AI505/AI801, Optimization – Obligatory Assignment 1

2026-03-28

Case 2: Optimization for Machine Learning

Context

The script `fashionMNIST.py` reported below is taken from AI506 and implements a convolutional neural network (CNN), specifically LeNet5, for the task of image classification.

The fashion data set consists of 70,000 images divided into 60,000 training and 10,000 testing samples. Each image is a 28x28 grayscale image associated with one of 10 classes. These classes include T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, and Ankle boot.

The script is explained in the Pytorch optimization tutorial. Before carrying out the tasks below you are recommended to read also the documentation of the pytorch optimization package.

Your Task

Your goal is to **improve the optimization** in training the given model and evaluating performance on test data. You will experiment with various **stochastic gradient descent (SGD) variants**, including:

- Basic SGD
- Mini-batch and batch SGD
- Noise reduction techniques
- Second-order methods
- Momentum-based optimization methods.

You are also free to experiment with other algorithms from the course.

Scenarios

You **must not modify the LeNet-5 architecture**. Instead, your task is to attain the best accuracy on the test set in the three scenarios:

1. **Standard Cross-Entropy Loss**

   Use the loss function as implemented in the provided starter code.

2. **Regularized Cross-Entropy Loss**

   Introduce

   $$L_2$$

   regularization using the weight_decay parameter in the optimization algorithms.

Evaluation Criteria

For model assessment, you will use:

- **Average loss** on test data, and
- **Test accuracy** (percentage of correctly classified samples).

You **must not modify the data split** beyond what is implemented in the starter code.

Performance Expectations

Run the **mini-batch SGD implementation** from the code handed out. Its result will be your baseline. Your goal is to **outperform this result**.

Computational Considerations

You may use as many **epochs and computational resources** as needed. However, achieving **strong results with fewer computations** is highly desirable.

Discussion

Assess the baseline in your own computational environment.

In other terms, as first thing, run the script provided without changing anything on the machine you will use for the assignment and write in the report: - the final test accuracy and average loss found and - the output of these lines:

```
# Hardware
device = torch.accelerator.current_accelerator().type if torch.accelerator.is_available() else "cpu"
print(f"Using {device} device")
```

There are considerable differences in performance among machines. In particular, using GPU improves run time performance considerably (and it seems also in accuracy, although I have not figured out why yet). In order to make testing feasible for everybody, each group will have access to a machine in U-Cloud with GPU resources. You will be provided a link to a project in you cloud from the Excel spreadsheet that we use to form and report groups in ItsLearning. A brief intro to U-Cloud will be provided in the first class after Easter.

A few hints on making results reproducible in pytorch can be found here: https://pytorch.org/docs/stable/notes/randomness.h
They have been included in the script.

Starter Code

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms, utils
#import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from prettytable import PrettyTable

seed = 42
torch.manual_seed(seed)
#random.seed(seed)
np.random.seed(seed)
torch.backends.cudnn.deterministic = True
torch.backends.cudnn.benchmark = False

# Download the MNIST dataset
transform = transforms.ToTensor()
train_dataset = datasets.FashionMNIST(root='./data', train=True, download=True, transform=transform)
test_dataset = datasets.FashionMNIST(root='./data', train=False, download=True, transform=transform)

print("number of training samples: " + str(len(train_dataset)) + "\n" +
      "number of testing samples: " + str(len(test_dataset)))

print("datatype of the 1st training sample: ", train_dataset[0][0].type())
print("size of the 1st training sample: ", train_dataset[0][0].size())

batch_size = 64

# Create data loaders.
train_loader = torch.utils.data.DataLoader(dataset=train_dataset, batch_size=batch_size, shuffle=True)
test_loader = torch.utils.data.DataLoader(dataset=test_dataset, batch_size=batch_size, shuffle=False)

# Verify size of batches
for X, y in test_loader:
    print(f"Shape of X [N, C, H, W]: {X.shape}")
    print(f"Shape of y: {y.shape} {y.dtype}")
    break

# Hardware
device = torch.accelerator.current_accelerator().type if torch.accelerator.is_available() else "cpu"
print(f"Using {device} device")

# Define the model
class LeNet5(nn.Module):
    def __init__(self):
        super(LeNet5, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, kernel_size=5, stride=1)
        self.conv2 = nn.Conv2d(6, 16, kernel_size=5, stride=1)
        self.fc1 = nn.Linear(256, 120)
```

```
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = torch.tanh(self.conv1(x))
        x = torch.max_pool2d(x, kernel_size=2, stride=2)
        x = torch.tanh(self.conv2(x))
        x = torch.max_pool2d(x, kernel_size=2, stride=2)
        x = x.view(-1, 256)
        x = torch.tanh(self.fc1(x))
        x = torch.tanh(self.fc2(x))
        x = self.fc3(x)
        return x


def count_parameters(model):
    table = PrettyTable(["Modules", "Parameters"])
    total_params = 0
    for name, parameter in model.named_parameters():
        if not parameter.requires_grad:
            continue
        params = parameter.numel()
        table.add_row([name, params])
        total_params += params
    print(table)
    print(f"Total Trainable Params: {total_params}")
    return total_params

model = LeNet5()

count_parameters(model)
fc12_params = [p for name, p in model.named_parameters() if name in ['fc1.weight', 'fc2.weight']]
print(fc12_params[0].numel())
print(fc12_params[1].numel())

# raise SystemExit

# Training loop
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.001)
n_epochs = 10
train_losses = []
test_losses = []

step = 0
for epoch in range(n_epochs):
    model.train()
    for i, (images, labels) in enumerate(train_loader):
        optimizer.zero_grad()
        output = model(images)
        loss = criterion(output, labels)
        loss.backward()
        optimizer.step()
```

```
        step += 1
        train_losses.append((step, loss.item()))
        print(f'Epoch {epoch}, Step {step}, Loss: {loss.item()}')

    model.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for images, labels in test_loader:
            output = model(images)
            test_loss += criterion(output, labels).item()
            pred = torch.argmax(output, dim=1)
            correct += pred.eq(labels).sum()

    test_loss /= len(test_loader.dataset)
    test_losses.append((step, test_loss))
    print(f'Test set: Average loss: {test_loss}, \
        Accuracy: {correct}/{len(test_loader.dataset)} ({100. * correct / len(test_loader.dataset)}%)


# plot train and test losses to file loss.png
train_steps, train_loss = zip(*train_losses)
test_steps, test_loss = zip(*test_losses)

fig, ax = plt.subplots(2, 1, figsize=(8, 6), sharex=True)  # sharex aligns x-axes

# Plot the first
ax[0].plot(train_steps, train_loss, label="Train Loss", color="blue")
ax[0].set_ylabel("Loss")
ax[0].legend()
ax[0].grid(True)

# Plot the second
ax[1].plot(test_steps, test_loss, label="Test Loss, Average", color="red")
interval = int(np.ceil(len(train_dataset)/batch_size))
ax[1].set_xticks(range(0,interval*(n_epochs+1),interval))
ax[1].set_xticklabels(range(n_epochs+1))
ax[1].set_xlabel("Epoch")
ax[1].set_ylabel("Loss")
ax[1].legend()
ax[1].grid(True)

# Adjust layout and show the plot
plt.tight_layout()
# plt.show()
plt.savefig('loss2.png')
```