mark_goldstein_mg3479_A3_code

March 12, 2020

0.1 0. Setup

```
In [ ]: # Import dependencies
        import torch
        import torch.nn as nn
        import torch.nn.functional as F
        import torch.optim as optim
        from torch.utils.data import DataLoader
        from torchvision import datasets, transforms
```

```
In [ ]: # Set up your device
        cuda = torch.cuda.is_available()
        device = torch.device("cuda:0" if cuda else "cpu")
```

```
In [ ]: # Set up random seed to 1008. Do not change the random seed.
        # Yes, these are all necessary when you run experiments!
        seed = 1008
        random.seed(seed)
        np.random.seed(seed)
        torch.manual_seed(seed)
        if cuda:
            torch.cuda.manual_seed(seed)
            torch.cuda.manual_seed_all(seed)
            torch.backends.cudnn.benchmark = False
            torch.backends.cudnn.deterministic = True
```

0.2 1. Data: MNIST

Load the MNIST training and test dataset using `torch.utils.data.DataLoader` **and** `torchvision.datasets`**.** Hint: You might find Alf's notebook useful: https://github.com/Atcold/pytorch-Deep-Learning/blob/master/06-convnet.ipynb, or see some of the PyTorch tutorials.

0.2.1 1.1. Load Training Set [4 pts]

```
In [ ]: # Load the MNIST training set with batch size 128, apply data shuffling and normalizatio
        # train_loader = TODO
```

0.2.2 1.1. Load Test Set [4 pts]

```
In [ ]: # Load the MNIST test set with batch size 128, apply data shuffling and normalization
        # test_loader = TODO
```

0.3 2. Models

You are going to define two convolutional neural networks which are trained to classify MNIST digits

0.3.1 2.1. CNN without Batch Norm [5 pts]

```
In [ ]: # Fill in the values below that make this network valid for MNIST data

        # conv1_in_ch = TODO
        # conv2_in_ch = TODO
        # fc1_in_features = TODO
        # fc2_in_features = TODO
        # n_classes = TODO
```

```
In [ ]: class NetWithoutBatchNorm(nn.Module):
            def __init__(self):
                super(NetWithoutBatchNorm, self).__init__()
                self.conv1 = nn.Conv2d(in_channels=conv1_in_ch, out_channels=20, kernel_size=5,
                self.conv2 = nn.Conv2d(in_channels=conv2_in_ch, out_channels=50, kernel_size=5,
                self.fc1 = nn.Linear(in_features=fc1_in_features, out_features=500)
                self.fc2 = nn.Linear(in_features=fc2_in_features, out_features=n_classes)

            def forward(self, x):
                x = F.relu(self.conv1(x))
                x = F.max_pool2d(x, kernel_size=2, stride=2)
                x = F.relu(self.conv2(x))
                x = F.max_pool2d(x, kernel_size=2, stride=2)
                x = x.view(-1, fc1_in_features) # reshaping
                x = F.relu(self.fc1(x))
                x = self.fc2(x)
                # Return the log_softmax of x.
                # return TODO
```

0.3.2 2.2. CNN with Batch Norm [5 pts]

```
In [ ]: # Fill in the values below that make this network valid for MNIST data

        # conv1_bn_size = TODO
        # conv2_bn_size = TODO
        # fc1_bn_size = TODO
```

```
In [ ]: # Define the CNN with architecture explained in Part 2.2
        class NetWithBatchNorm(nn.Module):
```

```python
    def __init__(self):
        super(NetWithBatchNorm, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=conv1_in_ch, out_channels=20, kernel_size=5,
        self.conv1_bn = nn.BatchNorm2d(conv1_bn_size)
        self.conv2 = nn.Conv2d(in_channels=conv2_in_ch, out_channels=50, kernel_size=5,
        self.conv2_bn = nn.BatchNorm2d(conv2_bn_size)
        self.fc1 = nn.Linear(in_features=fc1_in_features, out_features=500)
        self.fc1_bn = nn.BatchNorm1d(fc1_bn_size)
        self.fc2 = nn.Linear(in_features=fc2_in_features, out_features=n_classes)

    def forward(self, x):
        x = F.relu(self.conv1_bn(self.conv1(x)))
        x = F.max_pool2d(x, kernel_size=2, stride=2)
        x = F.relu(self.conv2_bn(self.conv2(x)))
        x = F.max_pool2d(x, kernel_size=2, stride=2)
        x = x.view(-1, fc1_in_features)
        x = F.relu(self.fc1_bn(self.fc1(x)))
        x = self.fc2(x)
        # Return the log_softmax of x.
        # return TODO
```

0.4 3. Training & Evaluation

0.4.1 3.1. Define training method [10 pts]

```python
In [ ]: def train(model, device, train_loader, optimizer, epoch, log_interval = 100):
            # Set model to training mode
            model.train()
            # Loop through data points
            for batch_idx, (data, target) in enumerate(train_loader):
                pass # remove once implemented

                # Send data and target to device
                # TODO

                # Zero out the ortimizer
                # TODO

                # Pass data through model
                # TODO

                # Compute the negative log likelihood loss
                # TODO

                # Backpropagate loss
                # TODO

                # Make a step with the optimizer
```

```
            # TODO

            # Print loss (uncomment lines below once implemented)
            # if batch_idx % log_interval == 0:
                # print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                    # epoch, batch_idx * len(data), len(train_loader.dataset),
                    # 100. * batch_idx / len(train_loader), loss.item()))
```

0.4.2 3.2. Define test method [10 pts]

```
In [ ]: # Define test method
        def test(model, device, test_loader):
            # Set model to evaluation mode
            model.eval()
            # Variable for the total loss
            test_loss = 0
            # Counter for the correct predictions
            num_correct = 0

            # don't need autograd for eval
            with torch.no_grad():
                # Loop through data points
                for data, target in test_loader:
                    pass # remove once implemented

                    # Send data to device
                    # TODO

                    # Pass data through model
                    # TODO

                    # Compute the negative log likelihood loss with reduction='sum' and add to t
                    # TODO

                    # Get predictions from the model for each data point
                    # TODO

                    # Add number of correct predictions to total num_correct
                    # TODO

            # Compute the average test_loss
            # avg_test_loss = TODO

            # Print loss (uncomment lines below once implemented)
            # print('\nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(
                # avg_test_loss, num_correct, len(test_loader.dataset),
                # 100. * num_correct / len(test_loader.dataset)))
```

0.4.3 3.3 Train NetWithoutBatchNorm() [5 pts]

In [ ]:
```
# Deifne model and sent to device
# model = TODO

# Optimizer: SGD with learning rate of 1e-2 and momentum of 0.5
# optimizer = TODO

# Training loop with 10 epochs
for epoch in range(1, 10 + 1):
    pass # remove once implemented

    # Train model
    # TODO

    # Test model
    # TODO
```

0.4.4 3.4 Train NetWithBatchNorm() [5 pts]

In [ ]:
```
# Deifne model and sent to device
# model = TODO

# Optimizer: SGD with learning rate of 1e-2 and momentum of 0.5
# optimizer = TODO

# Training loop with 10 epochs
for epoch in range(1, 10 + 1):
    pass # remove once implemented

    # Train model
    # TODO

    # Test model
    # TODO
```

0.5 4. Empirically, which of the models achieves higher accuracy faster? [2 pts]

Answer: