i,j] = np.sum(np.multiply(image[i:i+ky,j:j+kx], myfilter))
            return conv
        Convolution(a, b)
```

```
Out[1]: array([[27., 37.],
               [57., 67.]])
```

To calculate $U_k = W_{k,:,:,:} \cdot H + b_k$ in a more efficient way, where $\cdot$ is the element-wise-multiplication-and-summation on tensors, we can utilize `numpy.tensordot`.

```
In [3]: W = np.arange(24).reshape((3,2,2,2))
        H = np.arange(8).reshape((2,2,2))
        np.tensordot(W, H, axes=3).reshape(3,1)
```

```
Out[3]: array([[140],
               [364],
               [588]])
```

To calculate the δ, which is a $(d - k_y + 1, d - k_x + 1, C)$ tensor in a more efficient way we need use `numpy.squeeze`, `numpy.tensordot`, and the broadcast mechanism. Note the `squeeze` step is a must to avoid dimension issues.

```
In [5]: dU = np.array([1,2,3])
        W = np.arange(24).reshape((3,2,2,2))
        np.tensordot(dU.squeeze(), W, axes=1)

Out[5]: array([[[ 64,  70],
                [ 76,  82]],

               [[ 88,  94],
                [100, 106]]])
```

Similarly, for calculating dW, we can use code below.

```
In [7]: dU = np.array([1,2])
        H = np.arange(8).reshape((2,2,2))
        np.tensordot(dU.squeeze(), H, axes=0)

Out[7]: array([[[[ 0,  1],
                 [ 2,  3]],

                [[ 4,  5],
                 [ 6,  7]]],


               [[[ 0,  2],
                 [ 4,  6]],

                [[ 8, 10],
                 [12, 14]]]])
```

2 Experiments

If I train the CNN on theMNIST dataset based on my own `Convolution` function, the results are summarized below.

- Parameters

  ```
  training sample size: [(60000, 28, 28)]
  test sample size:[(10000, 28, 28)]
  channels:[5]
  filter_size: 5 * 5 * 5
  Gaussian noise for W and K with proper normalizing constants.
  ```

- Learning Rate: Follow the lecture notes.

  Training accuracy for each epoch, where a epoch is defined as sampling a input from `x_train` for 60000 times.

```
epoch:1 | Training Accuracy:[0.9484166666666667]

epoch:2 | Training Accuracy:[0.9765]

epoch:3 | Training Accuracy:[0.9808]

epoch:4 | Training Accuracy:[0.9850166666666667]

epoch:5 | Training Accuracy:[0.9872166666666666]
```

Results:
It took `5594.425630331039` seconds. (This result is run on a server.)
Test accuracy is `[0.978]`.
Remarks:
If I use `signal.correlate2d` in `scipy` to do the convolution, for the hyper-parameter setting setting, it took me around `[443.0835828781128]` seconds to run 10 epochs locally. And I can get `[0.9796]` test accuracy.