EE-559: Mini-project II

Pegolotti Luca - Martin Matthieu

May 14, 2018

1 Objective

The objective of this project is to design a mini "deep learning framework" using only tensor operations in pytorch and the standard math library, in particular without using autograd or the neural-network modules.

2 Code structure

The framework is composed by two modules: `modules` and `criterions`.

2.1 Modules

Modules implement some of the typical building blocks of a neural network. Each of these building blocks derives from the class

```
class Module(object):
    def forward(self,*input):
        raise NotImplementedError

    def backward(self,*gradwrtoutput):
        raise NotImplementedError

    def resetGradient(self):
        raise NotImplementedError

    def updateParameters(self):
        raise NotImplementedError

    def param(self):
        return []
```

the basic structure of which (except for the methods `resetGradient` and `updateParameters`) was suggested in the description of the project.

3 Test case

The structure of the test code, implementing a network with two input units, two output units, three hidden layers of 25 units is:

- Generate 1000 training sample, and 1000 testing points

- Normalize them with a zero mean and unit std

- Built a three hidden layer with linear neural network, and ReLU after each linear module "SimpleNet"

- Train the neural network using the 1000 training sample, for 1000 epochs and a constant learning rate $= 1e - 2$.

- Plot the training error and the testing error while training the network, and verify these results with the framework PyTorch.

The parameters of the sample length is hidden in the *mean* function, and we had to modify the eta in the final code: $eta = eta/nsample$.

```
class LossMSE(object):
    def function(self,output,expected):
        return torch.mean(torch.pow(expected - output,2))

    def grad(self,output,expected):
        return -2 * (expected - output)
```

The sequential class work as follow:

```
class Sequential(Module):
    def __init__(self,criterion):


    def registerModules(self,*modules):


    def checkIfModulesAreRegistered(self):


    def resetGradient(self):


    def updateParameters(self,eta,nsamples):


    def backward(self,*gradwrtoutput):


    def backwardPass(self, output, expected):
```

where the $registerModules$ needs to be called when we define a new network, in order to store the modules in a list. We will use the list ordered when we will call the methods $forward$, $backward$, and $update_Parameters$

References