AI_Assgn_2_Q4

November 10, 2021

1 Import the packages

[ ]:
```
import torch
import torchvision
from torch import nn, optim
from torchsummary import summary
```

2 Declare variables for the CNN

- **Epoch** is the number of passes of the entire training dataset through the neural network. A pair of forward and backward propagation indicates a single pass.
- **Batch Size** is the number of samples to work through before updating the weights and biases associated with the model.
- **Learning Rate** controls how much to change the model parameters in response to the prediction error each time the model weights are updated.

[ ]:
```
batch_size = 32
epoch = 30
learning_rate = 0.01
```

3 Load the training set and validation set using Dataset and DataLoader

[ ]:
```
trans = torchvision.transforms.ToTensor()
train_data = torch.utils.data.DataLoader(
  torchvision.datasets.MNIST(
    'mnist_data', train=True, download=True, transform=trans
    ), batch_size=batch_size
    )
val_data = torch.utils.data.DataLoader(
  torchvision.datasets.MNIST(
    'mnist_data', train=False, download=True, transform=trans
    ), batch_size=batch_size)
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to
mnist_data/MNIST/raw/train-images-idx3-ubyte.gz

  0%|          | 0/9912422 [00:00<?, ?it/s]


Extracting mnist_data/MNIST/raw/train-images-idx3-ubyte.gz to
mnist_data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to
mnist_data/MNIST/raw/train-labels-idx1-ubyte.gz

  0%|          | 0/28881 [00:00<?, ?it/s]


Extracting mnist_data/MNIST/raw/train-labels-idx1-ubyte.gz to
mnist_data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to
mnist_data/MNIST/raw/t10k-images-idx3-ubyte.gz

  0%|          | 0/1648877 [00:00<?, ?it/s]


Extracting mnist_data/MNIST/raw/t10k-images-idx3-ubyte.gz to
mnist_data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to
mnist_data/MNIST/raw/t10k-labels-idx1-ubyte.gz

  0%|          | 0/4542 [00:00<?, ?it/s]


Extracting mnist_data/MNIST/raw/t10k-labels-idx1-ubyte.gz to
mnist_data/MNIST/raw


/usr/local/lib/python3.7/dist-packages/torchvision/datasets/mnist.py:498:
UserWarning: The given NumPy array is not writeable, and PyTorch does not
support non-writeable tensors. This means you can write to the underlying
(supposedly non-writeable) NumPy array using the tensor. You may want to copy
the array to protect its data or make it writeable before converting it to a
tensor. This type of warning will be suppressed for the rest of this program.
(Triggered internally at  /pytorch/torch/csrc/utils/tensor_numpy.cpp:180.)
  return torch.from_numpy(parsed.astype(m[2], copy=False)).view(*s)
```

4 Define the ANN without convolutional layers for image classification

[ ]:
```
class ANNet(nn.Module):
  def __init__(self):
    super(ANNet, self).__init__()
    self.relu = nn.ReLU()
    self.linear1 = nn.Linear(28*28, 500)
    self.linear2 = nn.Linear(500, 100)
    self.linear3 = nn.Linear(100, 10)
  def forward(self, x):
    x = x.view(x.shape[0], -1)
    x = self.relu(self.linear1(x))
    x = self.relu(self.linear2(x))
    x = self.linear3(x)
    return x
```

5 Define a function for validating the model

[ ]:
```
def validate(model, data):
  total = 0
  correct = 0
  for i, (images, labels) in enumerate(data):
    images = images.cuda()
    labels = labels.cuda()
    y_pred = model(images)
    value, pred = torch.max(y_pred, 1)
    total += y_pred.size(0)
    correct += torch.sum(pred == labels)
  return correct * 100 / total
```

6 Initialize the neural network and optimizer

[ ]:
```
ann = ANNet().cuda()
optimizer = optim.Adam(ann.parameters(), lr=learning_rate)
cross_entropy = nn.CrossEntropyLoss()
```

7 Print the Model Summary

[ ]:
```
summary(ann, (1, 224, 224))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
```

```
================================================================
            Linear-1                  [-1, 500]         392,500
              ReLU-2                  [-1, 500]               0
            Linear-3                  [-1, 100]          50,100
              ReLU-4                  [-1, 100]               0
            Linear-5                   [-1, 10]           1,010
================================================================
Total params: 443,610
Trainable params: 443,610
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.19
Forward/backward pass size (MB): 0.01
Params size (MB): 1.69
Estimated Total Size (MB): 1.89
----------------------------------------------------------------
```

8 Display the validation accuracy on each epoch

[ ]:
```
for n in range(epoch):
  for i, (images, labels) in enumerate(train_data):
    images = images.cuda()
    labels = labels.cuda()
    optimizer.zero_grad()
    prediction = ann(images)
    loss = cross_entropy(prediction, labels)
    loss.backward()
    optimizer.step()
  accuracy = float(validate(ann, val_data))
  print("Epoch:", n+1, "Loss: ", float(loss.data), "Accuracy:", accuracy)
```

```
Epoch: 1 Loss:  0.0039152661338448524 Accuracy: 93.5199966430664
Epoch: 2 Loss:  0.0030660051852464676 Accuracy: 93.5
Epoch: 3 Loss:  0.008405933156609535 Accuracy: 93.75
Epoch: 4 Loss:  0.040073223412036896 Accuracy: 94.37999725341797
Epoch: 5 Loss:  0.015041790902614594 Accuracy: 94.68999481201172
Epoch: 6 Loss:  0.008267774246633053 Accuracy: 96.0999984741211
Epoch: 7 Loss:  0.010446286760270596 Accuracy: 96.30999755859375
Epoch: 8 Loss:  0.000716851616743952 Accuracy: 95.95999908447266
Epoch: 9 Loss:  0.09016045182943344 Accuracy: 95.25999450683594
Epoch: 10 Loss:  0.003279729513451457 Accuracy: 96.72999572753906
Epoch: 11 Loss:  0.003906412981450558 Accuracy: 96.68999481201172
Epoch: 12 Loss:  0.007484755013138056 Accuracy: 95.83999633789062
Epoch: 13 Loss:  0.0004099629877600819 Accuracy: 96.32999420166016
Epoch: 14 Loss:  0.017722729593515396 Accuracy: 96.40999603271484
Epoch: 15 Loss:  0.043866898864507675 Accuracy: 96.56999969482422
Epoch: 16 Loss:  0.0010566012933850288 Accuracy: 96.80999755859375
```

```
Epoch: 17 Loss:  0.00030694992165081203 Accuracy: 96.72999572753906
Epoch: 18 Loss:  0.01591970957815647 Accuracy: 96.58999633789062
Epoch: 19 Loss:  0.01136217825114727 Accuracy: 96.29000091552734
Epoch: 20 Loss:  0.03559637442231178 Accuracy: 96.29999542236328
Epoch: 21 Loss:  1.262848059013777e-06 Accuracy: 96.62999725341797
Epoch: 22 Loss:  0.0016550994478166103 Accuracy: 96.68000030517578
Epoch: 23 Loss:  0.0001836229785112664 Accuracy: 96.5
Epoch: 24 Loss:  0.0042234198190271854 Accuracy: 96.04000091552734
Epoch: 25 Loss:  0.27225372195243835 Accuracy: 96.72000122070312
Epoch: 26 Loss:  0.008386979810893536 Accuracy: 96.79999542236328
Epoch: 27 Loss:  2.2351693473865453e-07 Accuracy: 96.37999725341797
Epoch: 28 Loss:  0.00573017867282033 Accuracy: 96.5999984741211
Epoch: 29 Loss:  0.0838499516248703 Accuracy: 96.2699966430664
Epoch: 30 Loss:  2.011650508393359e-07 Accuracy: 96.47999572753906
```