quantized-federated-learning

December 20, 2023

0.1 Federated Learning Model with Quantization

[17]:
```python
import torch
from torchvision import models
import copy

# Load the saved model
model = models.resnet50(pretrained=False)
model.fc = torch.nn.Linear(model.fc.in_features, 6)  # Change the output layer␣
↪to match your task

# Load the model state_dict from the saved .pth file
model.load_state_dict(torch.load('updated_global_model_1.pth'))

# Set the model to evaluation mode
model.eval()

# Attach a global qconfig for quantization
model.qconfig = torch.ao.quantization.get_default_qconfig('x86')

# Fuse the activations to preceding layers
model_fused = torch.ao.quantization.fuse_modules(model, [['conv1', 'relu']])

# Prepare the model for static quantization
model_prepared = torch.ao.quantization.prepare(model_fused)

input_fp32 = torch.randn(4, 3, 224, 224)
model_prepared(input_fp32)

# Convert the observed model to a quantized model
quantized_model = torch.ao.quantization.convert(model_prepared)

# Save the quantized model
torch.save(quantized_model.state_dict(), "static_quantized_model.pth")

# Calculate the number of parameters in the quantized model
quantized_model_num_params = sum(p.numel() for p in quantized_model.
↪parameters())
```

```python
print(f"Quantized model size: {quantized_model_num_params} parameters")

# Calculate the number of parameters in the model
model_num_params = sum(p.numel() for p in model.parameters())
print(f"Model size: {model_num_params} parameters")


saved_percentage = ((model_num_params - quantized_model_num_params) /
 ↪model_num_params) * 100
print(f"Memory saved by using quantized parameters: {saved_percentage:.2f}%")
```

```
Quantized model size: 53120 parameters
Model size: 23520326 parameters
Memory saved by using quantized parameters: 99.77%
```

[18]:

```python
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




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)
```

```python
    # 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)
            images = images.to(torch.int8)  # Convert input to int8 if needed

            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}%')
```

[ ]:
```python
test_model('../Dataset/validation_data', "static_quantized_model.pth")
```

[ ]:
```python
test_model('../Dataset/test_data', "static_quantized_model.pth")
```