MDS 5122 Assignment 1

Guyuan Xu 224040074

A. Build a Neural Network Using PyTorch

1. reproducing example NeuralNetwork of example code A.

check notebook

2. Factors to improve accuracy

In my experiment, I tried to use 1) Simple NN (which is the baseline model of example code A) and NN with more complex architecture (vgg16 and ResNet18/50); 2) adding BatchNorm layer and dropout layer; 3) different optimziers SGD and Adam with popular *lr* or momentum params settings; 4) with/without data augmentation.

We mainly use the model of example code-A as our baseline model to examine how the above factors might affect accuracy. And due to computing resource limitation, i did not conduct strict experiment with strict variable controls. To gain insight is what we want.

- BaseLine Model : C3L3 (3 Conv layers + 3 Linear layers) on CIFAR-10 dataset
We first start with baseline model in example-code A, the steps of experiment are: 1)compare optimziers with all other settings the same, then pick the best optimziers (in terms of accuracy) for next step; 2) use the best optimizer and data augmentation to see if data augmentation could bring up accuracy in test set; 3) use best optimizer + data augmentation(note that data augmentation might not be useful, shall be given up if so) + BatchNorm Layer + throw away dropout layer in convolution layer to see whether we can have better accuracy on test set.

```
# our network architecture:
net = nn.Sequential(
    nn.Conv2d(3, 128, 3, padding=1), nn.ReLU(inplace=True), nn.MaxPool2d(2), nn.Dropout(0.3),
    nn.Conv2d(128, 256, 3, padding=1), nn.ReLU(inplace=True), nn.MaxPool2d(2), nn.Dropout(0.3),
    nn.Conv2d(256, 512, 3, padding=1), nn.ReLU(inplace=True),
    nn.Conv2d(512, 512, 3, padding=1), nn.ReLU(inplace=True),
    nn.Conv2d(512, 256, 3, padding=1), nn.ReLU(inplace=True), nn.MaxPool2d(2), nn.Dropout(0.3),
    nn.Flatten(),
    nn.Linear(256 * 4 * 4, 512), nn.ReLU(inplace=True), nn.Dropout(0.5),
    nn.Linear(512, 256), nn.ReLU(inplace=True), nn.Dropout(0.5),
    nn.Linear(256, 128), nn.ReLU(inplace=True), nn.Dropout(0.5),
    nn.Linear(128, 10),
)
# Params count: 3*128*3*3+128*256*3*3+256*512*3*2*3+512**2*3*3+256*16*512+512*256+256*128+1280=7.28M

# Using default data transformation in example code-A
```

```
transformation = dict()
for data_type in ("train", "test"):
  is_train = data_type=="train"
  transformation[data_type] = tv_transforms.Compose((
    [
      tv_transforms.RandomRotation(degrees=15),
      tv_transforms.RandomHorizontalFlip(),
      tv_transforms.RandomAffine(degrees=0, translate=(0.1, 0.1)),
    ] if is_train else []) +
  [
      tv_transforms.ToTensor(),
      tv_transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5]),
  ])
```

This model is exactly the same model in example code A, we then try to reproduce the accuracy and comapre accuracy with different settings:

Model	C3L3		
Adam	√		
SGD		√	
Data Aug			
dropout(conv)	√	√	
BatchNorm			
Accuracy	85.26	84.93	

Note: bacthsize = 32, Epoch num = 150, dropout(conv) indicate whether we use dropout layer in convlution layer. - Conclusion:

- Data augmentation can greatly improve accuracy
- choosing SGD or Adam will not significantly affect accuracy, they just converge to almost the same accuracy at different speeds (in terms of epochs).
- BatchNorm

MNIST dataset

We use simple CNN together with our best params on MNIST dataset. Check notebook codes.

Interesting findings and reflections

1) visualize data after data augmentation 2) we have i learnt.

B. Build a Neural Network From Scratch

做完 1 的试验后获取 best 参数，接着用 dezero 堆叠一个相同结构。

Reflections

From BackwardPropagation to AutoGrad: In example code part-B we can learn how forward pass and backward pass work together in a neural network, but implementing backward pass is complicated and sometimes impossible (especially in complicated computation). Meanwhile we know that all computation in deep learning are basically consist of few basic functions like exponents, add, subtract, multiplication etc, so we can utilizie chain rule to calculate gradient no matter how complicated a computation might be. This is how AutoGrad comes into play, and this is the core of all deep learning frameworks, including pytorch.

Understand by creating: During the process of building a framework, i learn extensively and experience moments of revelation like ”ah, this is how a neural network works!” or ”so this algorithm can be implemented this way!” These insights are unattainable through merely using existing tools like pytorch. Some concepts can only be understood by creating, some truths can only be seen by building.

For instance, some may view deep learning frameworks as mere libraries pieced together with layers and functions. In reality, frameworks go far beyond that. A framework can be considered a programming language—specifically, one with automatic differentiation capabilities (recently termed a ”differentiable programming language”).