E7: LoRA

Train a LoRA layer for our pretrained Neural Network from E5 and train it on MNIST:

Ein Bild, das Text, Screenshot, Schrift, Diagramm enthält.

KI-generierte Inhalte können fehlerhaft sein.

Figure : LoRA (<https://arxiv.org/abs/2106.09685>)

Original Source: <https://github.com/microsoft/LoRA/blob/main/loralib/layers.py>

**Preparation (Adaption of E5):**

import torch

from torch import nn

from torch.utils.data import DataLoader

from torchvision import datasets

from torchvision.transforms import ToTensor

# Download training data from open datasets.

training\_data = datasets.FashionMNIST(

    root="data",

    train=True,

    download=True,

    transform=ToTensor(),

)

# Download test data from open datasets.

test\_data = datasets.FashionMNIST(

    root="data",

    train=False,

    download=True,

    transform=ToTensor(),

)

batch\_size = 64

# Create data loaders.

train\_dataloader = DataLoader(training\_data, batch\_size=batch\_size)

test\_dataloader = DataLoader(test\_data, batch\_size=batch\_size)

for X, y in test\_dataloader:

    print(f"Shape of X [N, C, H, W]: {X.shape}")

    print(f"Shape of y: {y.shape} {y.dtype}")

    break

#print(f"Using {device} device")

# Define model

class NeuralNetwork(nn.Module):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        self.flatten = nn.Flatten()

        self.lin\_1 = nn.Linear(28\*28, 128)

        self.act\_1 = nn.ReLU()

        self.lin\_2 = nn.Linear(128, 10)

        self.act\_2 = nn.ReLU()

    def forward(self, x):

        x = self.flatten(x)

        logits = self.lin\_1(x)

        logits = self.act\_1(logits)

        logits = self.lin\_2(logits)

        logits = self.act\_2(logits)

        return logits

device = "cuda" if torch.cuda.is\_available() else "cpu"

model = NeuralNetwork().to(device)

print(model)

loss\_fn = nn.CrossEntropyLoss()

optimizer = torch.optim.SGD(model.parameters(), lr=1e-3)

def train(dataloader, model, loss\_fn, optimizer):

    size = len(dataloader.dataset)

    model.train()

    for batch, (X, y) in enumerate(dataloader):

        X, y = X.to(device), y.to(device)

        # Compute prediction error

        pred = model(X)

        loss = loss\_fn(pred, y)

        # Backpropagation

        loss.backward()

        optimizer.step()

        optimizer.zero\_grad()

        if batch % 100 == 0:

            loss, current = loss.item(), (batch + 1) \* len(X)

            print(f"loss: {loss:>7f}  [{current:>5d}/{size:>5d}]")

def test(dataloader, model, loss\_fn):

    size = len(dataloader.dataset)

    num\_batches = len(dataloader)

    model.eval()

    test\_loss, correct = 0, 0

    with torch.no\_grad():

        for X, y in dataloader:

            X, y = X.to(device), y.to(device)

            pred = model(X)

            test\_loss += loss\_fn(pred, y).item()

            correct += (pred.argmax(1) == y).type(torch.float).sum().item()

    test\_loss /= num\_batches

    correct /= size

    print(f"Test Error: \n Accuracy: {(100\*correct):>0.1f}%, Avg loss: {test\_loss:>8f} \n")

import numpy as np

model\_parameters = filter(lambda p: p.requires\_grad, model.parameters())

params = sum([np.prod(p.size()) for p in model\_parameters])

print(f"Number of trainable parameters: {params}")

epochs = 10

for t in range(epochs):

    print(f"Epoch {t+1}\n-------------------------------")

    train(train\_dataloader, model, loss\_fn, optimizer)

print("Done!")

torch.save(model.state\_dict(), "model.pth")

**TODO:**

Use the encoding Neural Network from E5 and add a LoRA matrix to the linear layer:

1. Create a copy of the class NeuralNetwork and name it LoRAModel
2. Set lin\_1.requires\_grad\_(False) and lin\_2.requires\_grad\_(False)
3. Add 2 Matrices “lora\_A” and “lora\_B” to LoRAModel
   1. Choose which layer you want to retrain with lora (my opinion: lin\_2)
   2. Choose an intermediate shape for the matrices (my opinion: 64)
4. Fill Matrix A with a Kaiming uniform distribution
5. Forward Function: Add matrix multiplication (input @ lora\_A @ lora\_B)
6. Initialize model and state\_dict from pretrained model
7. Print trainable parameters of model
8. Train model for n epochs