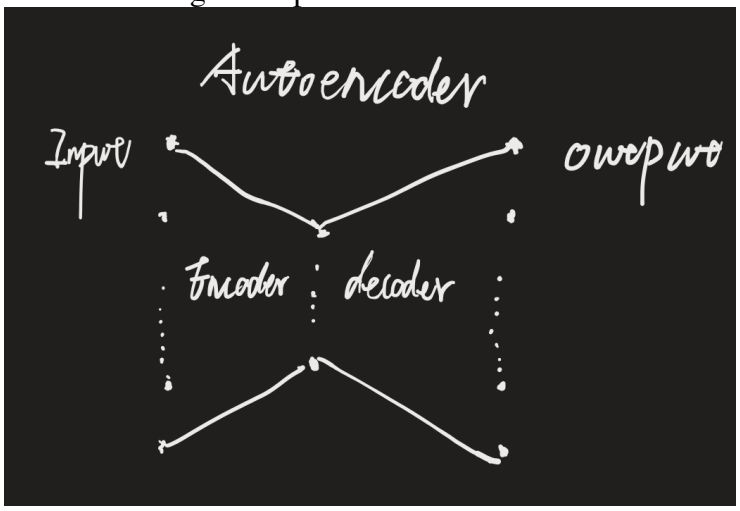
# CS559 homework 6

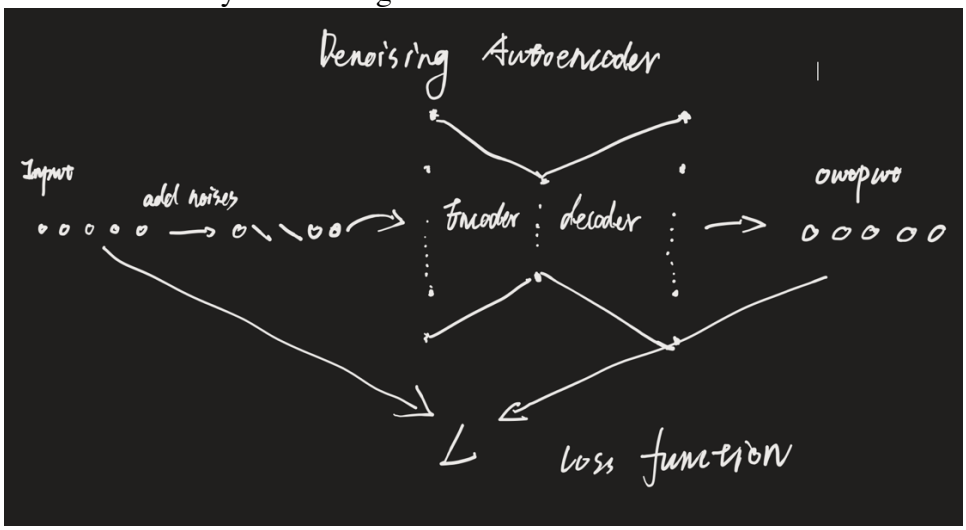
655490960
Huiyang Zhao

(a) Explain how the denoising autoencoder works.

Autoencoder can be considered as a neural network that is trained so that the output is the same as input. A schematic diagram is provided below:




Encoder: this part of network downsamples the input data to a lower dimension representation.
Decoder: this part of network rebuilt the original input data from the lower dimension representation.
In this process, high-dimensional data are transferred to low-dimensional data, and then got recovered. However, this process is also lossy in nature, which means that the output will be less accuracy.
**Denoising Autoencoder**: adding noises such as applying dropout to the input, so that after decoding there will be less noises. The network will also be more robust.
The network is trained by minimizing the loss function L.



(b) The model involves batch normalization. Learn what batch normalization is and briefly explain here.

Batch normalization is a method of normalizing the inputs by recentering and rescaling, thus making network's training faster and more stable.
A BN layer first calculates the mean and variance value of activation values, and using these values to normalize data so that each neuron's output follows a standard normal distribution.
Adding BN layers often leads to faster convergence. On the other hand, with BN layers we can use larger learning rate.
In Pytorch, batch normalization is implemented by torch.nn.BatchNorm2d, which works as follows:

$$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta$$ [1]

The mean and standard-deviation are calculated per-dimension over the mini-batches and γ and β are learnable parameter vectors of size C (where C is the input size). By default, the elements of γ are set to 1 and the elements of β are set to 0. The standard-deviation is calculated via the biased estimator, equivalent to torch.var(input, unbiased=False).[2]

(c) Write a script (to be appended to the end of the script - see the file) that generates 9 random images of digits arranged in a 3 × 3 matrix. Labels are not important. Include the images with your report. Comment on the results.

The images generated are attached below:

Comments:
It's surprising that randomly generated values can results in such images which are so similar to real digits. We can observe that some features of 'digits' are included in the images, such as rounded corner, combination of lines and circle, and combination of lines of different directions.

[1] https://pytorch.org/docs/stable/generated/torch.nn.BatchNorm2d.html BATCHNORM2D from Pytorch Docs.
[2] https://pytorch.org/docs/stable/generated/torch.nn.BatchNorm2d.html BATCHNORM2D from Pytorch Docs.

(d) Write a script that clusters the images in the dataset (to be appended at the end of the script - see the file). …… Report the accuracy you obtain over the training set, and describe your index reassignment algorithm.

The accuracy I got is 71.5%.

My reassignment algorithm is pretty similar to that of professor's, with a little improvement:

assignments: a 10*10 array consists of the number of agreements for each cluster mapping each true label class.
matches: an array of ten numbers that represents the label a cluster is assigned. -1 if not assigned a label.

```
while (not all clusters are assigned a label):
   for i in range(n):
      if (matches[i]!=-1):
          continue;
      max_match = 0
      for j in range(n):
          if assignment[i][j] > assignments[i][max_match]:
             if j is not assigned:
                map j to i
             else:
                if assignment[i][j] > number of agreements in previous assignments:
                    map j to i
                    discard previous assignments
```

The main point is: when conflicts happen, choose a better assignment.
The detailed code is attached.

```
# CS559 Neural Network
# Huiyang Zhao
# UIN 655490960
from sklearn.cluster import KMeans


# put your image generator here
def image_generate(decoder, n=3):
    images = torch.randn(9, 4)
    with torch.no_grad():
        decoded_image = decoder(images.to(device))

    plt.figure()
    for i in range(n):
        # ax = plt.subplot(3, n, i + 1)
        for j in range(n):
            img = decoded_image[i * 3 + j]
            ax = plt.subplot(3, n, j + 1 + i * n)
            plt.imshow(img.cpu().squeeze().numpy(), cmap='gist_gray')
            ax.get_xaxis().set_visible(False)
            ax.get_yaxis().set_visible(False)
    plt.savefig('generated_images.pdf')
    plt.show()
    return decoded_image


decoded_image = image_generate(decoder)

# put your clustering accuracy calculation here
kmeans = KMeans(init='k-means++', n_init=10, n_clusters=10, max_iter=60)
kmeans_loader = torch.utils.data.DataLoader(train_data,
batch_size=train_data.__len__())
for image_batch, labels in kmeans_loader:
    image_batch = image_batch.to(device)
    kmeans.fit(encoder(image_batch).cpu().detach().numpy())

assignments = []
for i in range(10):
    index_label = np.where(labels == i)
    assignment = [-1]
    for j in range(10):
        index_cluster = np.where(kmeans.labels_ == j)

        assignment.append(len(np.intersect1d(index_label, index_cluster)))
    assignments.append(assignment)

is_matched = [False, False, False, False, False, False, False, False, False,
False]
matches = [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
matches_row = [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1]

while all(is_matched) is False:
    for i in range(10):
        if matches[i] != -1:
            continue
        max_match = 0
        for j in range(1, 11):
```

```
            # print('curr: ' + str(assignments[i][j]) + ' max: ' +
str(assignments[i][max_match]))
            if assignments[i][j] > assignments[i][max_match]:
                if is_matched[j - 1] is False:
                    max_match = j - 1
                    matches_row[max_match] = i
                    max_match = j
                else:
                    if assignments[i][j] > assignments[matches_row[j -
1]][j]:
                        is_matched[j - 1] = False
                        matches[matches_row[j - 1]] = -1
                        max_match = j - 1
                        matches_row[max_match] = i
                        max_match = j
        is_matched[max_match - 1] = True
        matches[i] = max_match - 1

total_correct = 0
for index in range(10):
    total_correct = total_correct + assignments[index][matches[index] + 1]
    print(str(index) + ' is mapped to ' + str(matches[index]))

print('Accuracy: ' + '{:.1%}'.format(total_correct / 48000))
```