Neural Networks in Pytorch

Recitation 11/16 & 11/17

Creating a dataset

- Pytorch supports a lot of popular datasets
- Transforms can be applied to the data

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

train_data = torchvision.datasets.MNIST(root = './data', train = True, download = True, transform = transform)
test_data = torchvision.datasets.MNIST(root = './data', train = False, download = True, transform = transform)
```

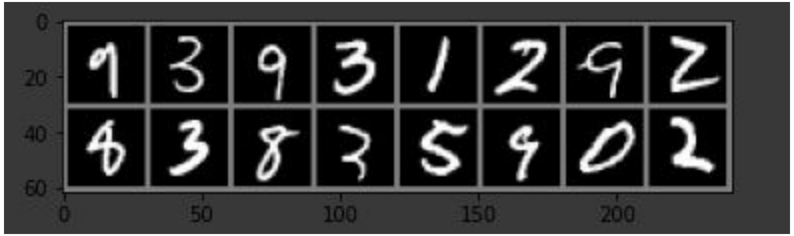
Dataloaders

- Customized iterator over the dataset
- Can change lots of parameters
- More convenient for training than looping through dataset manually

```
[ ]  trainloader = torch.utils.data.DataLoader(train_data, batch_size=16, shuffle=True)
     testloader = torch.utils.data.DataLoader(test_data, batch_size=len(test_data), shuffle=False)
```

Our data

- MNIST consists of visual representations of numbers from 0-9
- Goal is to identify number from image



MLP Architecture

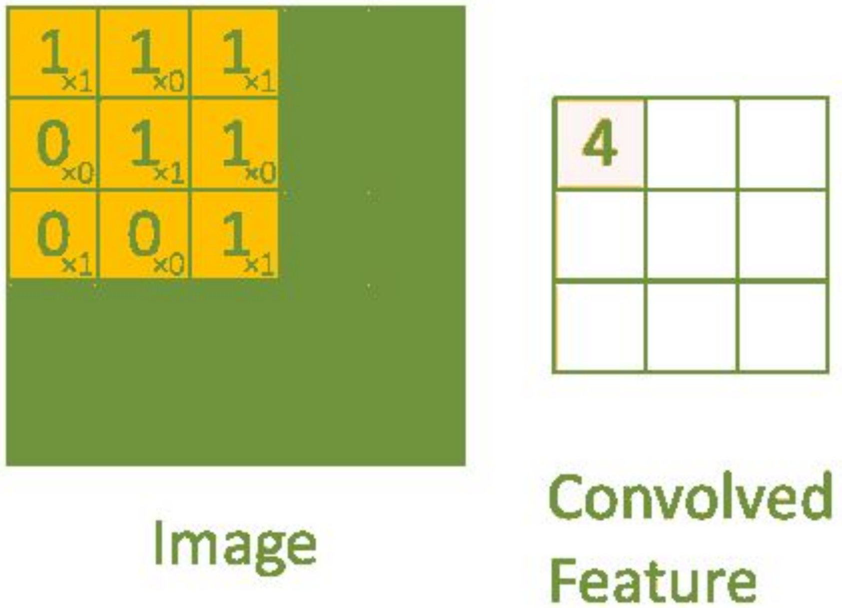
- Flatten is very important for feeding images through linear layers
- Linear layers require 2d input with shape (batch_size, x)
- Final layer output is of size 10, represents possible classes
- Softmax converts values to probabilities
- forward() is executed whenever model is called

```python
class FF(nn.Module):
  def __init__(self):
    super(FF, self).__init__()
    self.fc1 = nn.Linear(28*28, 1500)
    self.fc2 = nn.Linear(1500, 3500)
    self.fc3 = nn.Linear(3500, 1500)
    self.fc4 = nn.Linear(1500, 10)

  def forward(self, x):
    x = torch.flatten(x, 1)
    out = self.fc1(x)
    out = self.fc2(out)
    out = self.fc3(out)
    out = self.fc4(out)
    out = F.softmax(out, 1)

    return out
```

CNN

- Considers positional information
- Considered better for images than linear layers
- Weighted average calculated on sections of images

CNN



CNN Architecture

```python
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 4 * 4, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = torch.flatten(x,1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

Defining Model Architecture

```
ff = FF()
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(ff.parameters(), lr=1e-3, momentum=0.9)
```

CE Loss:

- Good for multiclass classification
- Attempts to maximize probability for true class, minimize for others

- E.g. Classes = {1,2,3}, true label = 2
- Tries to push model outputs towards [0,1,0]

Training Loop

How many loops over the dataset

Iterate through dataloader

```
for epoch in range(1):
  running_loss = 0.0
  for i, data in enumerate(tqdm.tqdm(trainloader)):
      # get the inputs; data is a list of [inputs, labels]
      inputs, labels = data

      # zero the parameter gradients
      optimizer.zero_grad()

      # forward + backward + optimize
      outputs = ff(inputs)
      loss = criterion(outputs, labels)
      loss.backward()
      optimizer.step()

      # print statistics
      running_loss += loss.item()
      if i % 500 == 499:    # print every 500 mini-batches
          print(f'Epoch: {epoch}, Eg: {i+1}, Loss: {running_loss/500}')
          running_loss = 0.0
```

Seperate data

Clears memory from previous minibatch

Calculates model outputs

Calculate loss over minibatch

Calculate backpropagation of loss

Update parameters

Testing loop

Do not accumulate gradients

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

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

Calculate class with highest probability