CS 559: Neural Networks

Homework 6

Author:
Gaetano Coppoletta
Email:
gcoppo2@uic.edu

November 3, 2022

1 A

Autoencoders are Neural Networks which are commonly used for feature selection and extraction. When we have more nodes in the inner layer than input, the risk is that we have a useless autoencoder, because the output could be equal to the input. Denoising Autoencoders solve this problem by corrupting the data on purpose by randomly turning some of the input values to zero. The percentage of input nodes set to zero depends on the amount of data and on how many input nodes we have in our neural network. The purpose of the denoising autoencoder is to identify and remove noise from our data.

2 B

Batch normalization is a method used to make training of neural networks faster and more stable through normalization of the layers' inputs by re-centering and re-scaling. The distribution of each layer's inputs changes during training, as the parameters of the previous layers change and this complicates the training of neural networks. This slows down the training by requiring lower learning rates and careful parameter initialization. We refer to this phenomenon as internal covariate shift. The strenght of this normalization is making it a part of the model architecture and performing the normalization for each training mini-batch. Batch Normalization allows us to use much higher learning rates and be less careful about initialization.

3 C

We want to generate nine random images of digits arranged in a 3x3 matrix. In order to do this we create nine random tensor with shape (1,4) and we pass them to the decoder which will generate the images, then we concatenate them in order to create a 3x3 matrix. The image as we can see contains a lot of noise, this is because it has been generated from a random tensor. The result is shown below.

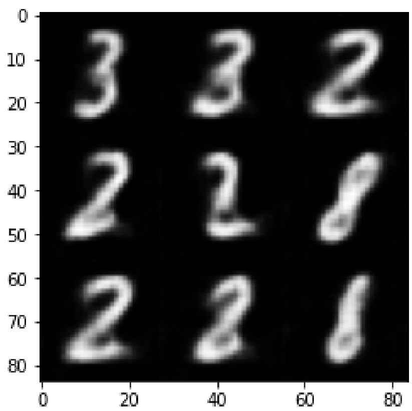

Figure 1: Matrix with 9 random images representing digits

4 D

In this section we want to cluster images representing digits. To do this we use the k-means algorithm, looking for 10 clusters. We use a dataset containing 48000 images with the corresponding labels. After the k-means we have to find the difference between the assignment found by k-means and the true labels in order to compute the accuracy. In order to do this we create a 10x10 matrix, where the index of the row corresponds to the true label (extracted from the dataset) and the column index corresponds to the label assigned by k-means. Scanning the matrix we compute the number of agreements for each cluster mapping each true label class. After that we have to find the maximum for each row and map the index, we can probably map two indexes to the same cluster, to solve this we scan the array that maps the indexes and if we find two equal values, we find which one maps the highest value in the matrix and we leave it as it is, while the lowest value will no longer be considered a maximum (is set to 0), so another maximum is found in the same row of the matrix and its index is mapped. After this we scan again the map vector because the new index could be already assigned. After that we can check if our index reassignment is correct computing the accuracy. With this algorithm the accuracy obtained is 80.03%.

5 Code

```

rnd_images=[]
for i in range(9):
  with torch.no_grad():
    tensor=torch.rand((1,4))
    image  = decoder(tensor).detach().squeeze().numpy()
    rnd_images.append(image)

matrix_with_images=np.zeros((28*3,28*3))
img_num=0
for i in range(3):
  for j in range(3):
    for pos_x in range(28):
      for pos_y in range(28):
        matrix_with_images[i*28 + pos_x][j*28 + pos_y]=rnd_images[img_num][pos_x
            ][pos_y]
    img_num+=1
plt.imshow(matrix_with_images, cmap='gist_gray')

# put your clustering accuracy calculation here
train_loader2 = torch.utils.data.DataLoader(train_data, batch_size=1)
output_of_encoder = []
labels=[]
for image,label in train_loader2:
  output = encoder(image).detach().squeeze().numpy()
  output_of_encoder.append(output)
  labels.append(label)

```

```
from sklearn.cluster import KMeans

output_of_encoder_numpy = np.array(output_of_encoder)
kmeans = KMeans(n_clusters=10).fit(output_of_encoder_numpy)

matrix = np.zeros((10,10))
for i in range(10):
  for j in range(10):
    for z in range(48000):
      if labels[z].item()==i and kmeans.labels_[z]==j:
        matrix[i][j]+=1

for i in range(10):
  max_index = np.argmax(matrix[i,:])
  map[i]=max_index
print(map)
i=0
j=0
while i<10:
  while j<10:
      if map[i]==map[j] and i!=j:
          if matrix[i][int(map[i])]>matrix[j][int(map[j])]:
              #reassign map[j]
              matrix[j][int(map[j])]=0
              map[j]=np.argmax(matrix[j,:])
              i=0
              j=0
      j+=1
  i+=1


print(map)
#compute accuracy
count=0
for i in range(10):
  for j in range(48000):
    if kmeans.labels_[j]==map[i] and labels[j].item()==i:
      count+=1
accuracy = count/48000 *100
print("The accuracy is: ",accuracy,"%")
```