federated-learning

December 20, 2023

0.1 Federated Learning

[15]:
```
import torch
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
import torch.nn as nn
import torch.optim as optim
import time
from torchvision.models import resnet50, ResNet50_Weights
```

[9]:
```
dataset_paths = ['../Dataset/5_percent_data', '../Dataset/10_percent_data', '../
 ↪Dataset/15_percent_data', '../Dataset/20_percent_data', '../Dataset/
 ↪25_percent_data']

train_transforms = transforms.Compose([
        transforms.Resize(64),
        transforms.ToTensor()
    ])

dataset_5 = datasets.ImageFolder(root=dataset_paths[0],␣
 ↪transform=train_transforms)
dataset_10 = datasets.ImageFolder(root=dataset_paths[1],␣
 ↪transform=train_transforms)
dataset_15 = datasets.ImageFolder(root=dataset_paths[2],␣
 ↪transform=train_transforms)
dataset_20 = datasets.ImageFolder(root=dataset_paths[3],␣
 ↪transform=train_transforms)
dataset_25 = datasets.ImageFolder(root=dataset_paths[4],␣
 ↪transform=train_transforms)

num_classes = 6
```

[10]:
```
def train_model(global_model_name, train_dataset, output_model_name):
    # Load the model architecture
    model = resnet50(weights=None)
    model.eval()
```

```
    # Replace the final fully connected layer for transfer learning with the␣
↪same num_classes
    num_ftrs = model.fc.in_features
    num_classes = 6
    model.fc = nn.Linear(num_ftrs, num_classes)

    # Load the trained weights from the saved .pth file
    model.load_state_dict(torch.load(global_model_name))
    model.eval()

    # Define the loss function and optimizer
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.SGD(model.fc.parameters(), lr=0.001, momentum=0.9)

    # Move the model to GPU if available
    device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
    model = model.to(device)

    train_dataloader = torch.utils.data.DataLoader(train_dataset,␣
↪batch_size=32, shuffle=True)
    # Train the model
    start_time = time.time()  # Record the end time of the epoch
    num_epochs = 20  # You can change this
    for epoch in range(num_epochs):
        model.train()
        running_loss = 0.0

        for inputs, labels in train_dataloader:
            inputs = inputs.to(device)
            labels = labels.to(device)

            optimizer.zero_grad()
            outputs = model(inputs)

            loss = criterion(outputs, labels)

            loss.backward()
            optimizer.step()

            running_loss += loss.item() * inputs.size(0)

        epoch_loss = running_loss / len(train_dataset)
        print(f'Epoch {epoch+1}/{num_epochs} | Loss: {epoch_loss:.4f}')
        if epoch_loss <= 0.01:
            print(" Early Stopping ")
            break
    end_time = time.time()  # Record the end time of the epoch
```

```
    print(" Training Time ", end_time - start_time)

    # Save the trained model
    torch.save(model.state_dict(), output_model_name)
    return model
```

[11]:
```
def test_model(data_dir, model_name):
    import torch
    from torchvision import datasets, transforms
    from torch.utils.data import DataLoader
    import torch.nn as nn

    # Define paths to your test dataset folder
    test_data_dir = data_dir  # Update with your test dataset path

    # Define transformations for testing (similar to training)
    test_transforms = transforms.Compose([
        transforms.Resize(64),
        transforms.ToTensor()
    ])

    # Load the test dataset using ImageFolder with the defined transformations
    test_dataset = datasets.ImageFolder(root=test_data_dir,␣
 ↪transform=test_transforms)

    # Define the test dataloader
    test_dataloader = DataLoader(test_dataset, batch_size=32, shuffle=False,␣
 ↪num_workers=4)

    # Load the model architecture
    model = resnet50(weights=None)
    model.eval()

    # Replace the final fully connected layer for transfer learning with the␣
 ↪same num_classes
    num_ftrs = model.fc.in_features
    num_classes = 6
    model.fc = nn.Linear(num_ftrs, num_classes)

    # Load the trained weights from the saved .pth file
    model.load_state_dict(torch.load(model_name))
    model.eval()

    # Move the model to GPU if available
    device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
    model = model.to(device)
```

```
    # Evaluate the model on the test dataset for both top-1 and top-3 accuracy
    correct_top1 = 0
    correct_top3 = 0

    total = 0
    with torch.no_grad():
        for images, labels in test_dataloader:
            images = images.to(device)
            labels = labels.to(device)

            outputs = model(images)
            _, preds = torch.topk(outputs, 3, dim=1)  # Get top-3 predictions
            total += labels.size(0)
            for i in range(labels.size(0)):
                if labels[i] == preds[i, 0]:  # Check top-1 accuracy
                    correct_top1 += 1
                if labels[i] in preds[i]:  # Check top-3 accuracy
                    correct_top3 += 1

    top1_accuracy = 100 * correct_top1 / total
    top3_accuracy = 100 * correct_top3 / total
    print(f'Top-1 Accuracy on the {data_dir} dataset: {top1_accuracy:.2f}%')
    print(f'Top-3 Accuracy on the {data_dir} dataset: {top3_accuracy:.2f}%')
```

[12]:
```
def update_global_model(model_5, model_10, model_15, model_20, model_25,␣
↪global_model, iteration):
    state_dict_5 = model_5.state_dict()
    state_dict_10 = model_10.state_dict()
    state_dict_15 = model_15.state_dict()
    state_dict_20 = model_20.state_dict()
    state_dict_25 = model_25.state_dict()


    for key in global_model.state_dict():
        w1 = state_dict_5[key]
        w2 = state_dict_10[key]
        w3 = state_dict_15[key]
        w4 = state_dict_20[key]
        w5 = state_dict_25[key]


        updated_weight = (w1 * 1 + w2 * 2 + w3 * 3 + w4 * 4 + w5 * 5) / 15
        global_model.state_dict()[key].copy_(updated_weight)
    global_model_name = 'updated_global_model_' + str(iteration) +'.pth'
    torch.save(global_model.state_dict(), global_model_name)

    return global_model_name
```

[13]:
```
# Load model
model = resnet50(weights=ResNet50_Weights.DEFAULT)
model.eval()
# Replace the final fully connected layer for transfer learning with the same␣
 ↪num_classes
num_ftrs = model.fc.in_features
num_classes = 6
model.fc = nn.Linear(num_ftrs, num_classes)

global_model_name = "trained_global_model.pth"
torch.save(model.state_dict(), global_model_name)
test_model('../Dataset/validation_data', global_model_name)

global_model_name = "trained_global_model.pth"
i = 0
interval = 1
for i in range(interval):
    print("Iteration: ", i+1)
    print(" Global model ", global_model_name)
    total_5 = len(dataset_5)
    start_index_5 = (i*total_5)//interval
    end_index_5 = min(total_5, ((i+1)*total_5)//interval)
    portion_dataset_5 = torch.utils.data.Subset(dataset_5,␣
 ↪list(range(start_index_5, end_index_5)))
    model_name_5 = "trained_model_5_" + str(i+1) + ".pth"
    print(" Model : ", model_name_5)
    model_5 = train_model(global_model_name, portion_dataset_5, model_name_5)
    test_model('../Dataset/validation_data', model_name_5)

    total_10 = len(dataset_10)
    start_index_10 = (i*total_10)//interval
    end_index_10 = min(total_10, ((i+1)*total_10)//interval)
    portion_dataset_10 = torch.utils.data.Subset(dataset_10,␣
 ↪list(range(start_index_10, end_index_10)))
    model_name_10 = "trained_model_10_" + str(i+1) + ".pth"
    print(" Model : ", model_name_10)
    model_10 = train_model(global_model_name, portion_dataset_10, model_name_10)
    test_model('../Dataset/validation_data', model_name_10)

    total_15 = len(dataset_15)
    start_index_15 = (i*total_15)//interval
    end_index_15 = min(total_15, ((i+1)*total_15)//interval)
    portion_dataset_15 = torch.utils.data.Subset(dataset_15,␣
 ↪list(range(start_index_15, end_index_15)))
    model_name_15 = "trained_model_15_" + str(i+1) + ".pth"
    print(" Model : ", model_name_15)
    model_15 = train_model(global_model_name, portion_dataset_15, model_name_15)
```

```
    test_model('../Dataset/validation_data', model_name_15)

    total_20 = len(dataset_20)
    start_index_20 = (i*total_20)//interval
    end_index_20 = min(total_20, ((i+1)*total_20)//interval)
    portion_dataset_20 = torch.utils.data.Subset(dataset_20,␣
↪list(range(start_index_20, end_index_20)))
    model_name_20 = "trained_model_20_" + str(i+1) + ".pth"
    print(" Model : ", model_name_20)
    model_20 = train_model(global_model_name, portion_dataset_20, model_name_20)
    test_model('../Dataset/validation_data', model_name_20)

    total_25 = len(dataset_25)
    start_index_25 = (i*total_25)//interval
    end_index_25 = min(total_25, ((i+1)*total_25)//interval)
    portion_dataset_25 = torch.utils.data.Subset(dataset_25,␣
↪list(range(start_index_25, end_index_25)))
    model_name_25 = "trained_model_25_" + str(i+1) + ".pth"
    print(" Model : ", model_name_25)
    model_25 = train_model(global_model_name, portion_dataset_25, model_name_25)
    test_model('../Dataset/validation_data', model_name_25)

    ## Update global model

    # Load the model architecture
    global_model = resnet50(weights=None)
    global_model.eval()


    # Replace the final fully connected layer for transfer learning with the␣
↪same num_classes
    num_ftrs = global_model.fc.in_features
    global_model.fc = nn.Linear(num_ftrs, num_classes)

    # Load the trained weights from the saved .pth file
    global_model.load_state_dict(torch.load(global_model_name))
    global_model.eval()

    global_model_name = update_global_model(model_5, model_10, model_15,␣
↪model_20, model_25, global_model, i+1)
    print(" Updated global Model : ", global_model_name)
    test_model('../Dataset/validation_data', global_model_name)
```

```
Top-1 Accuracy on the ../Dataset/validation_data dataset: 20.23%
Top-3 Accuracy on the ../Dataset/validation_data dataset: 56.12%
Iteration:  1
 Global model  trained_global_model.pth
```

```
 Model :  trained_model_5_1.pth
Epoch 1/20 | Loss: 1.0908
Epoch 2/20 | Loss: 0.5128
Epoch 3/20 | Loss: 0.3684
Epoch 4/20 | Loss: 0.3031
Epoch 5/20 | Loss: 0.2556
Epoch 6/20 | Loss: 0.2221
Epoch 7/20 | Loss: 0.1938
Epoch 8/20 | Loss: 0.1770
Epoch 9/20 | Loss: 0.1698
Epoch 10/20 | Loss: 0.1660
Epoch 11/20 | Loss: 0.1493
Epoch 12/20 | Loss: 0.1460
Epoch 13/20 | Loss: 0.1332
Epoch 14/20 | Loss: 0.1241
Epoch 15/20 | Loss: 0.1327
Epoch 16/20 | Loss: 0.1335
Epoch 17/20 | Loss: 0.1097
Epoch 18/20 | Loss: 0.1118
Epoch 19/20 | Loss: 0.1103
Epoch 20/20 | Loss: 0.1038
 Training Time  69.53484010696411
Top-1 Accuracy on the ../Dataset/validation_data dataset: 98.86%
Top-3 Accuracy on the ../Dataset/validation_data dataset: 99.80%
 Model :  trained_model_10_1.pth
Epoch 1/20 | Loss: 0.8849
Epoch 2/20 | Loss: 0.3878
Epoch 3/20 | Loss: 0.2823
Epoch 4/20 | Loss: 0.2301
Epoch 5/20 | Loss: 0.2100
Epoch 6/20 | Loss: 0.1798
Epoch 7/20 | Loss: 0.1659
Epoch 8/20 | Loss: 0.1579
Epoch 9/20 | Loss: 0.1425
Epoch 10/20 | Loss: 0.1336
Epoch 11/20 | Loss: 0.1296
Epoch 12/20 | Loss: 0.1226
Epoch 13/20 | Loss: 0.1164
Epoch 14/20 | Loss: 0.1118
Epoch 15/20 | Loss: 0.1054
Epoch 16/20 | Loss: 0.1051
Epoch 17/20 | Loss: 0.0980
Epoch 18/20 | Loss: 0.0969
Epoch 19/20 | Loss: 0.0919
Epoch 20/20 | Loss: 0.0901
 Training Time  112.94255065917969
Top-1 Accuracy on the ../Dataset/validation_data dataset: 99.24%
Top-3 Accuracy on the ../Dataset/validation_data dataset: 99.98%
```

```
 Model :  trained_model_15_1.pth
Epoch 1/20 | Loss: 0.7299
Epoch 2/20 | Loss: 0.3009
Epoch 3/20 | Loss: 0.2190
Epoch 4/20 | Loss: 0.1863
Epoch 5/20 | Loss: 0.1670
Epoch 6/20 | Loss: 0.1462
Epoch 7/20 | Loss: 0.1288
Epoch 8/20 | Loss: 0.1219
Epoch 9/20 | Loss: 0.1150
Epoch 10/20 | Loss: 0.1075
Epoch 11/20 | Loss: 0.1054
Epoch 12/20 | Loss: 0.0982
Epoch 13/20 | Loss: 0.0931
Epoch 14/20 | Loss: 0.0900
Epoch 15/20 | Loss: 0.0885
Epoch 16/20 | Loss: 0.0881
Epoch 17/20 | Loss: 0.0831
Epoch 18/20 | Loss: 0.0809
Epoch 19/20 | Loss: 0.0768
Epoch 20/20 | Loss: 0.0745
 Training Time  168.6939103603363
Top-1 Accuracy on the ../Dataset/validation_data dataset: 99.37%
Top-3 Accuracy on the ../Dataset/validation_data dataset: 99.98%
 Model :  trained_model_20_1.pth
Epoch 1/20 | Loss: 0.6615
Epoch 2/20 | Loss: 0.2692
Epoch 3/20 | Loss: 0.1916
Epoch 4/20 | Loss: 0.1630
Epoch 5/20 | Loss: 0.1429
Epoch 6/20 | Loss: 0.1272
Epoch 7/20 | Loss: 0.1205
Epoch 8/20 | Loss: 0.1040
Epoch 9/20 | Loss: 0.1024
Epoch 10/20 | Loss: 0.0991
Epoch 11/20 | Loss: 0.0943
Epoch 12/20 | Loss: 0.0921
Epoch 13/20 | Loss: 0.0836
Epoch 14/20 | Loss: 0.0812
Epoch 15/20 | Loss: 0.0820
Epoch 16/20 | Loss: 0.0710
Epoch 17/20 | Loss: 0.0719
Epoch 18/20 | Loss: 0.0724
Epoch 19/20 | Loss: 0.0670
Epoch 20/20 | Loss: 0.0654
 Training Time  219.90845322608948
Top-1 Accuracy on the ../Dataset/validation_data dataset: 99.40%
Top-3 Accuracy on the ../Dataset/validation_data dataset: 99.95%
```

```
 Model :  trained_model_25_1.pth
Epoch 1/20 | Loss: 0.5731
Epoch 2/20 | Loss: 0.2345
Epoch 3/20 | Loss: 0.1713
Epoch 4/20 | Loss: 0.1393
Epoch 5/20 | Loss: 0.1232
Epoch 6/20 | Loss: 0.1152
Epoch 7/20 | Loss: 0.1039
Epoch 8/20 | Loss: 0.0955
Epoch 9/20 | Loss: 0.0873
Epoch 10/20 | Loss: 0.0873
Epoch 11/20 | Loss: 0.0837
Epoch 12/20 | Loss: 0.0805
Epoch 13/20 | Loss: 0.0775
Epoch 14/20 | Loss: 0.0671
Epoch 15/20 | Loss: 0.0678
Epoch 16/20 | Loss: 0.0649
Epoch 17/20 | Loss: 0.0649
Epoch 18/20 | Loss: 0.0637
Epoch 19/20 | Loss: 0.0629
Epoch 20/20 | Loss: 0.0589
 Training Time  296.6254382133484
Top-1 Accuracy on the ../Dataset/validation_data dataset: 99.60%
Top-3 Accuracy on the ../Dataset/validation_data dataset: 99.98%
 Updated global Model :  updated_global_model_1.pth
Top-1 Accuracy on the ../Dataset/validation_data dataset: 99.50%
Top-3 Accuracy on the ../Dataset/validation_data dataset: 99.98%
```

[14]:
```
test_model('../Dataset/test_data', global_model_name)
```

```
Top-1 Accuracy on the ../Dataset/test_data dataset: 99.44%
Top-3 Accuracy on the ../Dataset/test_data dataset: 99.98%
```