ML TA hours

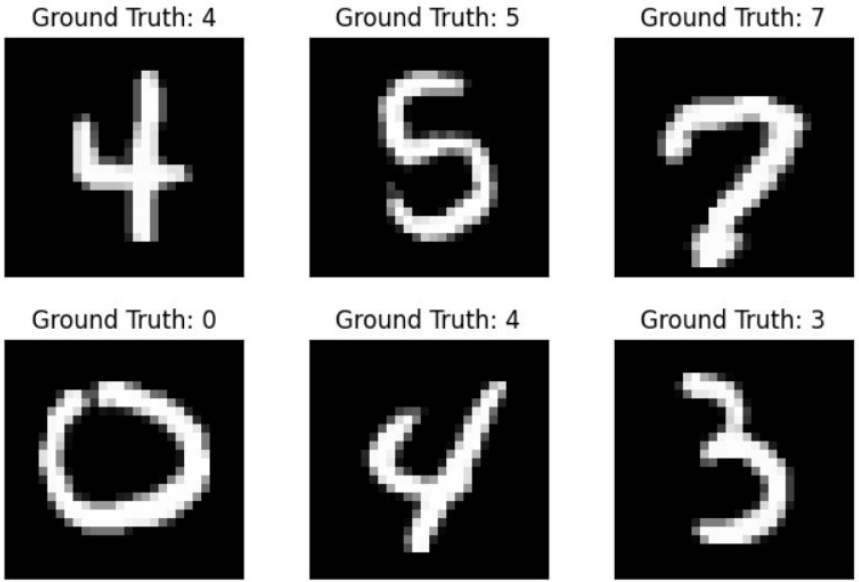
HW6

Colab experiment : MNIST classification

2024.11.5

Task description

- In this exercise, we apply CNN to MNIST data to classify the handwritten digits.



Deep learning in Pytorch

1. Data preprocessing
2. Make your dataset
3. Build your model
4. Train and test Model

Deep learning in Pytorch

1. **Data preprocessing**
2. Make your dataset
3. Build your model
4. Train and test Model

Data preprocessing

- Data cleaning (e.g., handling missing or abnormal values)
- Data standardization or normalization (e.g., scaling to the [0,1] range)
- Data augmentation (e.g., rotation, flipping, etc.) ...

```
torchvision.datasets.MNIST('/files/',   train=True,   download=True,
                                transform=torchvision.transforms.Compose([
                                    torchvision.transforms.ToTensor(),
                                    torchvision.transforms.Normalize(
                                        (0.1307,),   (0.3081,))
                                ])),
```

Deep learning in Pytorch

1. Data preprocessing
2. **Make your dataset**
3. Build your model
4. Train and test Model

Make your dataset

Dataset

1. **Loading a Dataset** (The method of this assignment)
2. **Creating a Custom Dataset for your files**

   from torch.utils.data.dataset import Dataset

   class Custom_Dataset(Dataset):

Dataloader

- **It is a PyTorch tool for batch loading data.**
  torch.utils.data.DataLoader(dataset, batch_size, shuffle,…)

Deep learning in Pytorch

1. Data preprocessing
2. Make your dataset
3. **Build your model**
4. Train and test Model

Build your model

```
class Your_Net_Name(nn.Module):
    def __init__(self):
        …
        …
    def forward(self, x):
        …
        …
```

def __init__(self):

- __init__ is the class constructor, used to initialize the layers of the model
- You can define the structure of each layer here

```
def __init__(self):
        super(Net, self).__init__()
        # Valid convolution, 1 channel in, 2 channels out, stride 1, kernel size = 3
        self.conv1 = nn.Conv2d(1, 2, kernel_size=3)
        # Dropout for convolutions
        self.drop = nn.Dropout2d()
        # Fully connected layer
        self.fc1 = nn.Linear(338, 10)
```

def forward(self, x):

- x is the input data to the model
- The forward method defines how data is propagated forward through each layer.

```
def forward(self, x):
    x = self.conv1(x)
    x = self.drop(x)
    x = F.max_pool2d(x, 2)
    x = F.relu(x)
    x = x.flatten(1)
    x = self.fc1(x)
    x = F.log_softmax(x)
    return x
```

def forward(self, x)

x = self.conv1(x)

x = self.drop(x)

x = F.max_pool2d(x,2)

x = F.relu(x)

...

TODO - Implement the Net2 with the steps

```
# TODO Change above Net to Net2 class to implement
# 1. A valid convolution with kernel size 5, 1 input channel and 10 output channels
# 2. A max pooling operation over a 2x2 area
# 3. A Relu
# 4. A valid convolution with kernel size 5, 10 input channels and 20 output channels
# 5. A 2D Dropout layer
# 6. A max pooling operation over a 2x2 area
# 7. A relu
# 8. A flattening operation
# 9. A fully connected layer mapping from (whatever dimensions we are at-- find out using .shape) to 50
# 10. A ReLU
# 11. A fully connected layer mapping from 50 to 10 dimensions
# 12. A softmax function.

class Net2(nn.Module):
        def __init__(self):


        def forward(self, x):
```

Deep learning in Pytorch

1. Data preprocessing
2. Make your dataset
3. Build your model
4. **Train and test Model**

TODO - Read and Understand the training and testing steps

- Write some descriptions or comments

```
# Main training routine
# TODO: Read it and understand what it does, you would need to implement it in the next colab HW
def train(epoch, model):
    model.train()
    # Get each
    for batch_idx, (data, target) in enumerate(train_loader):
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
        # Store results
        if batch_idx % 10 == 0:

            pred = output.data.max(1, keepdim=True)[1]
            correct = pred.eq(target.data.view_as(pred)).sum()
            print('Train Epoch: {} [{}/{}]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset), loss.item()))
```

TODO - Read and Understand the training and testing steps

- Write some descriptions or comments

```
# Run on test data
# TODO: Read it and understand what it does, you would need to implement it in the next colab HW
def test(model):
    model.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            output = model(data)
            test_loss += F.nll_loss(output, target, size_average=False).item()
            pred = output.data.max(1, keepdim=True)[1]
            correct += pred.eq(target.data.view_as(pred)).sum()
    test_loss /= len(test_loader.dataset)
    print('\nTest set: Avg. loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(
        test_loss, correct, len(test_loader.dataset),
        100. * correct / len(test_loader.dataset)))
    return 100. * correct / len(test_loader.dataset)
```

Summarize what you need to do

1. Implement the Net2 with the steps

2. Read and Understand the training and testing steps
   (write some description or comments)

Base line

```
Train Epoch: 10 [56960/60000]   Loss: 0.148194
Train Epoch: 10 [57600/60000]   Loss: 0.051312
Train Epoch: 10 [58240/60000]   Loss: 0.112016
Train Epoch: 10 [58880/60000]   Loss: 0.018661
Train Epoch: 10 [59520/60000]   Loss: 0.031718

Test set: Avg. loss: 0.0405, Accuracy: 9868/10000 (99%)


Model 2 Accuracy: 98.68%
```

Submission

- After executing your code, download the .ipynb file and submit it to NTU COOL
- Submitted file name: student ID_week10_colab_homework.ipynb
- HW3 Deadline : 2024/11/11 23:59 (Monday night)
- No late submission

- If there are any questions

  Email: r12945048@ntu.edu.tw (add “[ML HW6]” to the beginning of the title.)