MDS 5122 Assignment 1

Guyuan Xu 224040074

A. Build a Neural Network Using PyTorch

1. reproducing example NeuralNetwork of example code A.

Results are summaried in subsection 2.2 together with other settings, codes check notebook

2. Factors to improve accuracy

In my experiment, I tried to use 1) Simple NN structure(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.

Experiments Setup

2.1 Baseline Convolutional Neural Network (C5L3)

- architecture: 5 conv layers, each followed by a dropout layer (rate=0.3); 3 linear layers, each followed by a dropout layer (rate=0.5).
- optimizer: Adam with a learning rate of 3e-4 and weight decay of 1e-6.
- dataset: Trained on the CIFAR-10 dataset. The training set has dimensions (60,000, 3, 32, 32), and the test set has dimensions (10,000, 3, 32, 32).
- Training settings: Trained for 150 epochs with a batch size of 64. Model accuracy was evaluated on the test set.

```
# network(C5L3) 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]),
    ])
```

2.2 Additional Modifications Beyond the Baseline(C5L3):

- Extra Data Augmentation: Aside from default techniques in baselline model including random flipping, random rotation, random affine transformation, we add random brightness adjustments, and saturation modifications.
- Optimizer Variant: Replaced Adam with SGD (learning rate=1e-3).
- Architectural Varaint: Removed dropout layers after convolutional layers (retained dropout for linear layers), or adding BatchNorm layers after convolutional layers.

Results are summarized in the folloing table

Model	C5L3					
Adam	√		√	√	conv	
SGD		√				
ExtraDataAug			√	√		
dropout	all	all	linear	none	linear	
BatchNorm			conv	conv	conv	
Accuracy	85.26	84.93	89.78	89.52	88.54	

# Note: bacthsize = 64, 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) what have i learnt. 3) due to curiosity，可以用 batchsize 32 以及 256 epoch 对 best model 进行验证，看看 accu 能否上 90

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").