Assignment 2: Stochastic Gradient Descent

Deadline: 11:59PM CST(China Standard Time), May 24, 2025

1 Dataset

MNIST (Modified National Institute of Standards and Technology) is a widely used dataset in the field of machine learning and computer vision. It consists of a collection of handwritten digits, from 0 to 9, that have been manually labeled with their corresponding values. Each digit image in the dataset is a grayscale image of size 28x28 pixels.

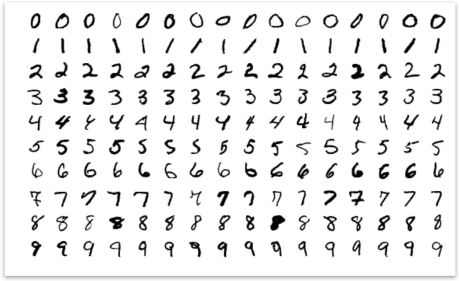

Figure 1: MNIST example

2 Homework Grade Policy

Submission Format: Please format your homework **in English** using one of the two options below. Failing to do so will result in a minimum deduction of **10 points** from your homework grade:

1. **One PDF** generated by Jupyter Notebook that includes your code, complete output results, plotted figures, and written answers to analytical questions.
2. **One PDF + One PY file.** The `.py` file must contain all your code and be fully runnable. The PDF should include screenshots of your output results, plotted figures, along with your written answers to analytical questions.

Late Policy: Homework submitted within 72 hours after the deadline will receive 80% of the earned score. Submissions made more than three days after the deadline without a prior extension request will receive a score of 0. The request must be made before the deadline of the homework.

3 Model

In this section, we will walk through a Python code that demonstrates the usage of PyTorch to perform stochastic optimization using the MNIST dataset. The code uses stochastic gradient descent (SGD) as the optimization algorithm and explores the effects of different hyperparameters such as learning rate and batch size.

3.1 Importing Required Libraries

The code begins by importing the necessary libraries: `torch` for PyTorch functionalities, `torch.nn` for neural network components, `torch.optim` for optimization algorithms, and `torchvision` for loading the MNIST dataset.

```
1 import torch
2 import torch.nn as nn
3 import torch.optim as optim
4 import torchvision.datasets.MNIST as MNIST
5 import torchvision.transforms as transforms
6 import torch.utils.data.DataLoader as DataLoader
```

3.2 MNIST Dataset

We load the MNIST dataset using the `torchvision.datasets` module. The dataset is divided into training and test sets. We apply the `transforms.ToTensor()` transformation to convert the images to PyTorch tensors.

```
1 batch_size = 32
2 train_dataset = MNIST(root='./data', train=True, transform=
      transforms.ToTensor(), download=True)
3 test_dataset = MNIST(root='./data', train=False, transform=
      transforms.ToTensor(), download=True)
4 train_loader = DataLoader(dataset=train_dataset, batch_size=
      batch_size, shuffle=True)
5 test_loader = DataLoader(dataset=test_dataset, batch_size=
      batch_size, shuffle=False)
```

3.3 Neural Network Model

We define our neural network model using the `torch.nn.Module` class. Our model consists of three fully connected layers with ReLU activation functions. The input size is 784 (28x28 pixels), and the output size is 10 (corresponding to the digits 0-9).

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(784, 512)
        self.fc2 = nn.Linear(512, 256)
        self.fc3 = nn.Linear(256, 10)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = x.view(-1, 784)
        x = self.relu(self.fc1(x))
        x = self.relu(self.fc2(x))
        x = self.fc3(x)
        return x

model = Net()
```

3.4 Loss Function and Optimizer

We specify the loss function as cross-entropy loss (`torch.nn.CrossEntropyLoss()`). For optimization, we use stochastic gradient descent (SGD) as the optimization algorithm (`torch.optim.SGD`). We pass the model parameters and the learning rate to the optimizer.

```
learning_rate = 0.1
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=learning_rate)
```

3.5 Training Loop

We start the training loop, which consists of nested iterations over the epochs and batches. Inside the loop, we perform the following steps:

- Forward pass: We pass the batch of images through the model and obtain the predicted outputs.
- Loss calculation: We calculate the loss between the predicted outputs and the actual labels using the defined loss function.

- Backward pass: We compute the gradients of the model's parameters with respect to the loss using automatic differentiation.
- Parameter update: We update the model's parameters using the gradients and the optimizer's update rule (`optimizer.step()`).

```
num_epochs = 20
for epoch in range(num_epochs):
    for i, (images, labels) in enumerate(train_loader):
        # Forward pass
        outputs = model(images)

        #Loss calculation
        loss = criterion(outputs, labels)

        # Backward
        optimizer.zero_grad()
        loss.backward()

        # Optimize
        optimizer.step()
```

You can use `loss.item()` to check the loss of the current batch.

3.6 Evaluate the Model on Train Dataset

After training, we switch the model to evaluation mode (`model.eval()`). We then iterate over the train dataset and calculate the accuracy of the model's predictions. The accuracy is printed as a percentage.

```
model.eval()
with torch.no_grad():
    correct = 0
    total = 0
    for images, labels in train_loader:
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

    print(f'Train Accuracy: {100 * correct / total}%')
```

4 Questions

1. **15pts:** Using the provided code as a reference, implement additional code to compute and print the prediction accuracy on the test dataset.

2. Initialize and train a new model using each of the specified hyperparameter configurations. For each run, **plot the training loss versus the number of epochs, and report the training and test accuracies.** Each run is expected to complete within 5 to 30 minutes, depending on your computer's specifications.

   (a) **15pts:** `batch_size = 32`, `num_epochs = 20`, `learning_rate = 0.1`.

   (b) **15pts:** `batch_size = 32`, `num_epochs = 20`, `learning_rate = 1`.

   (c) **15pts:** `batch_size = 32`, `num_epochs = 20`, `learning_rate = 0.01`.

   (d) **15pts:** `batch_size = 8`, `num_epochs = 20`, `learning_rate = 0.1`.

   (e) **15pts:** `batch_size = 128`, `num_epochs = 20`, `learning_rate = 0.1`.

3. **5pts:** Analyze the effect of the learning rate on the training process. Specifically, discuss what happens to the training loss when the learning rate is high vs low.

4. **5pts:** Analyze the effect of batch size on the training process. Specifically, discuss what happens to the training loss when the batch size is large vs small.