AI_Assgn_2_Q1

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 for image classification

[ ]:
```
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)
    self.conv2 = nn.Conv2d(in_channels=3, out_channels=6, kernel_size=3,␣
 ↪stride=1)
    self.tanh = nn.Tanh()
    self.linear1 = nn.Linear(3456, 10)
  def forward(self, x):
    x = self.tanh(self.conv1(x))
    x = self.tanh(self.conv2(x))
    x = x.view(x.shape[0], -1)
    x = self.linear1(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

[ ]:
```
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, 222, 222]              30
              Tanh-2          [-1, 3, 222, 222]               0
            Conv2d-3          [-1, 6, 220, 220]             168
              Tanh-4          [-1, 6, 220, 220]               0
            Linear-5                    [-1, 10]          34,570
================================================================
Total params: 34,768
Trainable params: 34,768
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.19
Forward/backward pass size (MB): 6.69
Params size (MB): 0.13
Estimated Total Size (MB): 7.01
----------------------------------------------------------------
```

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:  0.21218594908714294 Accuracy: 87.68999481201172
Epoch: 2 Loss:  0.15007132291793823 Accuracy: 83.15999603271484
Epoch: 3 Loss:  0.20684774219989777 Accuracy: 86.5
Epoch: 4 Loss:  0.07335351407527924 Accuracy: 87.16999816894531
Epoch: 5 Loss:  0.18528632819652557 Accuracy: 83.43000030517578
Epoch: 6 Loss:  0.22310829162597656 Accuracy: 83.65999603271484
Epoch: 7 Loss:  0.12215633690357208 Accuracy: 87.08999633789062
Epoch: 8 Loss:  0.13716231286525726 Accuracy: 83.90999603271484
Epoch: 9 Loss:  0.13552477955818176 Accuracy: 84.41999816894531
Epoch: 10 Loss:  0.1604619026184082 Accuracy: 84.18999481201172
Epoch: 11 Loss:  0.2963145971298218 Accuracy: 86.19999694824219
Epoch: 12 Loss:  0.12405643612146378 Accuracy: 86.25999450683594
Epoch: 13 Loss:  0.038135409355163574 Accuracy: 84.80999755859375
Epoch: 14 Loss:  0.06628179550170898 Accuracy: 83.58999633789062
Epoch: 15 Loss:  0.44077709317207336 Accuracy: 85.54000091552734
Epoch: 16 Loss:  0.08903875946998596 Accuracy: 84.32999420166016
Epoch: 17 Loss:  0.10942036658525467 Accuracy: 85.45999908447266
```

```
Epoch: 18 Loss:  0.08876486867666245 Accuracy: 84.68000030517578
Epoch: 19 Loss:  0.0831499993801117 Accuracy: 83.75
Epoch: 20 Loss:  0.029688647016882896 Accuracy: 86.0
Epoch: 21 Loss:  0.0964929386973381 Accuracy: 85.38999938964844
Epoch: 22 Loss:  0.06852761656045914 Accuracy: 85.16999816894531
Epoch: 23 Loss:  0.05163497105240822 Accuracy: 85.15999603271484
Epoch: 24 Loss:  0.04711154103279114 Accuracy: 86.72000122070312
Epoch: 25 Loss:  0.0784594789147377 Accuracy: 86.65999603271484
Epoch: 26 Loss:  0.19556772708892822 Accuracy: 85.07999420166016
Epoch: 27 Loss:  0.034340690821409225 Accuracy: 84.61000061035156
Epoch: 28 Loss:  0.16018325090408325 Accuracy: 84.15999603271484
Epoch: 29 Loss:  0.05336381122469902 Accuracy: 83.63999938964844
Epoch: 30 Loss:  0.12666332721710205 Accuracy: 87.04999542236328
```