EX 3 COMPARISON OF SGD WITH MOMENTUM VS ADAM OPTIMIZER

DATE: 28/08/2025

PROBLEM STATEMENT

Implement a training algorithm using Stochastic Gradient Descent (SGD) with momentum and compare it with the Adam optimizer. Train both models on a dataset and compare their convergence rates and performance.

Suggested Dataset: CIFAR-10

Objectives:

1. Understand the principles of optimization algorithms in deep learning.
2. Implement and train models using SGD with momentum and Adam.
3. Analyze and compare the learning behavior and convergence patterns of the two optimizers.
4. Visualize loss and accuracy across epochs for both optimization methods.

Scope:

This experiment gives students a comparative understanding of two widely used optimization strategies: SGD with momentum and Adam. Using a basic MLP and the CIFAR-10 dataset, students will learn the impact of optimizer choice on model convergence and final accuracy.

Tools and Libraries Used:

1. Python 3.x
2. PyTorch
3. Matplotlib
4. torchvision (for CIFAR-10 dataset)

Implementation Steps:

Step 1: Import Necessary Libraries

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
```

Step 2: Set Device and Load Dataset

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
```

```
])
trainset = torchvision.datasets.CIFAR10(root='./data', train=True, transform=transform,
download=True)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=128, shuffle=True)
```

Step 3: Define MLP Model

```
class MLP(nn.Module):
  def __init__(self):
    super().__init__()
    self.flatten = nn.Flatten()
    self.net = nn.Sequential(
      nn.Linear(3*32*32, 256),
      nn.ReLU(),
      nn.Linear(256, 10)
    )

  def forward(self, x):
    x = self.flatten(x)
    return self.net(x)
```

Step 4: Define Training Function

```
def train(model, optimizer, epochs=10):
  model.to(device)
  loss_fn = nn.CrossEntropyLoss()
  losses = []
  accuracies = []

  for epoch in range(epochs):
    total_loss = 0
    correct = 0
    total = 0
    model.train()

    for imgs, labels in trainloader:
      imgs, labels = imgs.to(device), labels.to(device)

      outputs = model(imgs)
      loss = loss_fn(outputs, labels)

      optimizer.zero_grad()
      loss.backward()
      optimizer.step()

      total_loss += loss.item()
      _, predicted = outputs.max(1)
      total += labels.size(0)
      correct += (predicted == labels).sum().item()
```

```
        avg_loss = total_loss / len(trainloader)
        accuracy = 100.0 * correct / total

        losses.append(avg_loss)
        accuracies.append(accuracy)

        print(f"Epoch {epoch+1}: Loss = {avg_loss:.4f}, Accuracy = {accuracy:.2f}%")

    return losses, accuracies
```

Step 5: Train with SGD + Momentum and with Adam

```
model_sgd = MLP()
sgd = optim.SGD(model_sgd.parameters(), lr=0.01, momentum=0.9)
losses_sgd, acc_sgd = train(model_sgd, sgd)
model_adam = MLP()
adam = optim.Adam(model_adam.parameters(), lr=0.001)
losses_adam, acc_adam = train(model_adam, adam)
```

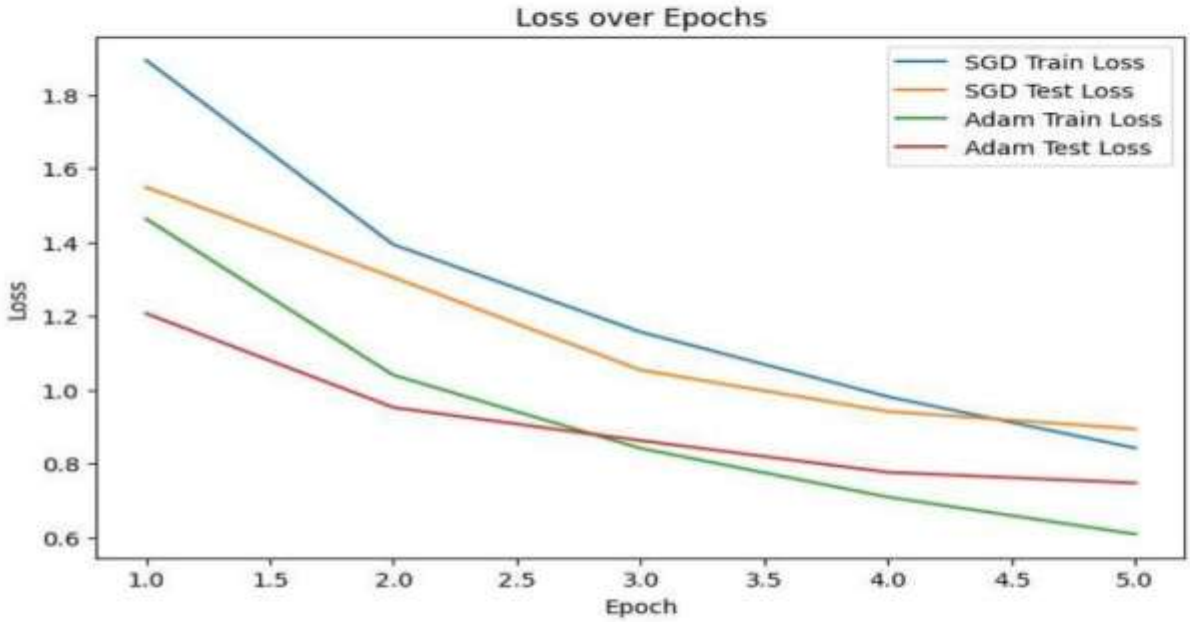
Step 6: Visualize Loss and Accuracy Comparison

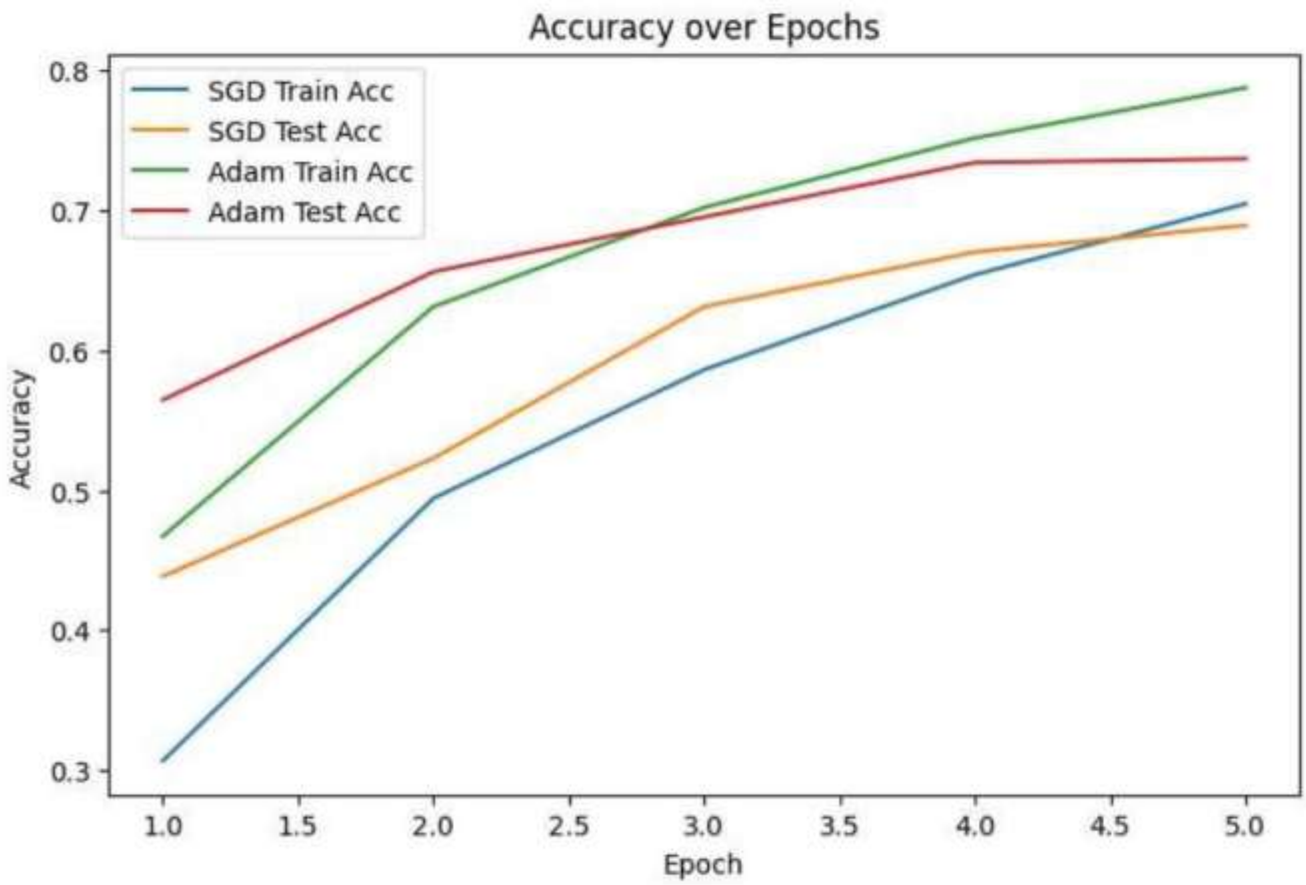
```
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(losses_sgd, label="SGD + Momentum")
plt.plot(losses_adam, label="Adam")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.title("Loss Comparison on CIFAR-10 (MLP)")
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(acc_sgd, label="SGD + Momentum")
plt.plot(acc_adam, label="Adam")
plt.xlabel("Epoch")
plt.ylabel("Accuracy (%)")
plt.title("Accuracy Comparison on CIFAR-10 (MLP)")
plt.legend()
plt.tight_layout()
plt.show()
```

Output:


Loss over Epochs
SGD Train Loss
SGD Test Loss
Adam Train Loss
Adam Test Loss
Loss
Epoch
1.8
1.6
1.4
1.2
1.0
0.8
0.6
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0


Accuracy over Epochs
SGD Train Acc
SGD Test Acc
Adam Train Acc
Adam Test Acc
Accuracy
Epoch
0.8
0.7
0.6
0.5
0.4
0.3
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0