A simple CNN from scratch for MNIST dataset

October 27, 2022

1 A simple CNN from scratch for MNIST dataset

Here We construct a simple CNN for the MNIST dataset.

Firstly, we set our hyperparameters.

Second, we download the MNIST data from torchvision and prepare the dataloaders.

Thirdly, we design a simple CNN that has three conv layers.

Finally, we train our simple CNN and check the accuracy.

1.1 1.hyper parameters

[3]:
```
# hyper parameters
in_channels = 1
num_classes = 10
bs = 32
learning_rate = 0.0001
num_epochs = 10
```

1.2 2.dataset & dataloader

[4]:
```
# dataset & dataloader
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import DataLoader

mnist_train_data = torchvision.datasets.MNIST(root="mnist/",download=True,␣
 ↪train=True, transform=transforms.ToTensor())
mnist_test_data = torchvision.datasets.MNIST(root="mnist/",download=True,␣
 ↪train=False, transform=transforms.ToTensor())

train_data_loader = DataLoader(dataset=mnist_train_data, batch_size=bs,␣
 ↪shuffle=True)
test_data_loader = DataLoader(dataset=mnist_test_data, batch_size=bs,␣
 ↪shuffle=True)
```

```
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/MNIST/raw/train-images-idx3-ubyte.gz

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

Extracting mnist/MNIST/raw/train-images-idx3-ubyte.gz to mnist/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/MNIST/raw/train-labels-idx1-ubyte.gz

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

Extracting mnist/MNIST/raw/train-labels-idx1-ubyte.gz to mnist/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/MNIST/raw/t10k-images-idx3-ubyte.gz

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

Extracting mnist/MNIST/raw/t10k-images-idx3-ubyte.gz to mnist/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/MNIST/raw/t10k-labels-idx1-ubyte.gz

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

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

1.3 3.a simple CNN with 3 conv layers

[5]:
```
# simple CNN
from torch import nn
import torch.nn.functional as F

class sCNN(nn.Module):
    def __init__(self, in_channels=1, num_classes=10):
        super(sCNN,self).__init__()
        self.conv1 = nn.Conv2d(in_channels=in_channels, out_channels=32,␣
 ↪kernel_size=(3,3), stride=(1,1), padding=(1,1))
        self.pool = nn.MaxPool2d(kernel_size=(2,2), stride=(2,2))
        self.conv2 = nn.Conv2d(in_channels=32, out_channels=64,␣
 ↪kernel_size=(3,3), stride=(1,1), padding=(1,1))
        self.conv3 = nn.Conv2d(in_channels=64, out_channels=128,␣
 ↪kernel_size=(3,3), stride=(1,1), padding=(1,1))
        self.fc1 = nn.Linear(1152, num_classes)

    def forward(self, x):
        x = F.relu(self.conv1(x))
```

```
        x = self.pool(x)
        x = F.relu(self.conv2(x))
        x = self.pool(x)
        x = F.relu(self.conv3(x))
        x = self.pool(x)
        x = x.reshape(x.shape[0], -1)
        x = self.fc1(x)
        return x
```

1.4 3.5.set device as cuda

[6]:
```
# set device
import torch
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = sCNN(in_channels=in_channels, num_classes=num_classes).to(device)
model
```

[6]:
```
sCNN(
  (conv1): Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1,
ceil_mode=False)
  (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (fc1): Linear(in_features=1152, out_features=10, bias=True)
)
```

1.5 3.5.loss & optimizer

[7]:
```
# loss & optimizer
from torch import optim
loss = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)
```

1.6 4.train the model

[8]:
```
# train the model
from tqdm import tqdm
for epoch in range(num_epochs):
    for batch_idx, (data, targets) in enumerate(tqdm(train_data_loader)):

        data = data.to(device=device)
        targets = targets.to(device=device)

        scores = model(data)
        los = loss(scores, targets)

        optimizer.zero_grad()
```

```
        los.backward()

        optimizer.step()
```

```
100%|                    | 1875/1875 [00:06<00:00, 297.71it/s]
100%|                    | 1875/1875 [00:05<00:00, 355.59it/s]
100%|                    | 1875/1875 [00:05<00:00, 367.66it/s]
100%|                    | 1875/1875 [00:05<00:00, 366.91it/s]
100%|                    | 1875/1875 [00:05<00:00, 358.38it/s]
100%|                    | 1875/1875 [00:05<00:00, 365.28it/s]
100%|                    | 1875/1875 [00:05<00:00, 359.37it/s]
100%|                    | 1875/1875 [00:05<00:00, 361.70it/s]
100%|                    | 1875/1875 [00:05<00:00, 362.68it/s]
100%|                    | 1875/1875 [00:05<00:00, 366.06it/s]
```

1.7 4.5.accuracy

[10]:
```
# accuracy
import torch
def check_accuracy(loader, model):
    num_correct = 0
    num_samples = 0
    model.eval()

    with torch.no_grad():
        for x, y in loader:
            x = x.to(device=device)
            y = y.to(device=device)

            scores = model(x)
            _, predictions = scores.max(1)
            num_correct += (predictions == y).sum()
            num_samples += predictions.size(0)
    model.train()
    return num_correct/num_samples
```

1.8 4.5.print the accuracy

[11]:
```
# check the accuracy
print(f"Accuracy on training set: {check_accuracy(train_data_loader, model)*100:
 ↪.2f}")
print(f"Accuracy on test set: {check_accuracy(test_data_loader, model)*100:.
 ↪2f}")
```

```
Accuracy on training set: 99.25
Accuracy on test set: 98.94
```

[ ]:
```
```