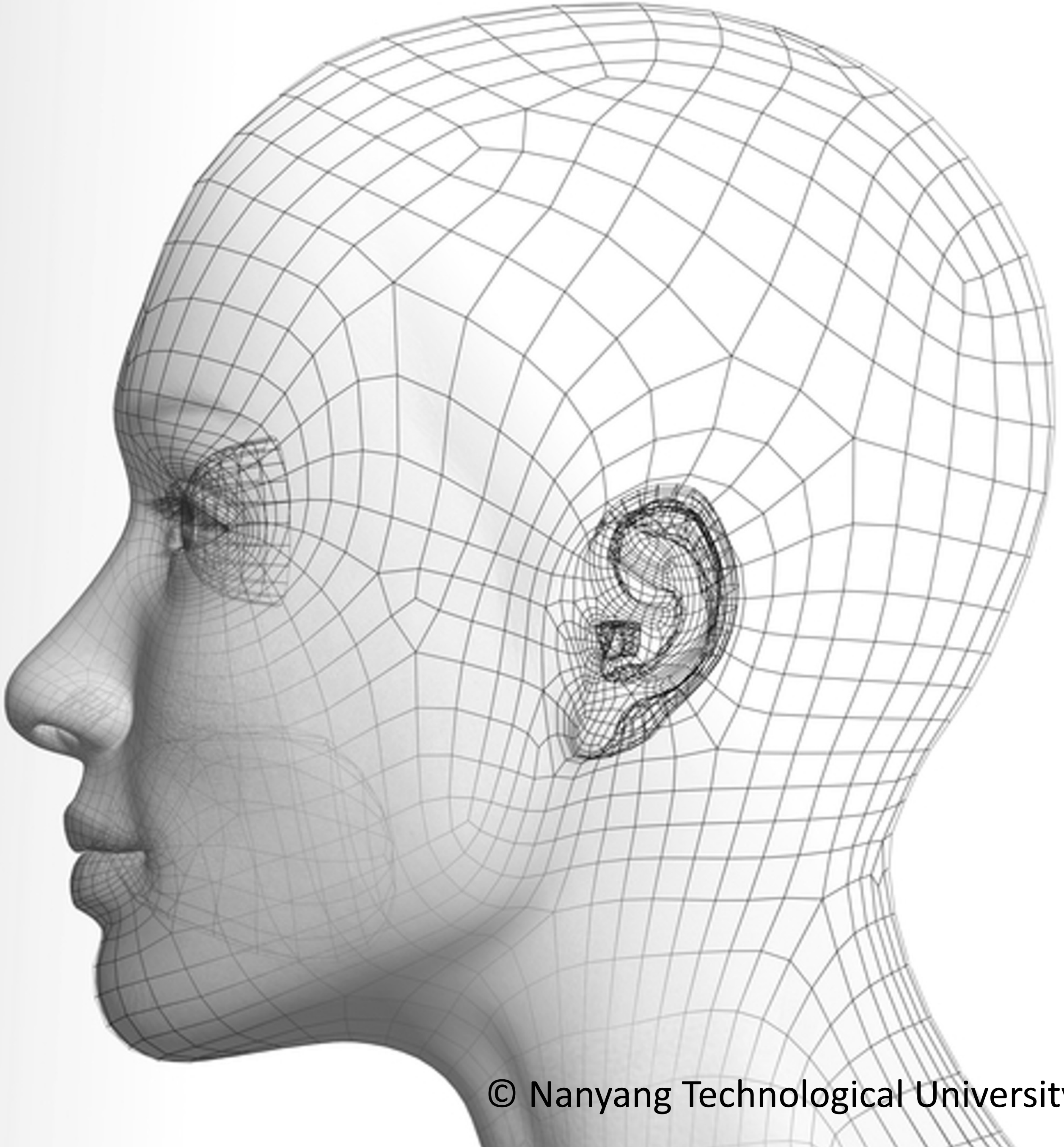
Tutorial 10
Autoencoders

Chen-Change Loy
吕健勤

https://www.mmlab-ntu.com/person/ccloy/

https://twitter.com/ccloy



Question 1a

Given five binary patterns:

$$\boldsymbol{x}_1 = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \boldsymbol{x}_2 = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}, \boldsymbol{x}_3 = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}, \boldsymbol{x}_4 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \boldsymbol{x}_5 = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}$$

Design an autoencoder with four hidden neurons to reconstruct the patterns, using gradient descent learning with a learning parameter $\alpha = 0.1$.
Find the weights, biases, hidden-layer activations and reconstructions of the input patterns at convergence.

Repeat the above by introducing a sparsity constraint with a penalty parameter $\beta = 0.5$ and sparsity parameter $\rho = 0.1$.

t10q1a.ipynb

[Recap] Training autoencoders

If the inputs are interpreted as bit vectors or vectors of bit probabilities, cross-entropy of the reconstruction can be used:

$$J_{cross-entropy} = -\sum_{p=1}^{P} \left(\boldsymbol{x}_p \log \boldsymbol{y}_p + \left(1 - \boldsymbol{x}_p\right) \log\left(1 - \boldsymbol{y}_p\right) \right)$$

Learning are done by using gradient descent:

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \alpha \nabla_{\boldsymbol{W}} J$$
$$\boldsymbol{b} \leftarrow \boldsymbol{b} - \alpha \nabla_{\boldsymbol{b}} J$$
$$\boldsymbol{c} \leftarrow \boldsymbol{c} - \alpha \nabla_{\boldsymbol{c}} J$$

Question 1

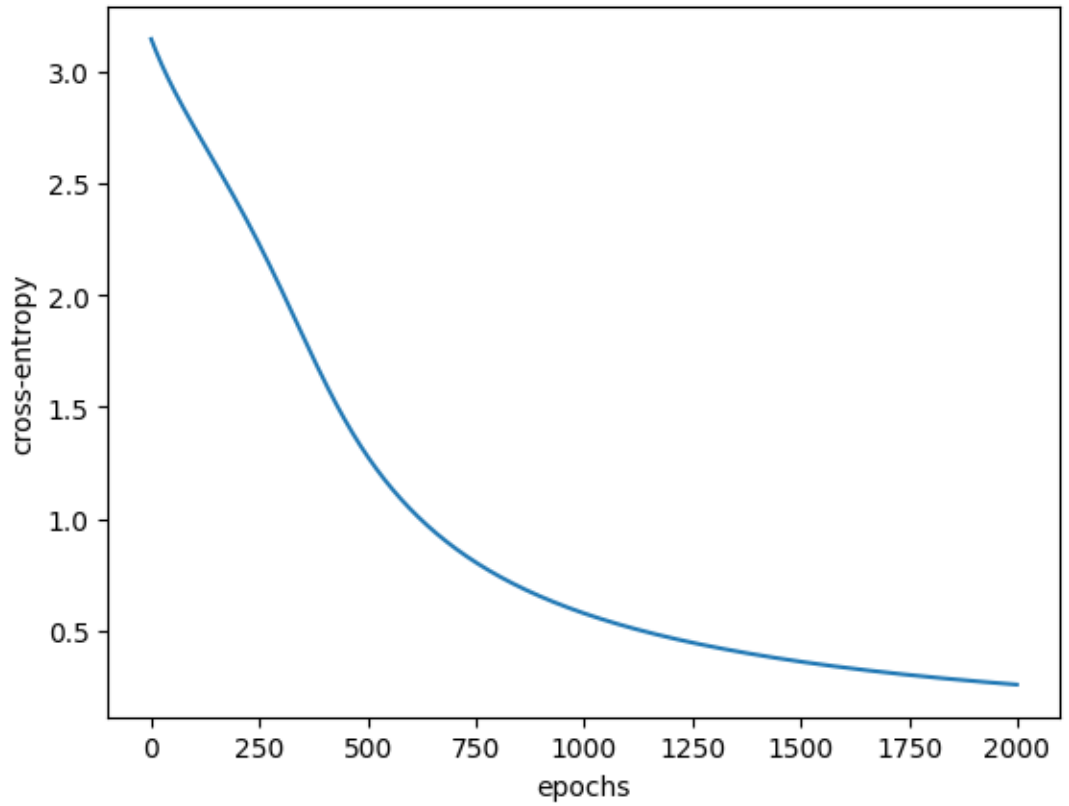
```
[17] # Display weights and biases
     print(f'W:\n {autoencoder.W.data}\n')
     print(f'W_prime:\n {autoencoder.W_prime.data}\n')
     print(f'b:\n {autoencoder.b.data}\n')
     print(f'b_prime:\n {autoencoder.b_prime.data}\n')
```

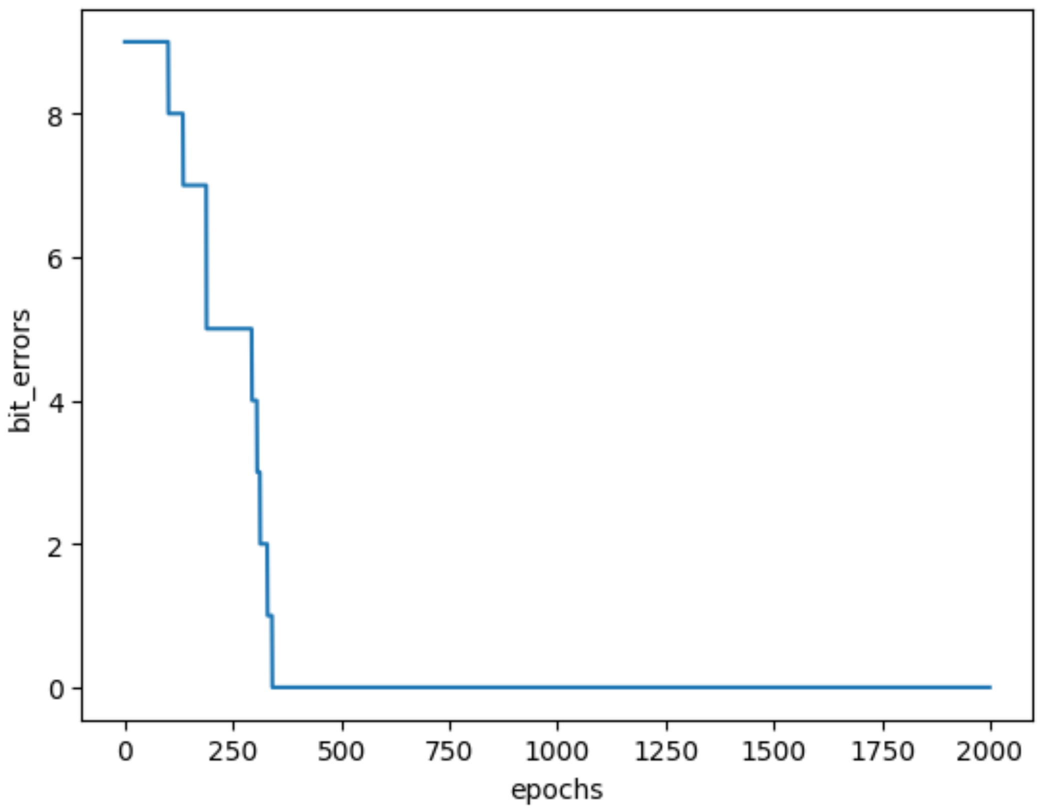
```
W:
 tensor([[ 1.3404,  1.4979,  1.1027, -1.6822],
        [ 0.7539, -0.4255,  0.8947, -2.8855],
        [-0.6857, -2.6170,  2.0785, -0.5724],
        [ 2.4732, -0.0064, -2.3399,  1.0806],
        [-3.8564,  0.2799,  1.9536,  1.9443],
        [ 0.0641,  0.3798,  0.0269, -0.2577],
        [ 1.1192, -2.2494, -0.7006,  2.4961],
        [ 0.4663,  1.2535, -2.7378,  1.2046],
        [-0.8174,  3.8036, -1.1091,  0.7902]])

W_prime:
 tensor([[ 4.5253,  0.8765, -2.3588,  4.4418, -6.2649, -1.2592,  1.4582,  0.7749,
         -2.4911],
        [ 5.3236, -2.2052, -6.1462,  0.9732,  0.2411, -1.1617, -4.2555,  3.2673,
          6.7966],
        [-1.0380,  2.4406,  4.2446, -5.9221,  2.3335, -1.6839, -3.9068, -6.1726,
         -2.6724],
        [-2.4691, -6.0329, -2.7513,  1.6180,  3.1008, -1.5105,  5.9788,  0.7012,
          1.2132]])

b:
 tensor([-0.4316, -0.8084,  1.3958,  0.3283])

b_prime:
 tensor([ 0.9000, -0.0059,  1.2902, -0.8071, -1.4364, -2.9330,  0.3787, -2.1170,
        -1.8984])
```

```
[18] # Evaluate result of reconstruction
     with torch.no_grad():
         h, y, o = autoencoder(X)
     print(f'Input:\n {X}\n')
     print(f'Hidden activation:\n {h}\n')
     print(f'Output:\n {y}\n')
     print(f'Output_binary:\n {o}\n')
```

```
Input:
 tensor([[1., 1., 1., 0., 0., 0., 0., 0., 0.],
        [1., 0., 0., 1., 0., 0., 1., 0., 0.],
        [0., 0., 1., 0., 1., 0., 1., 0., 0.],
        [1., 0., 0., 0., 1., 0., 0., 0., 1.],
        [1., 0., 0., 1., 0., 0., 1., 1., 1.]])

Hidden activation:
 tensor([[0.7265, 0.0868, 0.9958, 0.0081],
        [0.9890, 0.1727, 0.3677, 0.9023],
        [0.0207, 0.0045, 0.9912, 0.9852],
        [0.0226, 0.9916, 0.9659, 0.7991],
        [0.9845, 0.9704, 0.0123, 0.9855]])

Output:
 tensor([[9.7331e-01, 9.4381e-01, 9.6258e-01, 3.2928e-02, 2.6134e-02, 3.5483e-03,
         5.8771e-02, 6.0390e-04, 3.1119e-03],
        [9.7554e-01, 1.6859e-02, 4.6263e-02, 9.5418e-01, 1.9171e-02, 1.7244e-03,
         9.9359e-01, 8.1404e-02, 4.4102e-02],
        [7.9918e-02, 2.8694e-02, 9.3763e-01, 6.7819e-03, 9.7818e-01, 2.1902e-03,
         9.1739e-01, 5.4520e-04, 3.3148e-02],
        [9.6463e-01, 9.6007e-03, 4.9424e-02, 1.5239e-02, 9.6747e-01, 9.6071e-04,
         5.7109e-02, 1.3909e-02, 9.5981e-01],
        [9.9969e-01, 7.4719e-04, 6.4053e-05, 9.9761e-01, 1.3577e-02, 1.1024e-03,
         9.7149e-01, 9.1922e-01, 9.6794e-01]])

Output_binary:
 tensor([[1., 1., 1., 0., 0., 0., 0., 0., 0.],
        [1., 0., 0., 1., 0., 0., 1., 0., 0.],
        [0., 0., 1., 0., 1., 0., 1., 0., 0.],
        [1., 0., 0., 0., 1., 0., 0., 0., 1.],
        [1., 0., 0., 1., 0., 0., 1., 1., 1.]])
```

Question 1


3.0
2.5
2.0
1.5
1.0
0.5
cross-entropy
0
250
500
750
1000
1250
1500
1750
2000
epochs


8
6
4
2
0
bit_errors
0
250
500
750
1000
1250
1500
1750
2000
epochs

Question 1

Given five binary patterns:

$$\boldsymbol{x}_1 = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \boldsymbol{x}_2 = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}, \boldsymbol{x}_3 = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}, \boldsymbol{x}_4 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \boldsymbol{x}_5 = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}$$

Design an autoencoder with four hidden neurons to reconstruct the patterns, using gradient descent learning with a learning parameter $\alpha = 0.1$.
Find the weights, biases, hidden-layer activations and reconstructions of the input patterns at convergence.

Repeat the above by introducing a sparsity constraint with a penalty parameter $\beta = 0.5$ and sparsity parameter $\rho = 0.1$.

t10q1b.ipynb

Question 1

```
[7] # Display weights and biases
    print(f'W:\n {autoencoder.W.data}\n')
    print(f'W_prime:\n {autoencoder.W_prime.data}\n')
    print(f'b:\n {autoencoder.b.data}\n')
    print(f'b_prime:\n {autoencoder.b_prime.data}\n')
```

```
W:
 tensor([[ 0.2693, -0.6963,  0.7908,  1.3101],
        [-0.8612, -0.7763,  2.1997, -0.2562],
        [-2.0359, -2.2188,  1.6613, -1.7386],
        [ 2.1135,  1.8982, -0.8923, -1.1621],
        [-2.6880, -2.7191, -2.8330,  0.7545],
        [ 0.0641,  0.3798,  0.0269, -0.2577],
        [ 1.0243,  0.4042, -0.9752, -3.5422],
        [-1.5969,  2.7576, -1.3955, -0.0790],
        [-1.9770,  1.5598, -2.8313,  2.1123]])

W_prime:
 tensor([[ 2.7544, -1.6692, -3.3834,  4.2995, -3.1130, -0.8501,  2.5228, -0.7688,
         -2.0668],
        [ 3.2955, -2.3571, -4.0995,  5.2586, -4.9278, -1.1138,  3.3158,  5.5399,
          3.4433],
        [ 2.4002,  5.8303,  5.0095, -2.9082, -4.7423, -1.2832, -3.0701, -3.7878,
         -5.6506],
        [ 3.7825, -0.9833, -3.6115, -1.7625,  1.6002, -1.0698, -4.3225, -0.7160,
          4.1070]])

b:
 tensor([-1.2380, -2.2536, -0.6383, -0.4749])

b_prime:
 tensor([-1.0660, -2.9856,  1.0981, -2.2263,  1.9893, -4.4020,  1.6365, -3.0606,
        -1.3419])
```

```
# Evaluate result of reconstruction
with torch.no_grad():
    h, y, o = autoencoder(X)
print(f'Input:\n {X}\n')
print(f'Hidden activation:\n {h}\n')
print(f'Output:\n {y}\n')
print(f'Output_binary:\n {o}\n')
```

```
Input:
 tensor([[1., 1., 1., 0., 0., 0., 0., 0., 0.],
        [1., 0., 0., 1., 0., 0., 1., 0., 0.],
        [0., 0., 1., 0., 1., 0., 1., 0., 0.],
        [1., 0., 0., 0., 1., 0., 0., 0., 1.],
        [1., 0., 0., 1., 0., 0., 1., 1., 1.]])

Hidden activation:
 tensor([[0.0205, 0.0026, 0.9823, 0.2388],
        [0.8974, 0.3436, 0.1525, 0.0205],
        [0.0071, 0.0011, 0.0581, 0.0067],
        [0.0036, 0.0162, 0.0040, 0.9759],
        [0.1971, 0.9752, 0.0026, 0.1375]])

Output:
 tensor([[9.0549e-01, 9.2173e-01, 9.9380e-01, 4.4885e-03, 8.5999e-02, 2.6298e-03,
         8.7004e-02, 9.5446e-04, 2.6120e-03],
        [9.5173e-01, 1.1840e-02, 6.5584e-02, 9.5070e-01, 3.9621e-02, 3.1252e-03,
         9.8883e-01, 8.0203e-02, 5.7781e-02],
        [2.9362e-01, 6.4900e-02, 7.9191e-01, 8.5439e-02, 8.4509e-01, 1.1085e-02,
         8.1012e-01, 3.6097e-02, 1.6060e-01],
        [9.3691e-01, 1.8599e-02, 7.6944e-02, 2.0678e-02, 9.6897e-01, 4.1843e-03,
         7.3666e-02, 2.4417e-02, 9.3653e-01],
        [9.6147e-01, 3.2267e-03, 1.7124e-02, 9.7066e-01, 3.8350e-02, 3.0001e-03,
         9.9154e-01, 8.8912e-01, 8.9647e-01]])

Output_binary:
 tensor([[1., 1., 1., 0., 0., 0., 0., 0., 0.],
        [1., 0., 0., 1., 0., 0., 1., 0., 0.],
        [0., 0., 1., 0., 1., 0., 1., 0., 0.],
        [1., 0., 0., 0., 1., 0., 0., 0., 1.],
        [1., 0., 0., 1., 0., 0., 1., 1., 1.]])
```

Question 1

Comparing the sparsity:

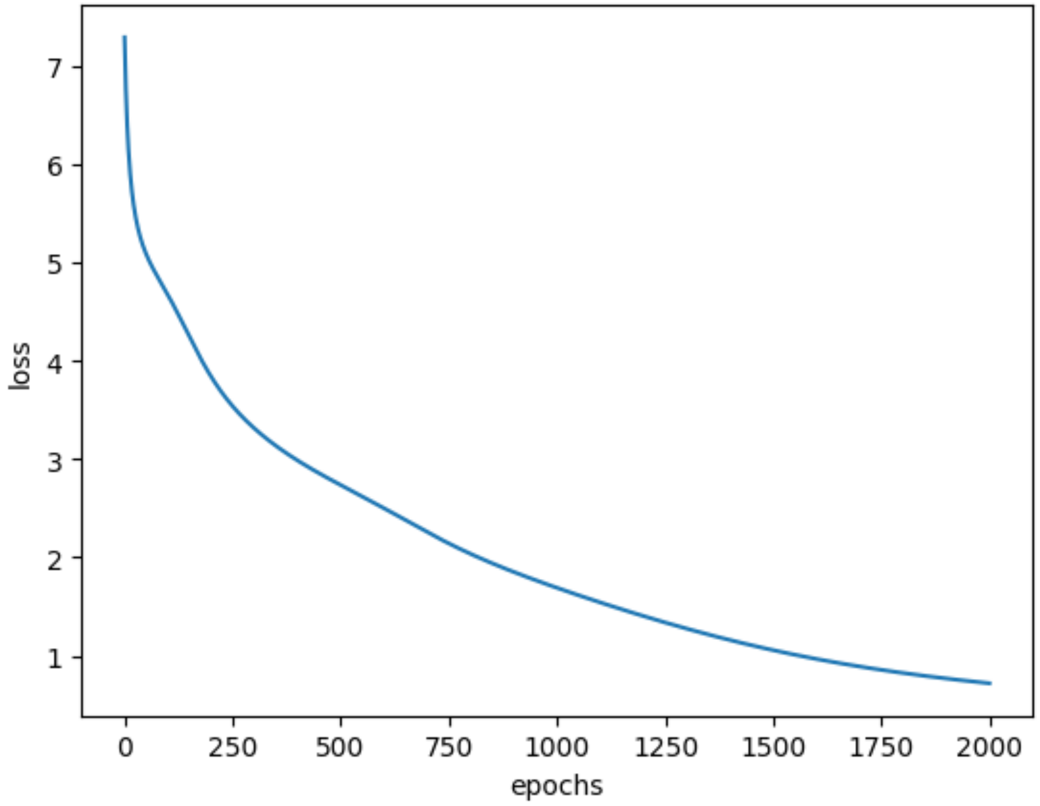
Reconstruction loss only

```
Hidden activation:
 tensor([[0.7265, 0.0868, 0.9958, 0.0081],
        [0.9890, 0.1727, 0.3677, 0.9023],
        [0.0207, 0.0045, 0.9912, 0.9852],
        [0.0226, 0.9916, 0.9659, 0.7991],
        [0.9845, 0.9704, 0.0123, 0.9855]])
```

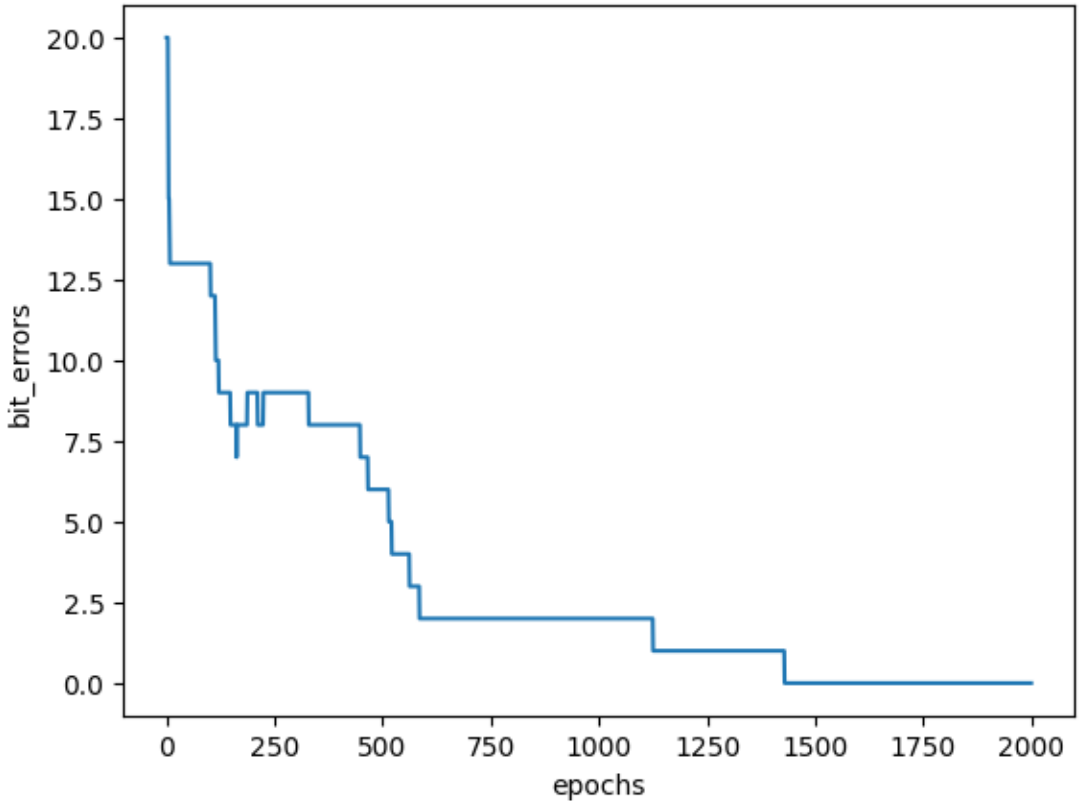
Reconstruction loss + Sparsity loss

```
Hidden activation:
 tensor([[0.0205, 0.0026, 0.9823, 0.2388],
        [0.8974, 0.3436, 0.1525, 0.0205],
        [0.0071, 0.0011, 0.0581, 0.0067],
        [0.0036, 0.0162, 0.0040, 0.9759],
        [0.1971, 0.9752, 0.0026, 0.1375]])
```

Question 1



7
6
5
4
3
2
1
loss
0
250
500
750
1000
1250
1500
1750
2000
epochs



20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
bit_errors
0
250
500
750
1000
1250
1500
1750
2000
epochs

Question 2

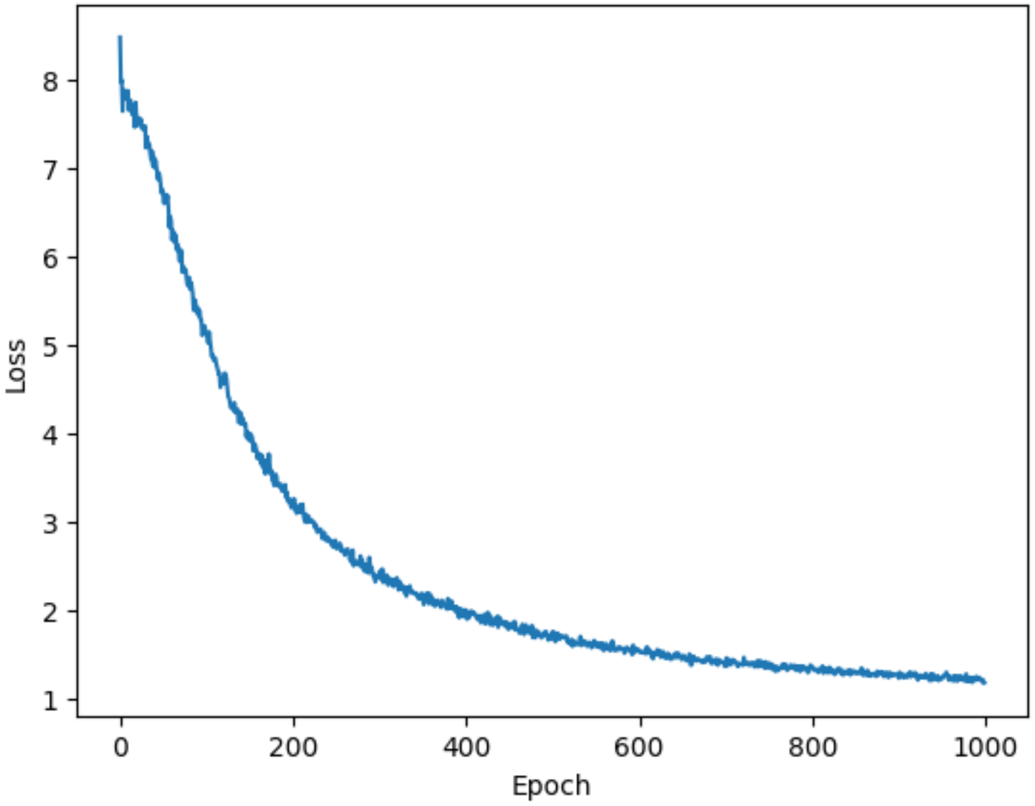
Create 100 images of 10x10 size by randomly generating pixel values between 0.0 and 1.0 from a uniform distribution.
Design the following autoencoders to reconstruct the input patterns, using mean square error as the cost function:

a. An undercomplete autoencoder with 49 hidden neurons
b. An overcomplete autoencoder with 144 hidden neurons
c. A sparse autoencoder with 144 hidden neurons and training with sparsity parameter $\rho = 0.05$ and penalty parameter $\beta = 0.5$.

Compare features learned by different autoencoders.

t10q2a.ipynb

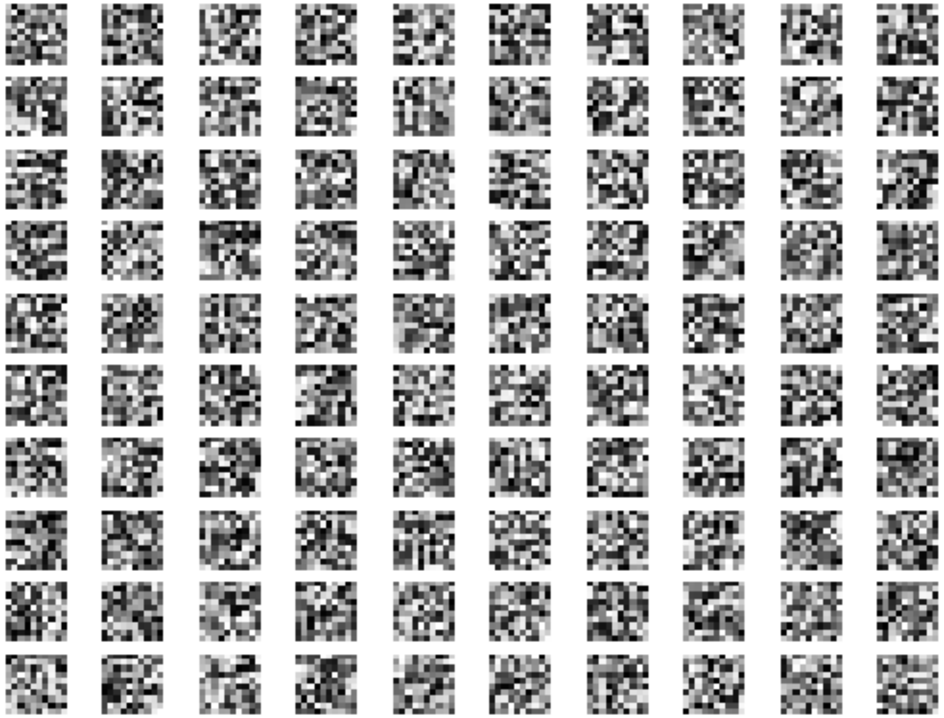
Question 2a



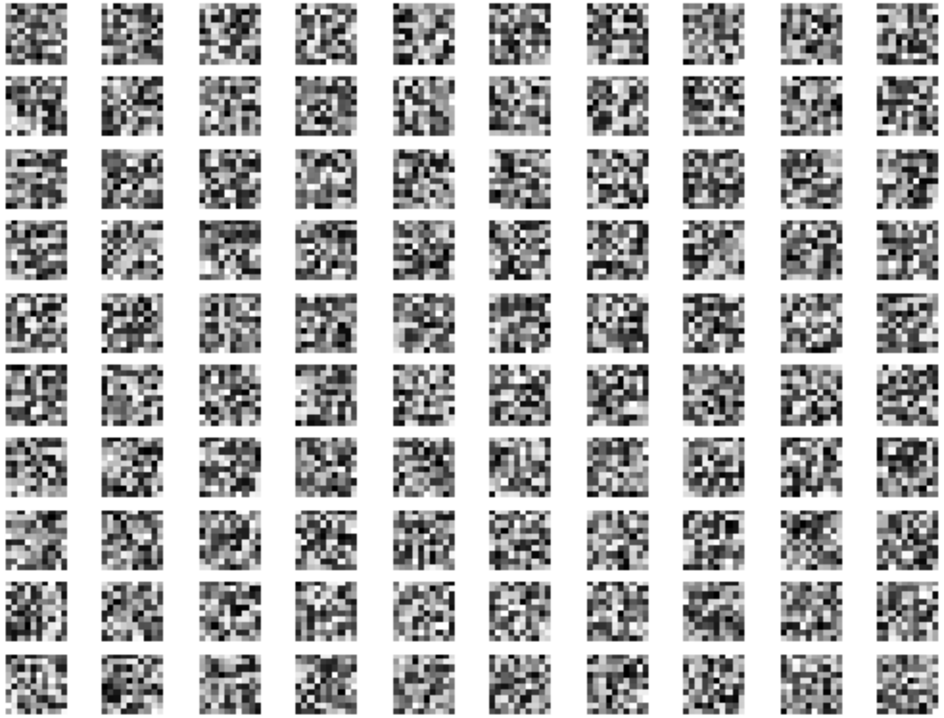
Question 2a

10x10 weights of 49 filters

Question 2a

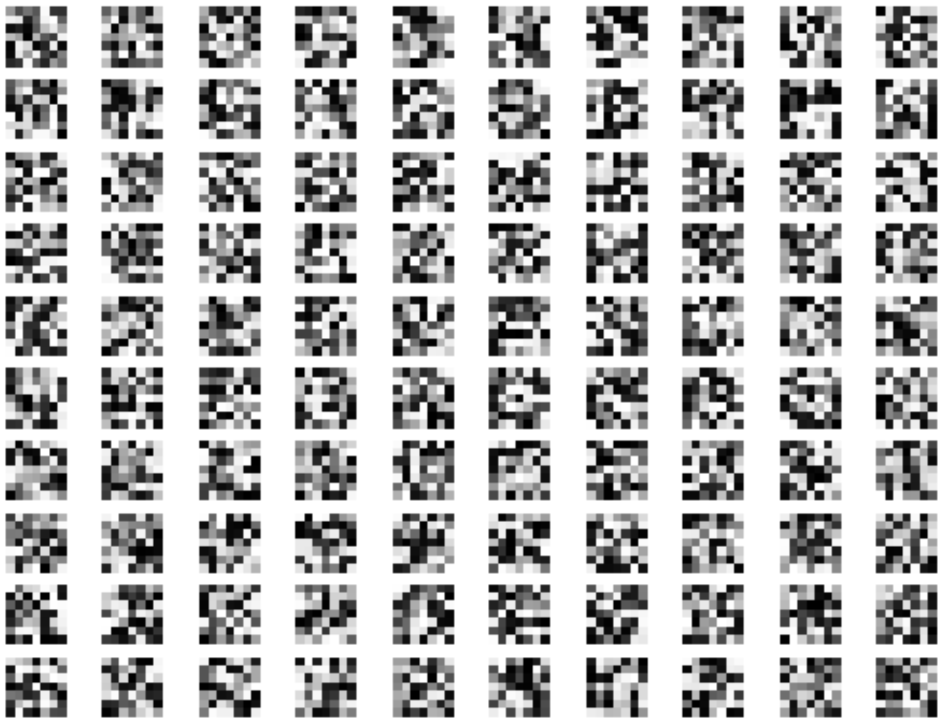


Input



Reconstruction

Question 2a



7x7 hidden activations (reshapred from 49 dimensions) of 100 samples

Question 2

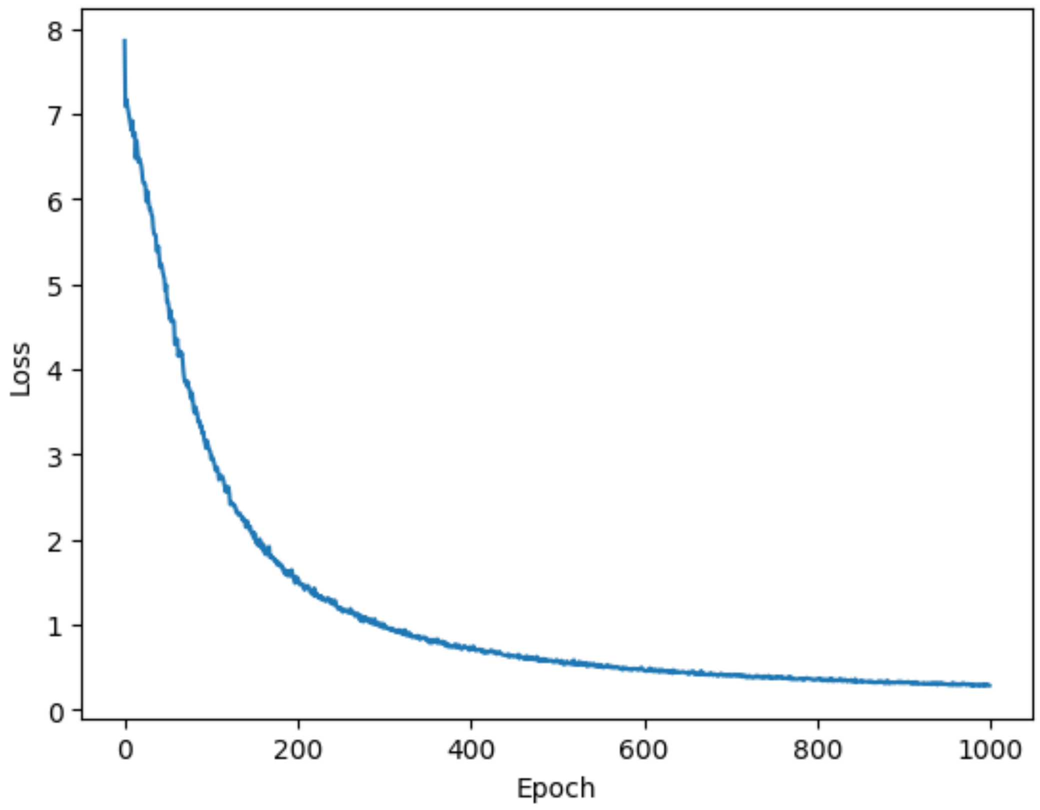
Create 100 images of 10x10 size by randomly generating pixel values between 0.0 and 1.0 from a uniform distribution.
Design the following autoencoders to reconstruct the input patterns, using mean square error as the cost function:

a. An undercomplete autoencoder with 49 hidden neurons
b. An overcomplete autoencoder with 144 hidden neurons
c. A sparse autoencoder with 144 hidden neurons and training with sparsity parameter $\rho = 0.05$ and penalty parameter $\beta = 0.5$.

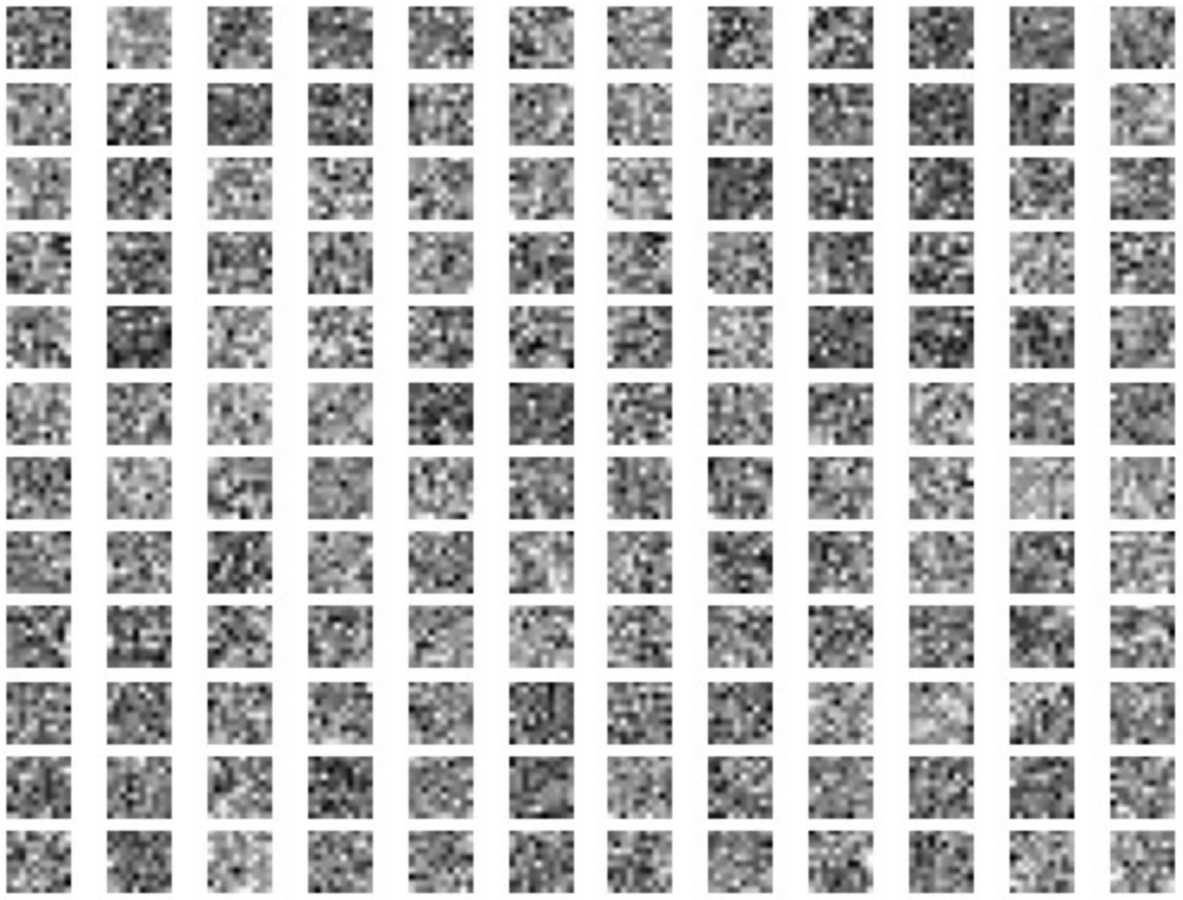
Compare features learned by different autoencoders.

t10q2b.ipynb

Question 2b

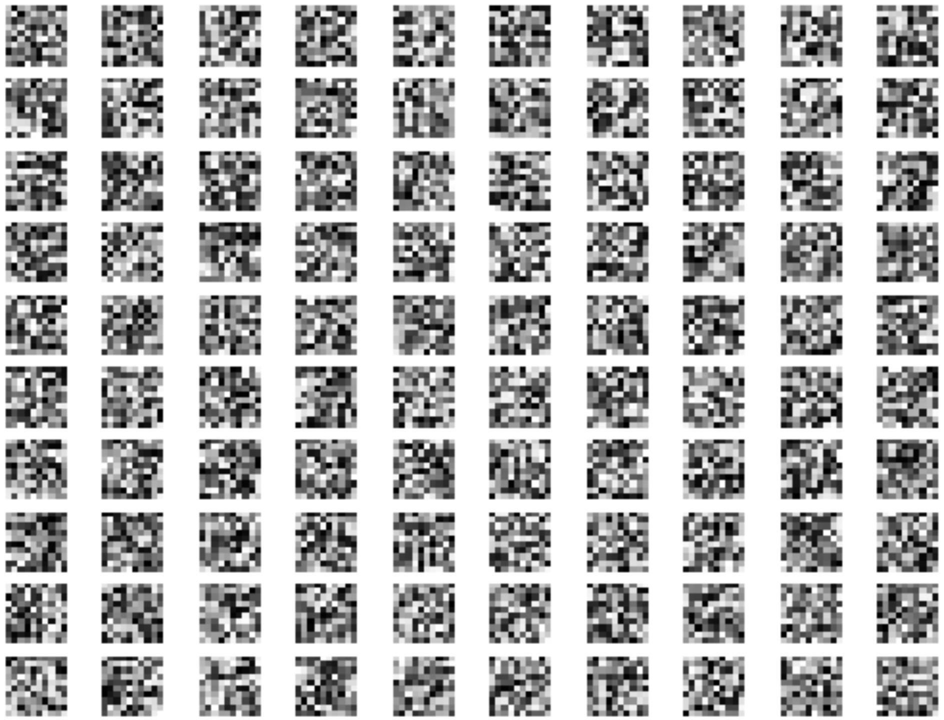


Question 2b

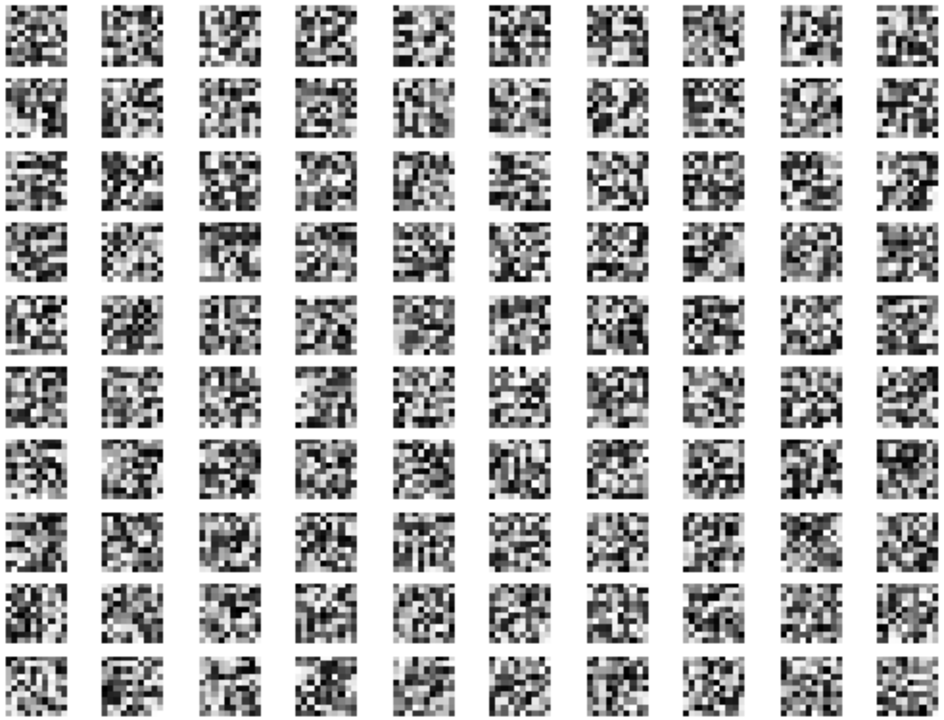


10x10 weights of 144 filters

Question 2b

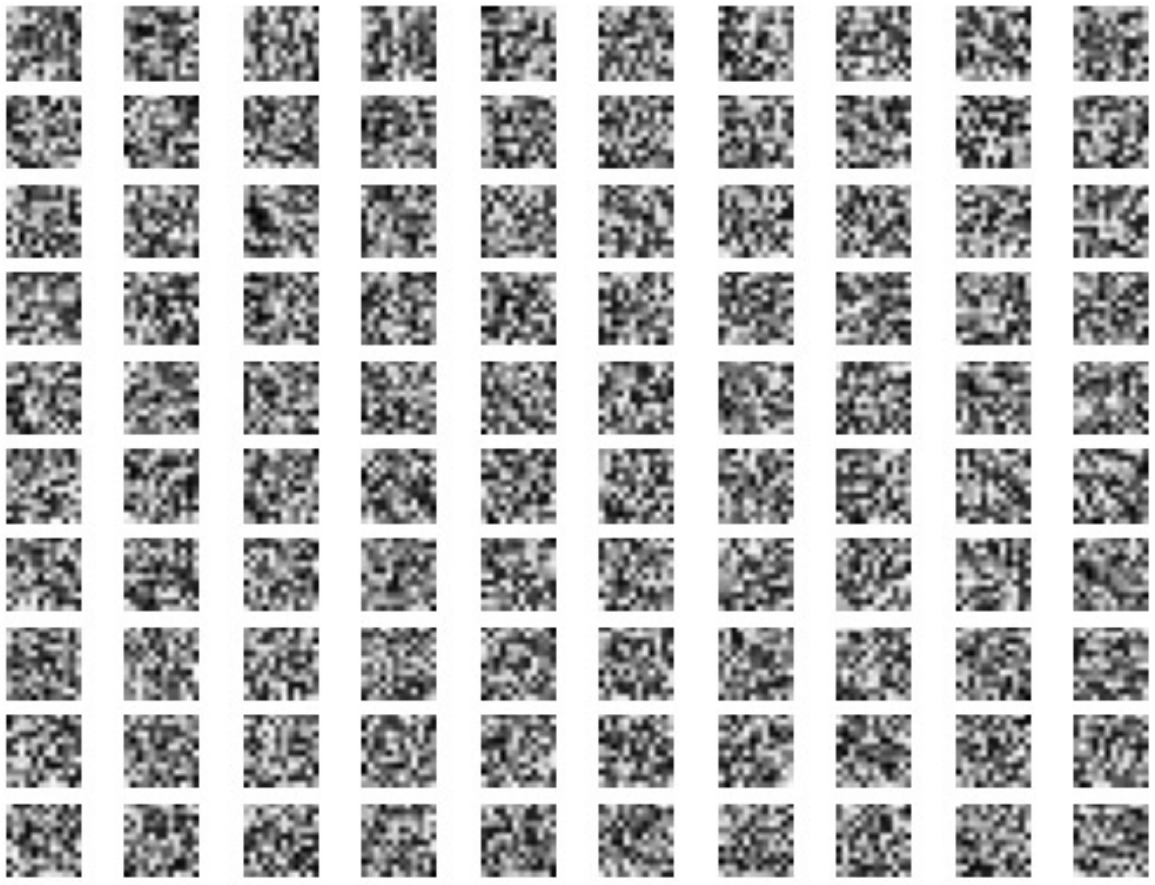


Input



Reconstruction

Question 2b



12x12 hidden activations (reshaped from 144 dimensions) of 100 samples

Question 2

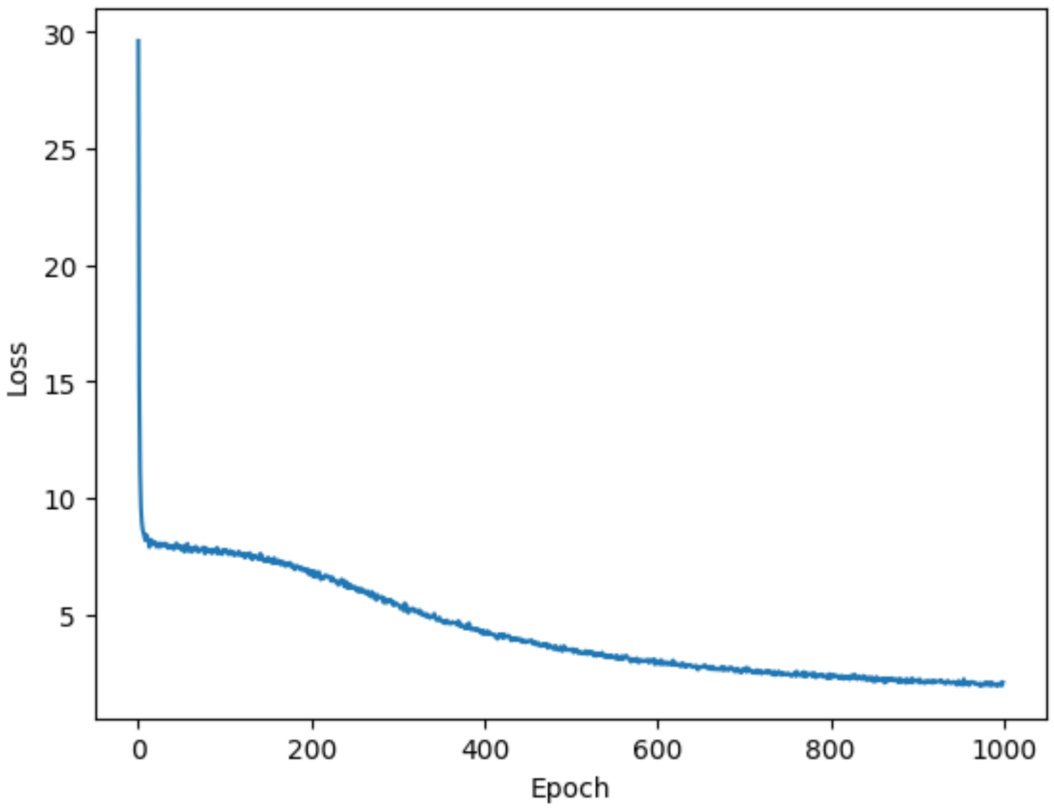
Create 100 images of 10x10 size by randomly generating pixel values between 0.0 and 1.0 from a uniform distribution.
Design the following autoencoders to reconstruct the input patterns, using mean square error as the cost function:

a. An undercomplete autoencoder with 49 hidden neurons
b. An overcomplete autoencoder with 144 hidden neurons
c. A sparse autoencoder with 144 hidden neurons and training with sparsity parameter $\rho = 0.05$ and penalty parameter $\beta = 0.5$.

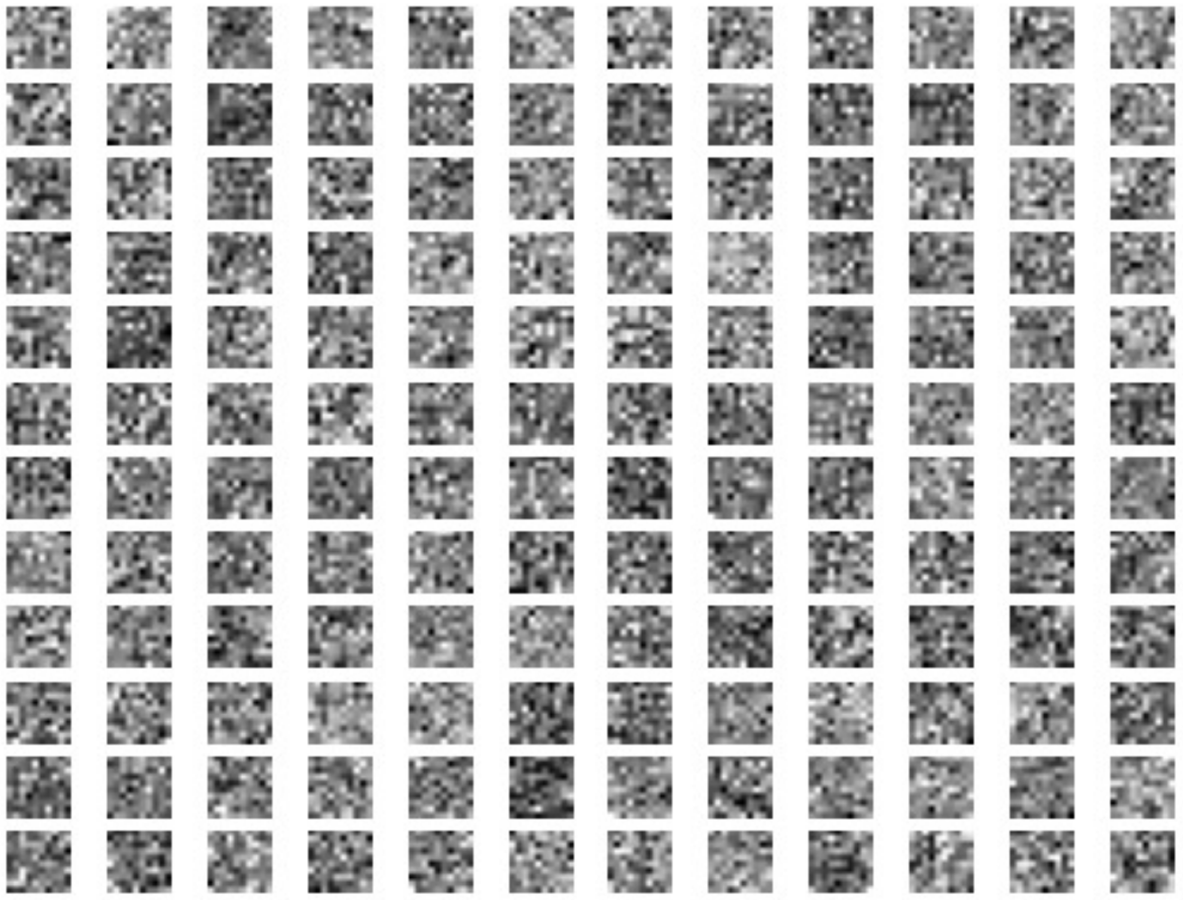
Compare features learned by different autoencoders.

t10q2c.ipynb

Question 2c

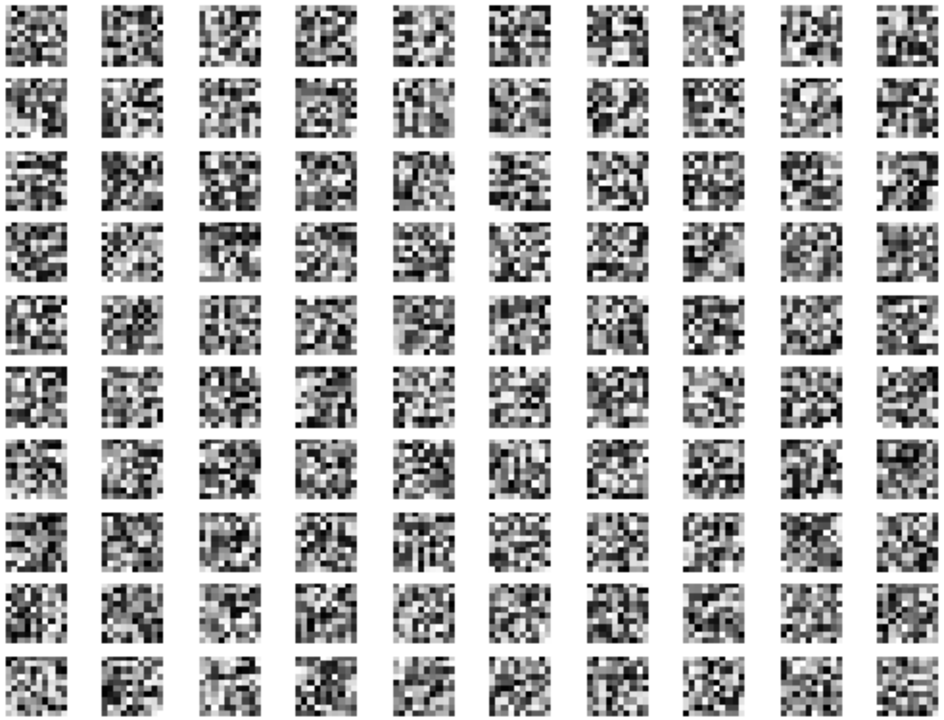


Question 2c

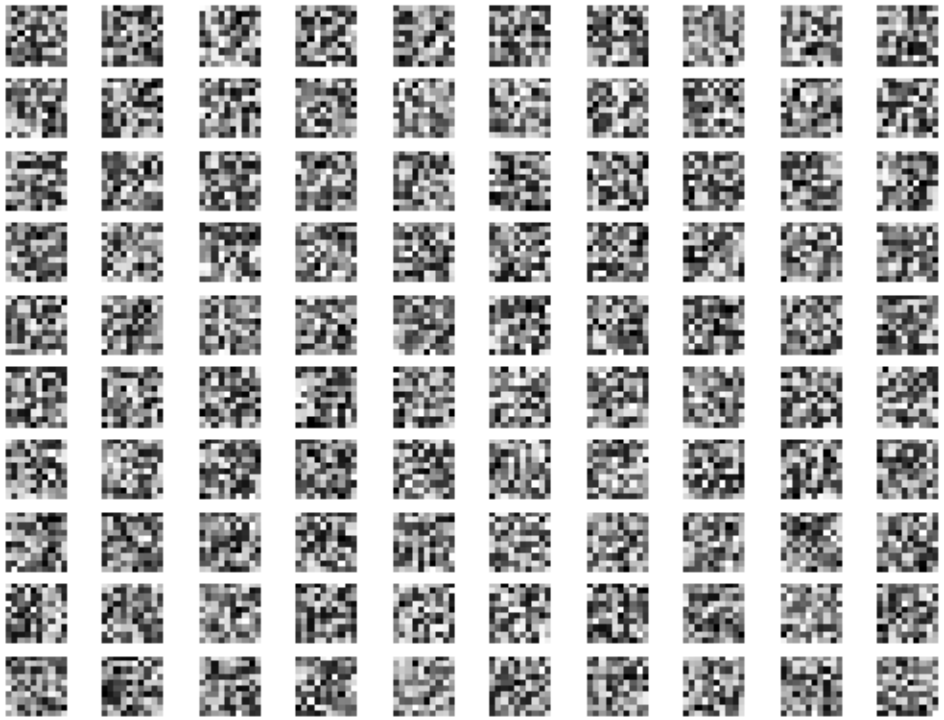


10x10 weights of 144 filters

Question 2c

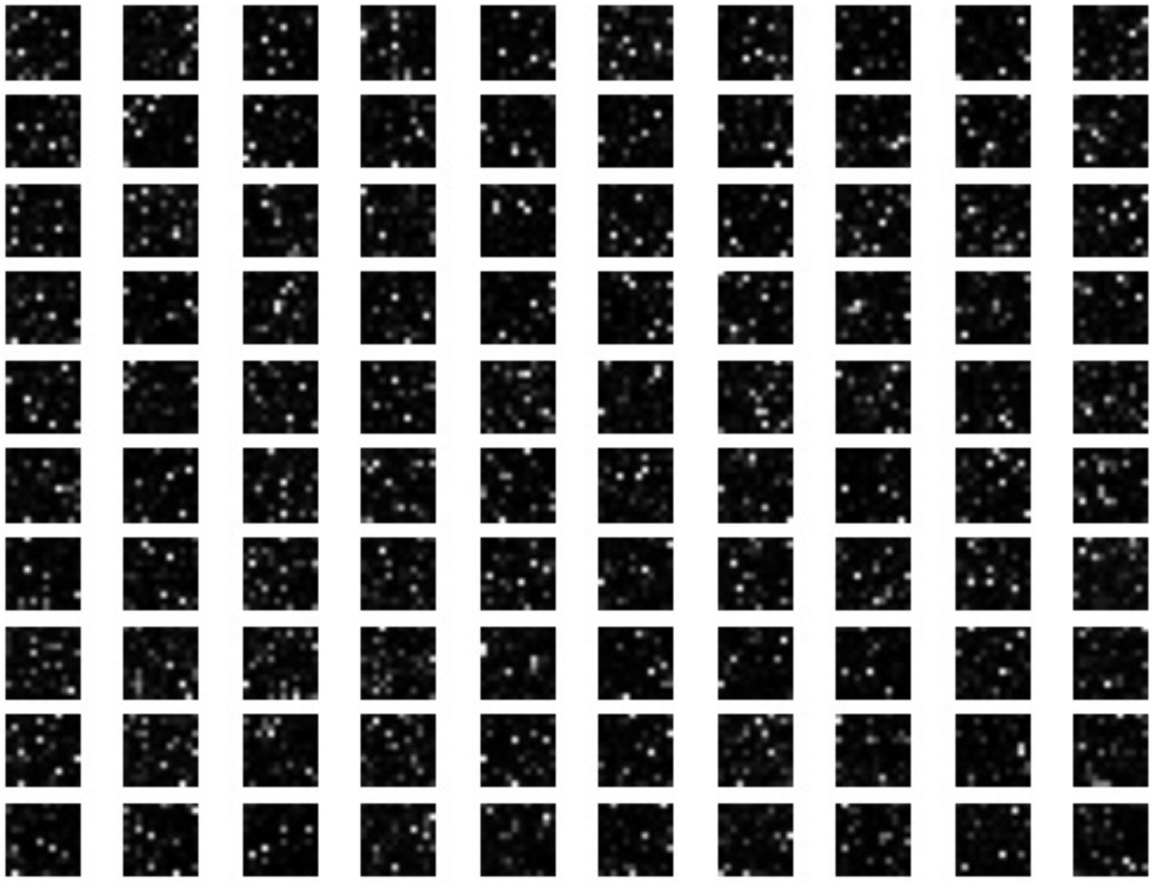


Input



Reconstruction

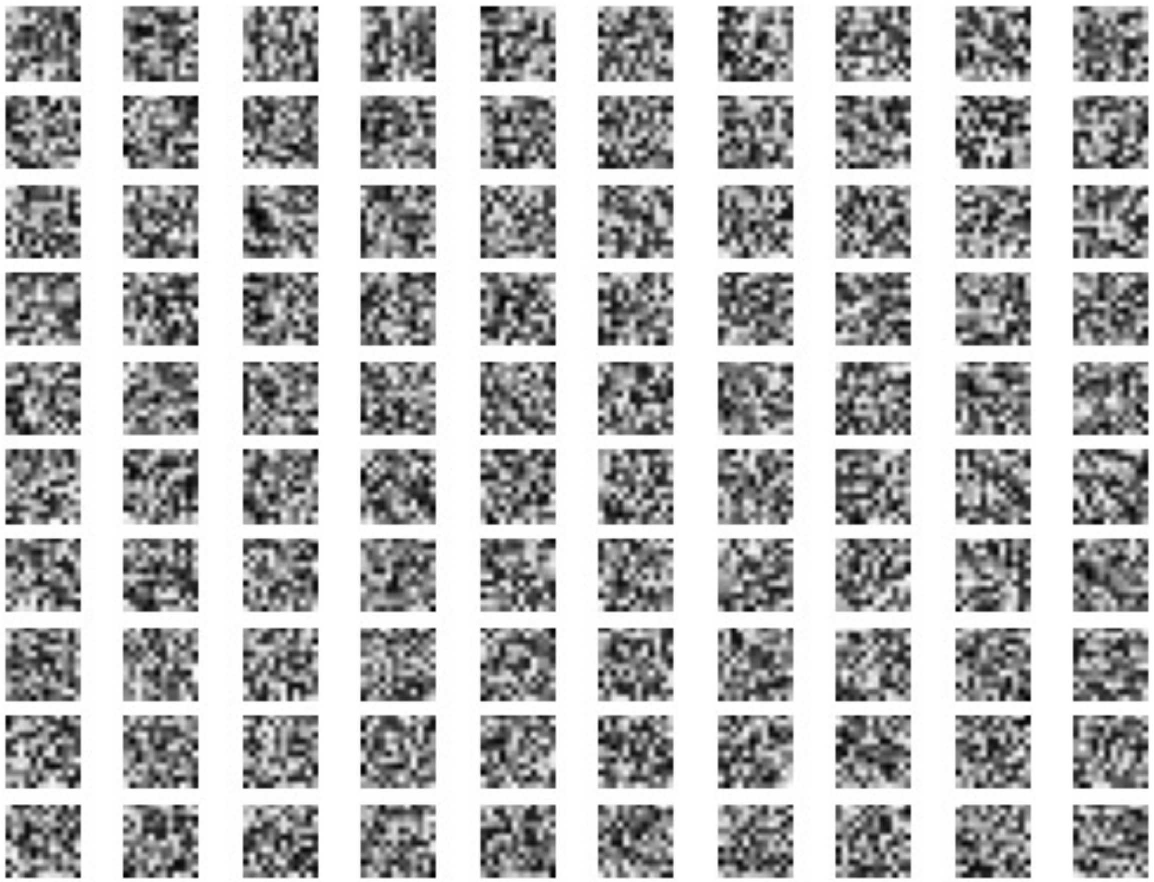
Question 2c



12x12 hidden activations (reshaped from 144 dimensions) of 100 samples

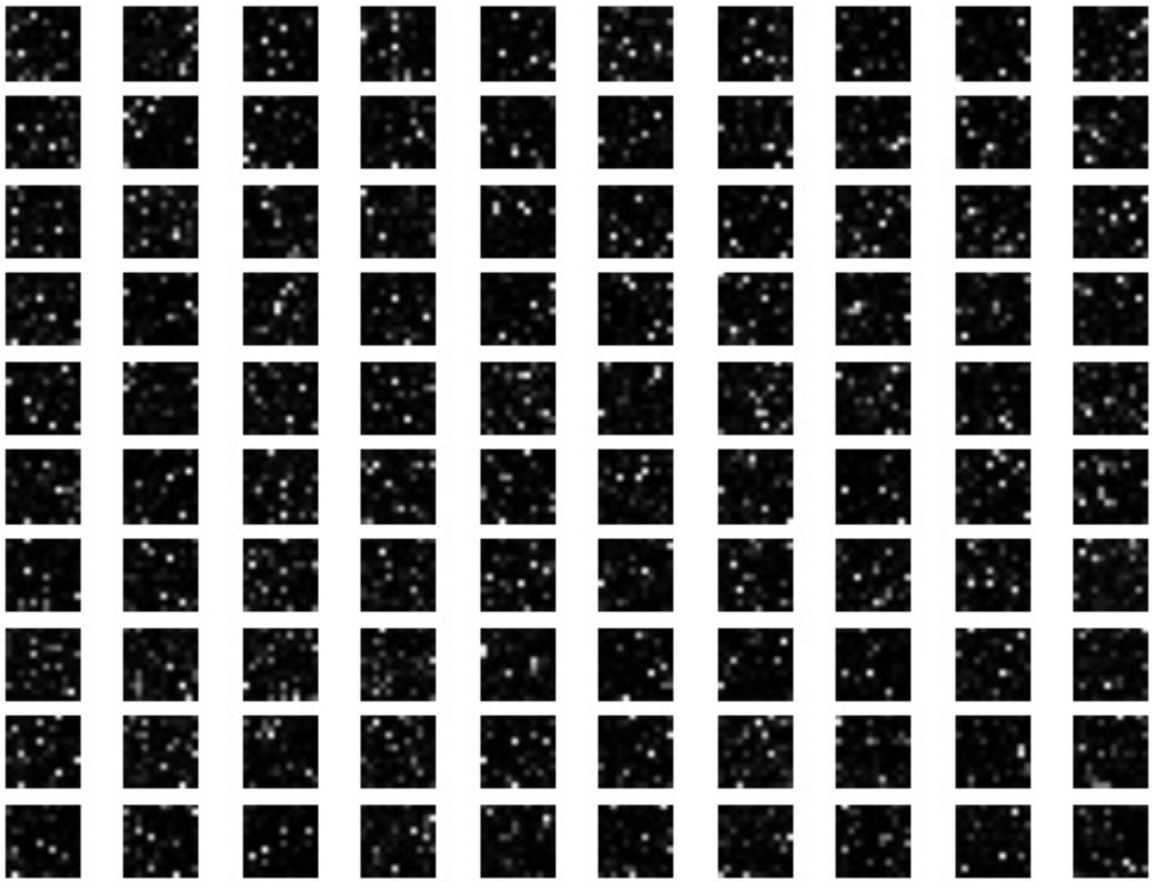
Question 2c

Without sparsity



12x12 hidden activations (reshaped from 144 dimensions) of 100 samples

With sparsity



12x12 hidden activations (reshaped from 144 dimensions) of 100 samples

Question 3

Design a denoising autoencoder to reconstruct MNIST images:
http://yann.lecun.com/exdb/mnist/

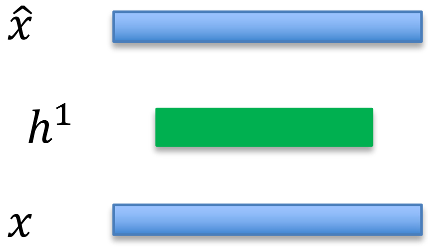
(a) Assume one hidden layer with 625 neurons, multiplicative noise, and cross-entropy cost function. Use 10% corruption level, learning factor $\alpha = 0.1$, batch size = 128, sparsity constant $\rho = 0.02$, and penalty parameter $\beta = 0.4$.
Plot the learning curves, the weights, and the hidden layer activations for sample test images.

(b) Add another hidden layer with 100 neurons and train the autoencoder as before. Plot the feature maps.
Plot the learning curves, the weights, and the hidden layer activations for sample test images

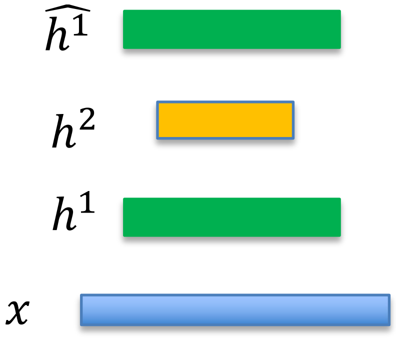
(c) Add a softmax layer on top of the second hidden layer to design a classifier. Show learning curves and find the accuracy of the classifier.
Plot the learning curves and the weights and find the accuracy for test patterns.

t10q3.ipynb

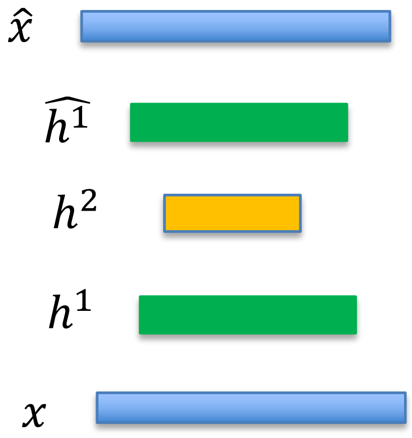
Deep stacked autoencoders



Training first hidden layer

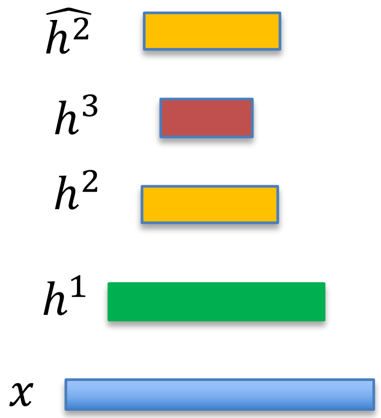


Training second hidden layer. Reconstructs first-hidden layer output

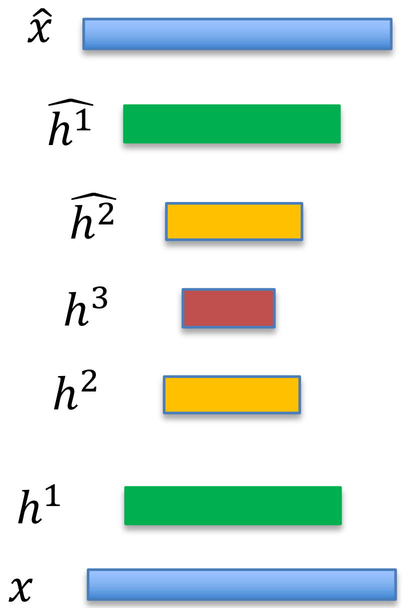


Autoencoder with two hidden layers

Deep stacked autoencoders



Training the third hidden layer.
Reconstructs second hidden layer output

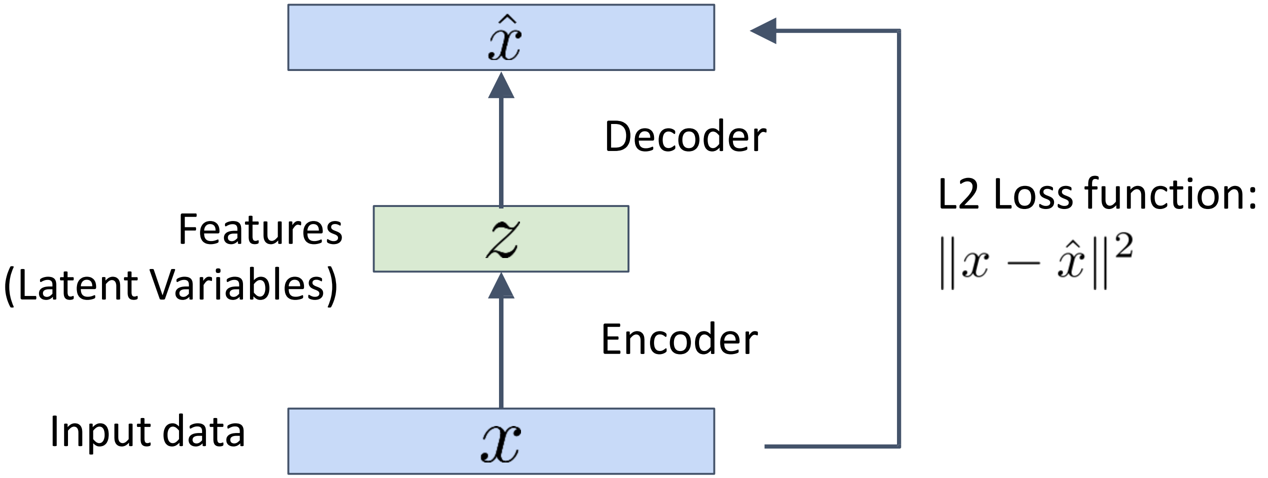


Autoencoder with three hidden layers

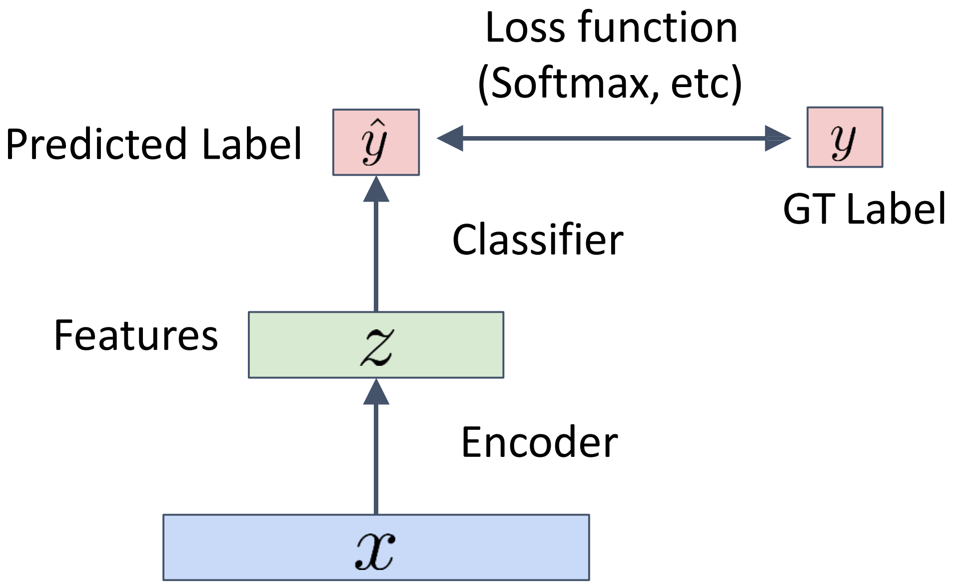
Semi-Supervised Classification

- Many images, but few ground truth labels

start unsupervised
train autoencoder on many images



supervised fine-tuning
train classification network on labeled images



Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n