Prithvi_Poddar_17191_Report_3_Autoencoders

June 24, 2020

1 Report 3: Autoencoders

1.1 Prithvi Poddar 17191

In this reort, we'll be developing Sparse autoencoder

1.2 Sparse Autoencoder

1.2.1 This autoencoder structure has been inspired by the note of Andrew NG on sparse autoencoders. We'll use the parameters mentioned in the notes itself

[1]:
```
!pip3 install torch torchvision
```

```
Requirement already satisfied: torch in /usr/local/lib/python3.6/dist-packages
(1.5.1+cu101)
Requirement already satisfied: torchvision in /usr/local/lib/python3.6/dist-
packages (0.6.1+cu101)
Requirement already satisfied: future in /usr/local/lib/python3.6/dist-packages
(from torch) (0.16.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages
(from torch) (1.18.5)
Requirement already satisfied: pillow>=4.1.1 in /usr/local/lib/python3.6/dist-
packages (from torchvision) (7.0.0)
```

[39]:
```
import torch
import matplotlib.pyplot as plt
import numpy as np
import torch.nn.functional as F
from torch import nn
from torchvision import datasets, transforms
```

[2]:
```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
device
```

[2]:
```
device(type='cuda', index=0)
```

[3]:
```
transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.
↪5,),(0.5))])
```

[4]:
```
training_dataset = datasets.MNIST(root='./data', train=True, download=True,␣
 ↪transform=transform)
validation_dataset = datasets.MNIST(root='./data', train=False, download=True,␣
 ↪transform=transform)
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to
./data/MNIST/raw/train-images-idx3-ubyte.gz

HBox(children=(FloatProgress(value=1.0, bar_style='info', max=1.0), HTML(value='')))

Extracting ./data/MNIST/raw/train-images-idx3-ubyte.gz to ./data/MNIST/raw
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to
./data/MNIST/raw/train-labels-idx1-ubyte.gz

HBox(children=(FloatProgress(value=1.0, bar_style='info', max=1.0), HTML(value='')))

Extracting ./data/MNIST/raw/train-labels-idx1-ubyte.gz to ./data/MNIST/raw
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to
./data/MNIST/raw/t10k-images-idx3-ubyte.gz

HBox(children=(FloatProgress(value=1.0, bar_style='info', max=1.0), HTML(value='')))

Extracting ./data/MNIST/raw/t10k-images-idx3-ubyte.gz to ./data/MNIST/raw
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to
./data/MNIST/raw/t10k-labels-idx1-ubyte.gz

HBox(children=(FloatProgress(value=1.0, bar_style='info', max=1.0), HTML(value='')))

Extracting ./data/MNIST/raw/t10k-labels-idx1-ubyte.gz to ./data/MNIST/raw
Processing...
Done!

/pytorch/torch/csrc/utils/tensor_numpy.cpp:141: UserWarning: The given NumPy
array is not writeable, and PyTorch does not support non-writeable tensors. This
means you can write to the underlying (supposedly non-writeable) NumPy array
using the tensor. You may want to copy the array to protect its data or make it
writeable before converting it to a tensor. This type of warning will be
suppressed for the rest of this program.
```

[40]:
```
training_loader = torch.utils.data.DataLoader(training_dataset, batch_size=32,␣
 ↪shuffle=True, num_workers=0)
validation_loader = torch.utils.data.DataLoader(validation_dataset, batch_size␣
 ↪= 32, shuffle=False, num_workers=0)
```

We start off by defining our autoencoder structure. It will have 5 linear enoding layers and 5 linear decoding layers as can be seen below

[6]:
```
class SparseAutoencoder(nn.Module):

    def __init__(self):
        super(SparseAutoencoder, self).__init__()

        #encoder
        self.en1 = nn.Linear(784, 256)
        self.en2 = nn.Linear(256, 128)
        self.en3 = nn.Linear(128, 64)
        self.en4 = nn.Linear(64, 32)
        self.en5 = nn.Linear(32, 16)

        #decoder
        self.de1 = nn.Linear(16, 32)
        self.de2 = nn.Linear(32, 64)
        self.de3 = nn.Linear(64, 128)
        self.de4 = nn.Linear(128, 256)
        self.de5 = nn.Linear(256, 784)

    def forward(self, x):
        #encoding
        x = F.relu(self.en1(x))
        x = F.relu(self.en2(x))
        x = F.relu(self.en3(x))
        x = F.relu(self.en4(x))
        x = F.relu(self.en5(x))

        #decoding
        x = F.relu(self.de1(x))
        x = F.relu(self.de2(x))
        x = F.relu(self.de3(x))
        x = F.relu(self.de4(x))
        x = F.relu(self.de5(x))

        return x
```

[7]:
```
model = SparseAutoencoder().to(device)
```

Nex we extract the layers in the autoencoder individually so that we can use them to calculate the sparse loss by passing the imput images separately through all the layers

[8]:
```
model_children = list(model.children())
```

Defining the KL divergence and Sparse Loss

[9]:
```
def kl_div(rho, rho_hat):
  rho_hat = torch.mean(torch.sigmoid(rho_hat), 1) #sigmoid to squash the output␣
↪between 0-1 to get probability distribution
  rho = torch.tensor([rho] * len(rho_hat)).to(device) # converting rho into a␣
↪tensor of same size as rho_hat, for further calculations
  return torch.sum(rho * torch.log(rho/rho_hat) + (1 - rho) * torch.log((1 -␣
↪rho)/(1 - rho_hat)))

def sparse_loss(rho, images):
  values = images
  loss = 0
  for i in range(len(model_children)):
    values = model_children[i](values) #computing the activation after passing␣
↪through the layer i of the model
    loss+=kl_div(rho, values)
  return loss
```

Finally we define the functin to train our model:

[10]:
```
def train(model, epochs, learning_rate, RHO, BETA):
  criterion = nn.MSELoss()
  optimizer = torch.optim.Adam(model.parameters(), lr = learning_rate)

  training_losses = []
  test_losses = []

  for e in range(epochs):
    training_loss = 0.0
    test_loss = 0.0
    for inputs, labels in training_loader:
      inputs = inputs.view(inputs.shape[0], -1)
      inputs = inputs.to(device)
      outputs = model.forward(inputs)
      mse_loss = criterion(outputs, inputs)
      sparsity = sparse_loss(RHO, inputs)
      loss = mse_loss + BETA*sparsity
      optimizer.zero_grad()
      loss.backward()
      optimizer.step()

      training_loss+=loss.item()

    else:
      with torch.no_grad():
        for val_inputs, val_labels in validation_loader:
          val_inputs = val_inputs.to(device)
          val_inputs = val_inputs.view(val_inputs.shape[0], -1)
          val_outputs = model.forward(val_inputs)
```

```python
            val_mse_loss = criterion(val_outputs, val_inputs)
            val_sparsity = sparse_loss(RHO, val_inputs)
            val_loss = val_mse_loss + BETA*val_sparsity
            test_loss += val_loss.item()

        epoch_loss = training_loss/len(training_loader)
        training_losses.append(epoch_loss)
        val_epoch_loss = test_loss/len(validation_loader)
        test_losses.append(val_epoch_loss)
        print('epoch :', (e+1))
        print('training loss: {:.4f} '.format(epoch_loss))
        print('validation loss: {:.4f} '.format(val_epoch_loss))

    return training_losses, test_losses
```

[11]:
```python
training_losses, test_losses = train(model, 100, 0.0001, 0.05, 0.001)
```

```
epoch : 1
training loss: 0.9419
validation loss: 0.9367
epoch : 2
training loss: 0.9363
validation loss: 0.9367
epoch : 3
training loss: 0.9359
validation loss: 0.9341
epoch : 4
training loss: 0.9323
validation loss: 0.9275
epoch : 5
training loss: 0.9268
validation loss: 0.9254
epoch : 6
training loss: 0.9247
validation loss: 0.9229
epoch : 7
training loss: 0.9223
validation loss: 0.9198
epoch : 8
training loss: 0.9195
validation loss: 0.9178
epoch : 9
training loss: 0.9170
validation loss: 0.9152
epoch : 10
training loss: 0.9154
validation loss: 0.9138
epoch : 11
```

```
training loss: 0.9143
validation loss: 0.9127
epoch : 12
training loss: 0.9129
validation loss: 0.9109
epoch : 13
training loss: 0.9106
validation loss: 0.9079
epoch : 14
training loss: 0.9080
validation loss: 0.9058
epoch : 15
training loss: 0.9063
validation loss: 0.9044
epoch : 16
training loss: 0.9050
validation loss: 0.9033
epoch : 17
training loss: 0.9039
validation loss: 0.9021
epoch : 18
training loss: 0.9024
validation loss: 0.8999
epoch : 19
training loss: 0.9008
validation loss: 0.8991
epoch : 20
training loss: 0.8998
validation loss: 0.8979
epoch : 21
training loss: 0.8990
validation loss: 0.8957
epoch : 22
training loss: 0.8966
validation loss: 0.8948
epoch : 23
training loss: 0.8958
validation loss: 0.8939
epoch : 24
training loss: 0.8950
validation loss: 0.8930
epoch : 25
training loss: 0.8941
validation loss: 0.8921
epoch : 26
training loss: 0.8931
validation loss: 0.8911
epoch : 27
```

```
training loss: 0.8924
validation loss: 0.8906
epoch : 28
training loss: 0.8918
validation loss: 0.8900
epoch : 29
training loss: 0.8909
validation loss: 0.8890
epoch : 30
training loss: 0.8903
validation loss: 0.8885
epoch : 31
training loss: 0.8897
validation loss: 0.8877
epoch : 32
training loss: 0.8884
validation loss: 0.8860
epoch : 33
training loss: 0.8874
validation loss: 0.8856
epoch : 34
training loss: 0.8869
validation loss: 0.8851
epoch : 35
training loss: 0.8865
validation loss: 0.8847
epoch : 36
training loss: 0.8859
validation loss: 0.8840
epoch : 37
training loss: 0.8853
validation loss: 0.8836
epoch : 38
training loss: 0.8843
validation loss: 0.8823
epoch : 39
training loss: 0.8835
validation loss: 0.8815
epoch : 40
training loss: 0.8828
validation loss: 0.8809
epoch : 41
training loss: 0.8822
validation loss: 0.8803
epoch : 42
training loss: 0.8817
validation loss: 0.8799
epoch : 43
```

```
training loss: 0.8811
validation loss: 0.8792
epoch : 44
training loss: 0.8806
validation loss: 0.8787
epoch : 45
training loss: 0.8801
validation loss: 0.8782
epoch : 46
training loss: 0.8795
validation loss: 0.8775
epoch : 47
training loss: 0.8790
validation loss: 0.8770
epoch : 48
training loss: 0.8786
validation loss: 0.8767
epoch : 49
training loss: 0.8782
validation loss: 0.8764
epoch : 50
training loss: 0.8779
validation loss: 0.8761
epoch : 51
training loss: 0.8776
validation loss: 0.8750
epoch : 52
training loss: 0.8766
validation loss: 0.8748
epoch : 53
training loss: 0.8763
validation loss: 0.8745
epoch : 54
training loss: 0.8759
validation loss: 0.8741
epoch : 55
training loss: 0.8757
validation loss: 0.8740
epoch : 56
training loss: 0.8754
validation loss: 0.8737
epoch : 57
training loss: 0.8751
validation loss: 0.8733
epoch : 58
training loss: 0.8748
validation loss: 0.8731
epoch : 59
```

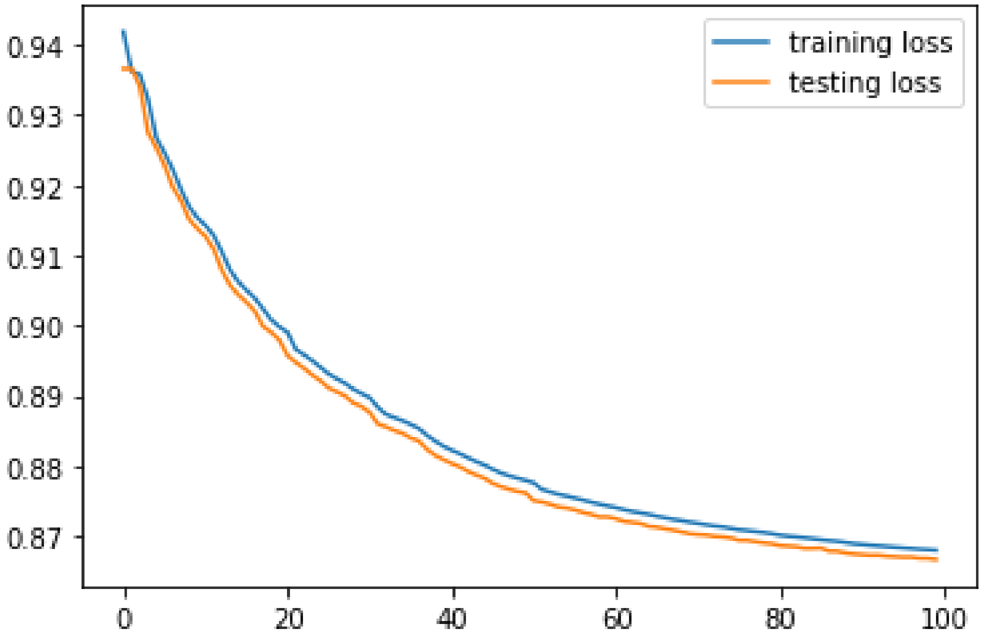
```
training loss: 0.8745
validation loss: 0.8727
epoch : 60
training loss: 0.8743
validation loss: 0.8727
epoch : 61
training loss: 0.8740
validation loss: 0.8725
epoch : 62
training loss: 0.8737
validation loss: 0.8721
epoch : 63
training loss: 0.8735
validation loss: 0.8719
epoch : 64
training loss: 0.8733
validation loss: 0.8718
epoch : 65
training loss: 0.8731
validation loss: 0.8714
epoch : 66
training loss: 0.8728
validation loss: 0.8712
epoch : 67
training loss: 0.8726
validation loss: 0.8710
epoch : 68
training loss: 0.8724
validation loss: 0.8708
epoch : 69
training loss: 0.8722
validation loss: 0.8706
epoch : 70
training loss: 0.8720
validation loss: 0.8703
epoch : 71
training loss: 0.8718
validation loss: 0.8702
epoch : 72
training loss: 0.8716
validation loss: 0.8701
epoch : 73
training loss: 0.8714
validation loss: 0.8699
epoch : 74
training loss: 0.8713
validation loss: 0.8699
epoch : 75
```

```
training loss: 0.8711
validation loss: 0.8697
epoch : 76
training loss: 0.8709
validation loss: 0.8694
epoch : 77
training loss: 0.8708
validation loss: 0.8694
epoch : 78
training loss: 0.8706
validation loss: 0.8692
epoch : 79
training loss: 0.8705
validation loss: 0.8690
epoch : 80
training loss: 0.8703
validation loss: 0.8689
epoch : 81
training loss: 0.8701
validation loss: 0.8686
epoch : 82
training loss: 0.8700
validation loss: 0.8685
epoch : 83
training loss: 0.8699
validation loss: 0.8685
epoch : 84
training loss: 0.8697
validation loss: 0.8682
epoch : 85
training loss: 0.8696
validation loss: 0.8682
epoch : 86
training loss: 0.8695
validation loss: 0.8683
epoch : 87
training loss: 0.8693
validation loss: 0.8678
epoch : 88
training loss: 0.8692
validation loss: 0.8678
epoch : 89
training loss: 0.8691
validation loss: 0.8676
epoch : 90
training loss: 0.8689
validation loss: 0.8674
epoch : 91
```

```
training loss: 0.8688
validation loss: 0.8673
epoch : 92
training loss: 0.8687
validation loss: 0.8672
epoch : 94
training loss: 0.8685
validation loss: 0.8671
epoch : 95
training loss: 0.8684
validation loss: 0.8670
epoch : 96
training loss: 0.8683
validation loss: 0.8670
epoch : 97
training loss: 0.8682
validation loss: 0.8670
epoch : 98
training loss: 0.8681
validation loss: 0.8668
epoch : 99
training loss: 0.8680
validation loss: 0.8668
epoch : 100
training loss: 0.8680
validation loss: 0.8666
```

[12]:
```
def plot_loss(training_losses, testing_losses):
  plt.plot(training_losses, label='training loss')
  plt.plot(testing_losses, label='testing loss')
  plt.legend()
```

[13]:
```
plot_loss(training_losses, test_losses)
```



1.2.2 Now let's see how well this autoencoder regenerates the images in the MNIST validation dataset

We'll input 20 images from the validation set and see the outputs

[14]:
```
def im_convert(tensor):
  image = tensor.cpu().clone().detach().numpy()
  image = image.transpose(1, 2, 0)
  image = image * np.array((0.5, 0.5, 0.5)) + np.array((0.5, 0.5, 0.5))
  image = image.clip(0, 1)
  return image
```

[15]:
```
def output_convert(image):
  img = image.view(28, 28)
  img = img.cpu().clone().detach().numpy()
  return img
```

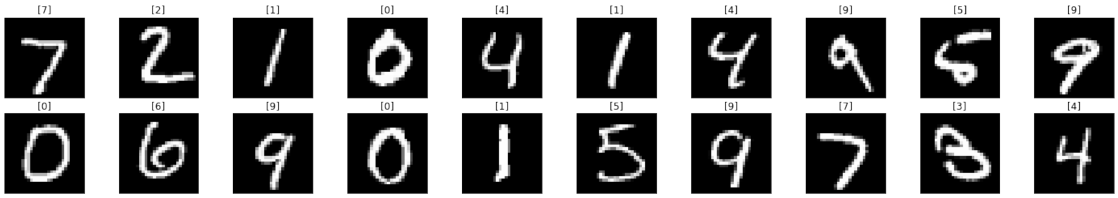
[16]:
```
data = iter(validation_loader)
images, labels = data.next()
```

The input images

[17]:
```
fig = plt.figure(figsize=(25, 4))

for idx in np.arange(20):
  ax = fig.add_subplot(2, 10, idx+1, xticks=[], yticks=[])
  plt.imshow(im_convert(images[idx]))
```

```
  ax.set_title([labels[idx].item()])
```

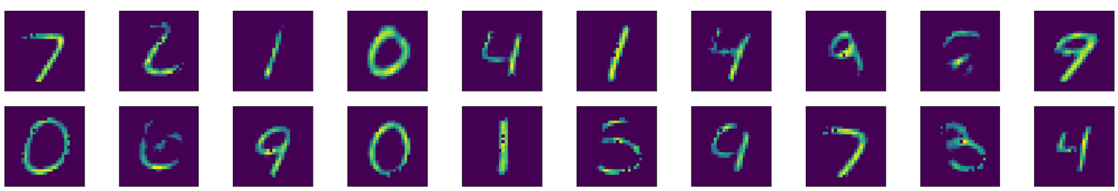


[18]:
```
imgs = images.to(device)
imgs = imgs.view(imgs.shape[0], -1)
reconstructed = model.forward(imgs)
```

The output images

[19]:
```
fig = plt.figure(figsize=(25, 4))

for idx in np.arange(20):
  ax = fig.add_subplot(2, 10, idx+1, xticks=[], yticks=[])
  plt.imshow(output_convert(reconstructed[idx]))
```



[ ]:
```

```