Homework Assignment 5

Chaoji Zuo chaoji.zuo@rutgers.edu
CZ296 / 190003416

Problem (Prune a LeNet-5). In this problem, you are asked to train and test a neural network for LeNet-5 on MNIST dataset.

I tried different compression ration, and their performances are pretty great. The difference of accuracy is very small.

```
Test set: Average loss: 0.0388, Accuracy: 9876/10000 (99%)

the sparsity of model is 0.500
```

Figure 1: Compression Ratio 0.5

```
Test set: Average loss: 0.0404, Accuracy: 9869/10000 (99%)

the sparsity of model is 0.250
```

Figure 2: Compression Ratio 0.25

Core code here: I attach the whole code files in attachment.

```
1
2  class PruningWeight(object):
3      def __init__(self, ratio=0.0):
4          self.MaskList = []
5          self.threshold = []
6          self.ratio = ratio
7
8      def Init(self, model):
9          for m in model.modules():
10             if isinstance(m, nn.Linear) or isinstance(m, nn.Conv2d):
11                 with torch.no_grad():
12                     m.weight.copy_(self._SetUpPruning(m.weight, self.ratio))
13
14     def _SetUpPruning(self, weight, ratio):
15         _threshold = self._FindMidValue(weight, ratio)
16         sparse_weight, _Mask = self._InitMask(weight, _threshold)
17         self.threshold.append(_threshold)
18         self.MaskList.append(_Mask)
```

```
        return sparse_weight

    def _FindMidValue(self, weight, ratio):
        flatten_weight = torch.flatten(torch.abs(weight))
        print(flatten_weight.shape)
        sorted, _ = torch.sort(flatten_weight)
        index = int(ratio*flatten_weight.size()[0])
        threshold = sorted[index]
        return threshold

    def _InitMask(self, w, threshold):
        mask = torch.abs(w).ge(threshold).type(dtype=torch.float32)
        w[torch.abs(w)<threshold] =0.0
        return w, mask

    def RecoverSparse(self, model):
        _idx = 0
        for m in model.modules():
            if isinstance(m, nn.Linear) or isinstance(m, nn.Conv2d):
                with torch.no_grad():
                    m.weight.copy_(m.weight * self.MaskList[_idx])
                _idx += 1

    def TestSparse(self, model):
        zero_cnt = 0
        all_cnt = 0
        for m in model.modules():
            if isinstance(m, nn.Linear) or isinstance(m, nn.Conv2d):
                w = m.weight
                zero_cnt += torch.sum((w == 0).int()).item()
                all_cnt += w.nelement()
        print ('the sparsity of model is %.3f' % (1.0*zero_cnt/all_cnt))
```