1. Consider an autoencoder which has one input layer, one hidden layer and one output layer. There is a weight connecting every neuron in the input layer to every neuron in the hidden layer and similarly there is a weight connecting every neuron in the hidden layer to every neuron in the output layer. There are no bias parameters in the network. Now, if the input to the network is $x \in \mathbb{R}^{10}$ and the total number of weights in the network is 160, then what kind of autoencoder is this ?

   A. Overcomplete autoencoder

   B. Undercomplete autoencoder

   C. Can't say because the size of the hidden layer in not known

Solution: First, let us note that since the input to the network is $x \in \mathbb{R}^{10}$, the output of the network will also be $\mathbb{R}^{10}$. This is simply because the output layer of the autoencoder tries to reconstruct the input and hence its dimensions should be the same as that of the input. Now let the number of neurons in the hidden layer be k. Since there is a weight connecting every neuron in the input layer to every neuron in the hidden layer, the total number of weights in the input layer will be $10 * k$. Similarly, the total number of weights in the input layer will be $10 * k$. Thus, we have,

$$10 * k + 10 * k = 160$$
$$\therefore k = 8$$

Since the number of neurons in the hidden layer is less than the dimension of the input this is an undercomplete autoencoder. Hence, **Option B** is the correct answer.

2. In the lecture we saw that an autoencoder is equivalent to PCA if we use a linear encoder, a linear decoder, the squared error loss function and standardize the elements of the input matrix in a certain way. Keeping this in mind, what would the following matrix look like after its elements are standardized.

$$\begin{bmatrix} 1 & 6 & 12 \\ 3 & 12 & 24 \\ 5 & 18 & 72 \\ 7 & 48 & 36 \end{bmatrix}$$

A. $\begin{bmatrix} -3 & -15 & -24 \\ -1 & -9 & -12 \\ 1 & -3 & 36 \\ 3 & 27 & 0 \end{bmatrix}$

B. $\begin{bmatrix} 0.5 & 3 & 6 \\ 1.5 & 6 & 12 \\ 2.5 & 9 & 36 \\ 3.5 & 24 & 18 \end{bmatrix}$

C. $\begin{bmatrix} -1.5 & -7.5 & -12 \\ -0.5 & -4.5 & -6 \\ 0.5 & -1.5 & 18 \\ 1.5 & 13.5 & 0 \end{bmatrix}$

Solution: Option C is the correct answer. As mentioned on Slide 16 of Lecture 7, we need to first compute the mean of every column. We then need to subtract the mean from every element of the column. Finally, we need to divide every element by the square root of the number of elements in that column. If we follow this procedure we will get the matrix given in Option C. For example, the mean of the second column is 21. If we subtract the mean from every element of this column we get the values $[-15, -9, -3, 27]$. We now need to divide each entry by the square root of the number of data points, *i.e.*, the square root of the number of rows (square root of, by $\sqrt{4} = 2$) to get $[-7.5, -4.5, -1.5, 13.5]$.

3. Consider a dataset containing m black and white images *i.e.*, every pixel in the image is either black or white. Further, assume that each image contains k pixels and x_{ij} denotes the j-th pixel of the i-th image. Suppose you want to train an autoencoder using this data such that the autoencoder takes an image x as input and reconstructs it as $\hat{x}$. Which of the following is the most appropriate loss function to use in this case?

   A.

$$\sum_{i=1}^{m}\sum_{j=1}^{k}(x_{ij} - \hat{x}_{ij})$$

   B.

$$\sum_{i=1}^{m}\sum_{j=1}^{k}(x_{ij} - \hat{x}_{ij})^2$$

   C.

$$-\sum_{i=1}^{m}\sum_{j=1}^{k}(x_{ij}\log\hat{x}_{ij} + (1 - x_{ij})\log(1 - \hat{x}_{ij}))$$

   D.

$$-\sum_{i=1}^{m}\log\hat{x}_{ij}$$

Solution: Each pixel can either be black (0) or white (1) and hence the input data is binary, i.e., $x \in (0,1)^k$. We discussed in class that for such binary inputs the most appropriate loss function is the Bernoulli cross entropy loss function given by

$$-\sum_{i=1}^{m}\sum_{j=1}^{k}(x_{ij}\log\hat{x}_{ij} + (1 - x_{ij})\log(1 - \hat{x}_{ij}))$$

Hence, *Option C* is the correct answer.

4. We saw that we can use L2 regularization along with the reconstruction loss to regularize autoencoders. The total loss function then contains two parts

$$\sum_{i=1}^{m}\sum_{j=1}^{k}(x_{ij}-\hat{x}_{ij})^2+\lambda||\theta||^2$$

where θ is a collection of all the parameters of the network .The second term, *i.e.*, $\lambda||\theta||^2$ is the L2-regularization loss and the first term is the reconstruction loss. Suppose you use the same training data containing m points and train the autoencoder using different values of λ. For example, you train one autoencoder using $\lambda = 1$, another using $\lambda = 10$ and another using $\lambda = 100$. What do you expect will happen to the reconstruction loss as you increase the value of λ. Assume that the data is a complex real world dataset (for example, images uploaded on facebook, instagram, *etc*).

A. the reconstruction loss will be unaffected as λ only gets multiplied by $||\theta||^2$

B. the reconstruction loss will increase as λ increases

C. the reconstruction loss will decrease as λ increases

Solution: *Option B* is the correct answer. As the value of λ increases the weightage given to the second term, *i.e.*, $||\theta||^2$ increases. Hence, the emphasis will be on minimizing $||\theta||^2$ even at the expense of the reconstruction loss. Hence, the reconstruction loss will increase.

5. We have trained two autoencoders on a small dataset containing 1000 training points where each point $\in \mathbb{R}^{50}$. Each of these autoencoders has one input layer, one hidden layer and one output layer. Each autoencoder has the same number of neurons in the hidden layer and uses the following encoder and decoder functions.

$$h = \frac{1}{1 + e^{-(W_e x + b)})}$$
$$x = \frac{1}{1 + e^{-(W_d h + c)})}$$

where, $x \in \mathbb{R}^{50}$, $W_e \in \mathbb{R}^{20\times 50}$ is the weight matrix for the input layer (*i.e.*, it contains the weights connecting the input layer to the hidden layer). $b \in R^{20}$ contains the biases for the hidden layer (one per neuron in the hidden layer). Similarly, $W_d \in \mathbb{R}^{50\times 20}$ is the weight matrix for the output layer (*i.e.*, it contains the weights connecting the hidden layer to the output layer). Finally $c \in R^{50}$ contains the biases for the hidden layer (one per neuron in the hidden layer). We have trained the two autoencoders using two different algorithms and hence we have different values for W_e, W_d, b and c. You can download the weights learned by the two networks and load them using the following code:

```python
# Load packages
import numpy as np

# Load weights for autoencoder 1
parameters1 = np.load('autoencoder1.npy')
W_e_1 = parameters1[0]
b_1   = parameters1[1]
W_d_1 = parameters1[2]
c_1   = parameters1[3]

# Load weights for autoencoder 2
parameters2 = np.load('autoencoder2.npy')
W_e_2 = parameters2[0]
b_2   = parameters2[1]
W_d_2 = parameters2[2]
c_2   = parameters2[3]
```

You can download the training data and the weights from this URL :
https://drive.google.com/drive/folders/1pbEyovRAA4OBxqPxOkyefwitXRvR-4SO?usp=sharing.
You need to find out which autoencoder gives a smaller squared error loss on the training data.

A. the loss of autoencoder1 < the loss of autoencoder2

B. the loss of autoencoder1 > the loss of autoencoder2

C. the loss of autoencoder2 = the loss of autoencoder1

Solution:

```
# Load packages
import numpy as np
import pandas as pd

# Define the sigmoid function
def sigmoid(x):
    return 1.0 / (1.0 + np.exp(-x))

# Load weights for autoencoder 1
parameters1 = np.load('autoencoder1.npy')
W_e_1 = parameters1[0]
b_1   = parameters1[1]
W_d_1 = parameters1[2]
c_1   = parameters1[3]


# Load weights for autoencoder 2
parameters2 = np.load('autoencoder2.npy')
W_e_2 = parameters2[0]
b_2   = parameters2[1]
W_d_2 = parameters2[2]
c_2   = parameters2[3]


# Load data
X = pd.read_csv("data.csv").values

loss_1 = 0
loss_2 = 0

# Iterate over each data point
for x_i in X:

    # compute hidden representation for autoencoder1
    h_1 = sigmoid(np.dot(W_e_1, x_i) + b_1)

    # compute hidden representation for autoencoder2
    h_2 = sigmoid(np.dot(W_e_2, x_i) + b_2)

    # compute reconstructed output for autoencoder1
    x_hat_1 = sigmoid(np.dot(W_d_1, h_1) + c_1)

    # compute reconstructed output for autoencoder2
    x_hat_2 = sigmoid(np.dot(W_d_2, h_2) + c_2)

    # compute square error loss between input and reconstructed output
    of autoencoder1
    error_1 = np.mean(np.square(x_i - x_hat_1))

    # add it to the total loss
    loss_1 = loss_1 + error_1

    # compute square error loss between input and reconstructed output
    of autoencoder2
    error_2 = np.mean(np.square(x_i - x_hat_2))

    # add it to the total loss
    loss_2 = loss_2 + error_2

# Compare the two losses
if loss_1 < loss_2:
    print("The loss of autoencoder1 < The loss of autoencoder2")
elif loss_1 > loss_2:
    print("The loss of autoencoder1 > The loss of autoencoder2")
else:
    print("The loss of autoencoder1 = The loss of autoencoder2")
```

∴ **Option A** is the correct answer.

6. This question is in the context of contractive autoencoders. Consider an autoencoder where the input $x \in \mathbb{R}^3$ and the hidden layer $h \in \mathbb{R}^2$. Suppose, we use the following encoder function:

$$h = Wx + b$$

where, $x \in \mathbb{R}^3, h \in \mathbb{R}^2, W \in \mathbb{R}^{2\times 3}, b \in \mathbb{R}^2$. Further, let us assume that the network has the following parameters:

$$W = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 1 & 4 \end{bmatrix}, b = \begin{bmatrix} 1 \\ 3 \end{bmatrix}$$

The ij-th entry of the matrix W, *i.e.*, the element in the i-th row and j-th column is the weight connecting the j-th input to the i-th neuron in the hidden layer. Which of the following matrices is the Jacobian matrix for this autoencoder (refer to the definition of the Jacobian matrix as discussed in the lecture).

A. $\begin{bmatrix} 2 & 5 \\ 3 & 4 \\ 2 & 7 \end{bmatrix}$

B. $\begin{bmatrix} 2 & 3 & 2 \\ 5 & 4 & 7 \end{bmatrix}$

C. $\begin{bmatrix} 1 & 2 \\ 2 & 1 \\ 1 & 4 \end{bmatrix}$

D. $\begin{bmatrix} 1 & 2 & 1 \\ 2 & 1 & 4 \end{bmatrix}$

Solution: The Jacobian matrix contains the partial derivative of every hidden neuron w.r.t. every input neuron. In the above case, the Jacobian will be a 2×3 matrix as shown below :

$$J_{\mathbf{x}}(\mathbf{h}) = \begin{bmatrix} \frac{\partial h_1}{\partial x_1} & \frac{\partial h_1}{\partial x_2} & \frac{\partial h_1}{\partial x_3} \\ \\ \frac{\partial h_2}{\partial x_1} & \frac{\partial h_2}{\partial x_2} & \frac{\partial h_2}{\partial x_3} \end{bmatrix}$$

First let us compute the partial derivative $\frac{\partial h_1}{\partial x_1}$. We observer that,

$$h_1 = w_{11}x_1 + w_{12}x_2 + w_{13}x_3 + b_1$$
$$\therefore \frac{\partial h_1}{\partial x_1} = w_{11}$$

In general,

$$\frac{\partial h_i}{\partial x_j} = w_{ij}$$

Hence, we have,

$$J_{\mathbf{x}}(\mathbf{h}) = \begin{bmatrix} \frac{\partial h_1}{\partial x_1} & \frac{\partial h_1}{\partial x_2} & \frac{\partial h_1}{\partial x_3} \\ \\ \frac{\partial h_2}{\partial x_1} & \frac{\partial h_2}{\partial x_2} & \frac{\partial h_2}{\partial x_3} \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ \\ w_{21} & w_{22} & w_{23} \end{bmatrix} = \begin{bmatrix} 1 & 2 & 1 \\ \\ 2 & 1 & 4 \end{bmatrix}$$

Hence, **Option D** is the correct answer