AI_Assgn_2_Q2

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 CNN with Pooling layers for image classification

[ ]:
```python
class ConvNet(nn.Module):
  def __init__(self):
    super(ConvNet, self).__init__()
    self.conv1 = nn.Conv2d(in_channels=1, out_channels=3, kernel_size=3,␣
↪stride=1, padding=1)
    self.conv2 = nn.Conv2d(in_channels=3, out_channels=6, kernel_size=3,␣
↪stride=1)
    self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
    self.tanh = nn.Tanh()
    self.linear1 = nn.Linear(6*6*6, 10)
  def forward(self, x):
    x = self.tanh(self.conv1(x))
    x = self.pool(x)
    x = self.tanh(self.conv2(x))
    x = self.pool(x)
    x = x.view(x.shape[0], -1)
    x = self.linear1(x)
    return x
```

5 Define a function for validating the model

[ ]:
```python
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

[ ]:
```python
convnet = ConvNet().cuda()
optimizer = optim.Adam(convnet.parameters(), lr=learning_rate)
cross_entropy = nn.CrossEntropyLoss()
```

7 Print the Model Summary

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

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 3, 224, 224]              30
              Tanh-2          [-1, 3, 224, 224]               0
         MaxPool2d-3          [-1, 3, 112, 112]               0
            Conv2d-4          [-1, 6, 110, 110]             168
              Tanh-5          [-1, 6, 110, 110]               0
         MaxPool2d-6            [-1, 6, 55, 55]               0
            Linear-7                   [-1, 10]           2,170
================================================================
Total params: 2,368
Trainable params: 2,368
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.19
Forward/backward pass size (MB): 3.83
Params size (MB): 0.01
Estimated Total Size (MB): 4.03
----------------------------------------------------------------
```

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 = convnet(images)
    loss = cross_entropy(prediction, labels)
    loss.backward()
    optimizer.step()
  accuracy = float(validate(convnet, val_data))
  print("Epoch:", n+1, "Loss: ", float(loss.data), "Accuracy:", accuracy)
```

```
Epoch: 1 Loss:  0.05365052446722984 Accuracy: 95.25
Epoch: 2 Loss:  0.021441258490085602 Accuracy: 96.19999694824219
Epoch: 3 Loss:  0.05124111473560333 Accuracy: 96.32999420166016
Epoch: 4 Loss:  0.023698387667536736 Accuracy: 96.05999755859375
Epoch: 5 Loss:  0.005422498565167189 Accuracy: 96.54999542236328
Epoch: 6 Loss:  0.021976687014102936 Accuracy: 96.45999908447266
Epoch: 7 Loss:  0.0030060645658522844 Accuracy: 96.2699966430664
```

```
Epoch: 8 Loss:  0.004817866254597902 Accuracy: 96.5199966430664
Epoch: 9 Loss:  0.013756404630839825 Accuracy: 96.47000122070312
Epoch: 10 Loss:  0.004688573535531759 Accuracy: 96.72999572753906
Epoch: 11 Loss:  0.061570823192596436 Accuracy: 96.36000061035156
Epoch: 12 Loss:  0.004826270043849945 Accuracy: 95.50999450683594
Epoch: 13 Loss:  0.021710053086280823 Accuracy: 96.05999755859375
Epoch: 14 Loss:  0.0696922317147255 Accuracy: 96.1199951171875
Epoch: 15 Loss:  0.006976943463087082 Accuracy: 96.19999694824219
Epoch: 16 Loss:  0.12426336854696274 Accuracy: 96.07999420166016
Epoch: 17 Loss:  0.05064542964100838 Accuracy: 96.04000091552734
Epoch: 18 Loss:  0.08069183677434921 Accuracy: 95.91999816894531
Epoch: 19 Loss:  0.017632251605391502 Accuracy: 96.68999481201172
Epoch: 20 Loss:  0.0728498324751854 Accuracy: 96.0199966430664
Epoch: 21 Loss:  0.04495245963335037 Accuracy: 96.54000091552734
Epoch: 22 Loss:  0.14952804148197174 Accuracy: 96.88999938964844
Epoch: 23 Loss:  0.06291703879833221 Accuracy: 96.5999984741211
Epoch: 24 Loss:  0.02389199659228325 Accuracy: 96.72000122070312
Epoch: 25 Loss:  0.025268472731113434 Accuracy: 96.87999725341797
Epoch: 26 Loss:  0.18226934969425201 Accuracy: 96.97999572753906
Epoch: 27 Loss:  0.013176980428397655 Accuracy: 96.75999450683594
Epoch: 28 Loss:  0.03337092325091362 Accuracy: 96.7699966430664
Epoch: 29 Loss:  0.07979714870452881 Accuracy: 96.86000061035156
Epoch: 30 Loss:  0.02474566549062729 Accuracy: 96.89999389648438
```