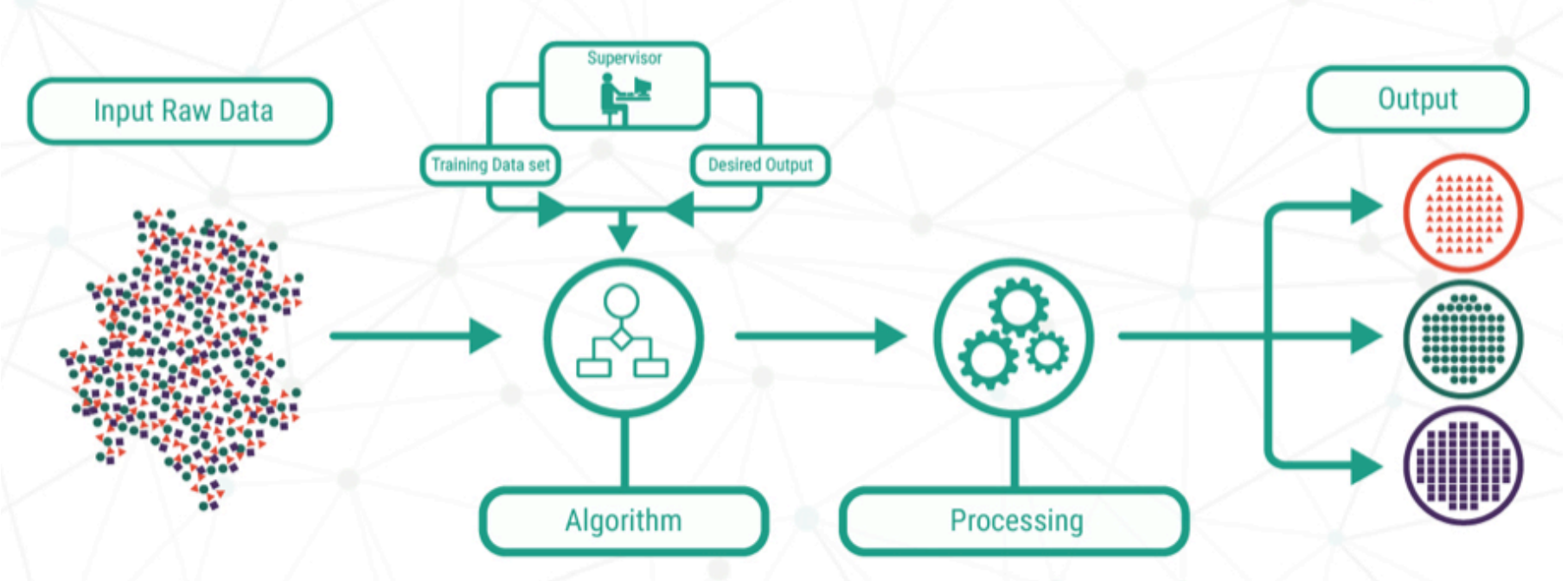
Unsupervised learning

Kun Suo
Computer Science, Kennesaw State University
https://kevinsuo.github.io/

Supervised Learning

- Supervised learning is to learn a function (model parameters) from a given annotated training data set. When new data arrives, the result can be predicted based on this function



Supervised Learning Example: classification

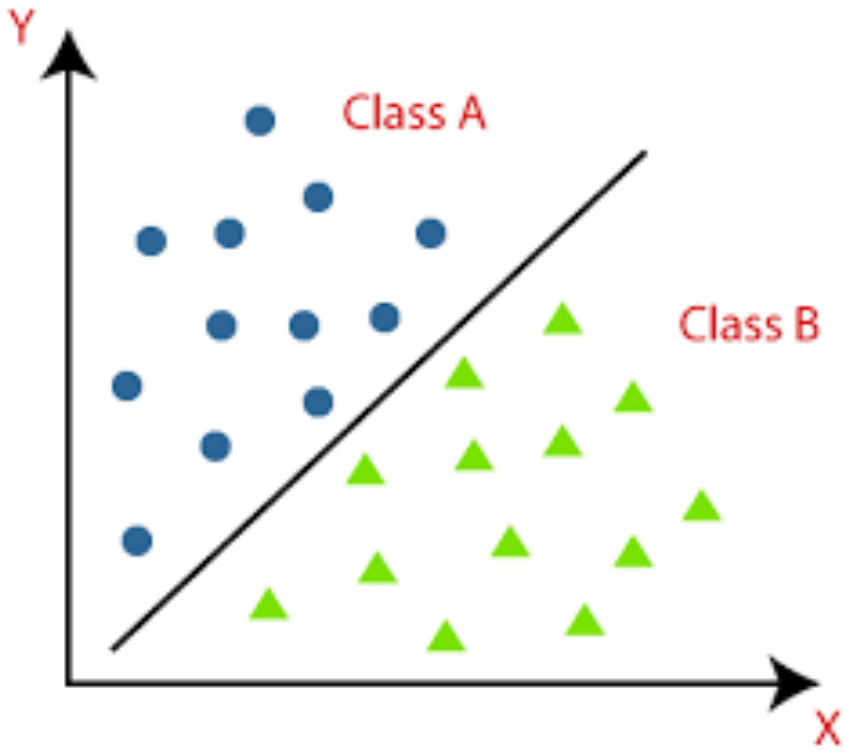
Training Data:

Y=1: (1, 99)、(0.9, 80)、(0.80, 100) …

Y=-1: (0.2, 30)、(0.5, 50)、(0.4, 30) …

Test:

X=(0.85, 98), Y=?



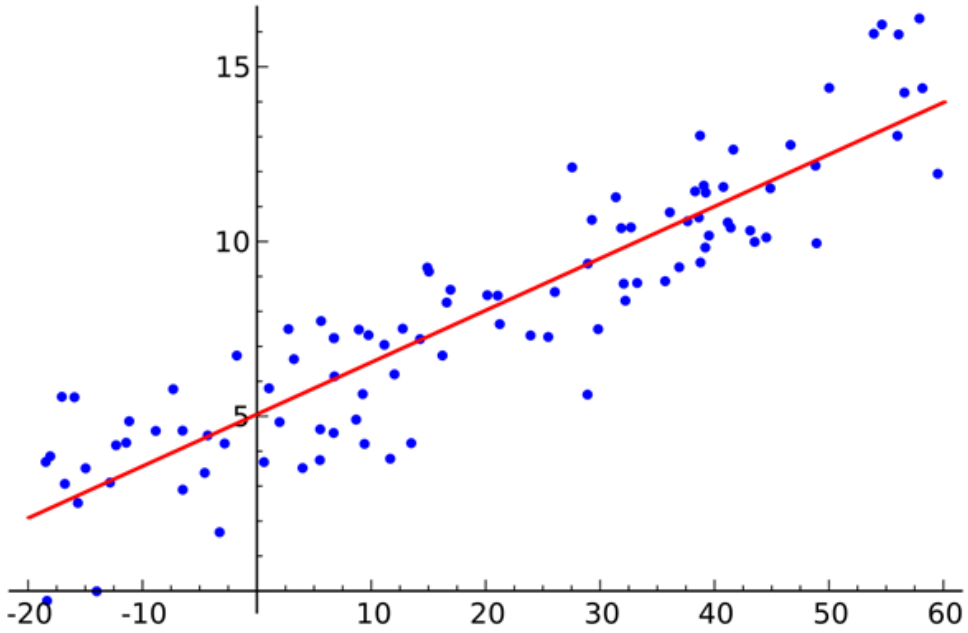
Supervised Learning Example: regression

Training Data:

35	150
40	170
45	190
65	200
74	224
80	245
120	320
140	400
230	640
300	780
400	900
500	1100
600	1300

Test: X=90 Y= ?

Simple Regression → y = mx + b



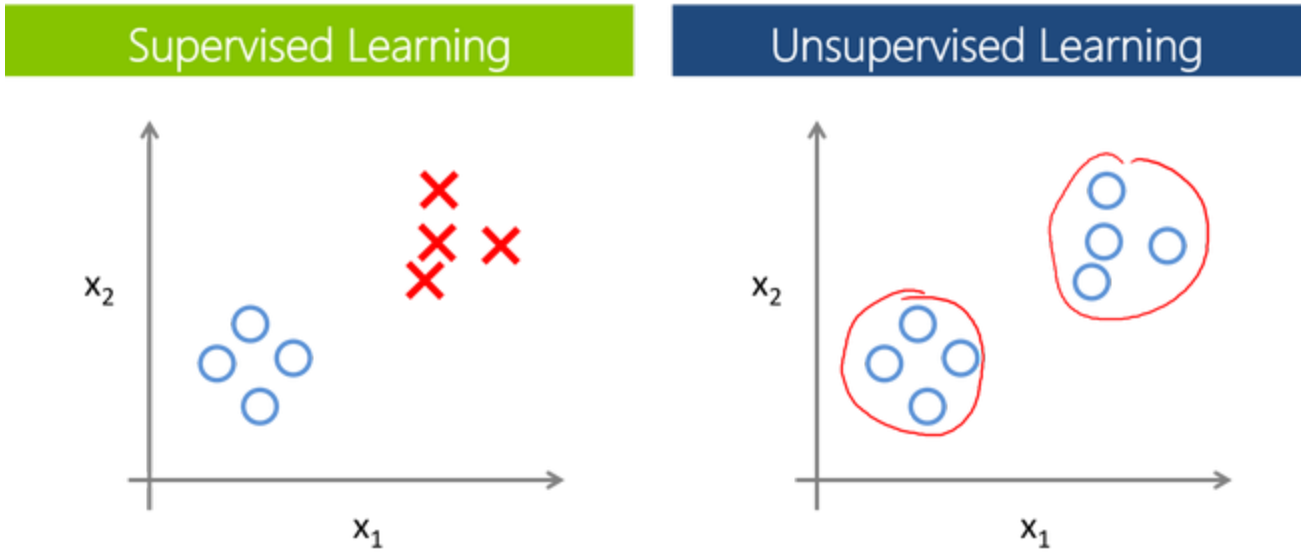
Unsupervised Learning

- Supervised ≈ Is any data labeled

- Unsupervised learning is not to tell the computer what to do, but to let the machine learn what to do

Unsupervised Learning vs. Supervised Learning

- 1. The training set and test set are used differently.
  - The purpose of supervised learning is to find a pattern in the training set, and then apply this pattern to the test set.
  - However, unsupervised learning has no training set, only one set of data, and find the rules in this set of data.



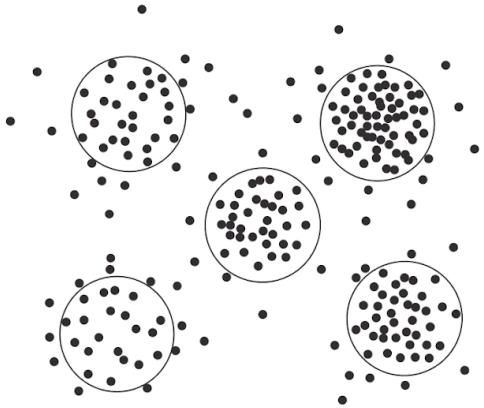
Unsupervised Learning vs. Supervised Learning

- 2. Whether the training set have labels.

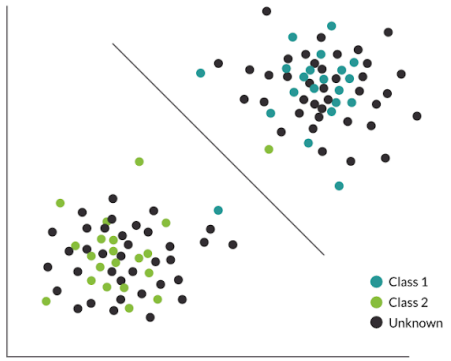
  - The result of supervised learning recognition is the data to be recognized is labeled, so the training sample set must be composed of labeled samples.

  - The unsupervised learning method only has the data set to be analyzed, and there is no label in advance.

Unsupervised

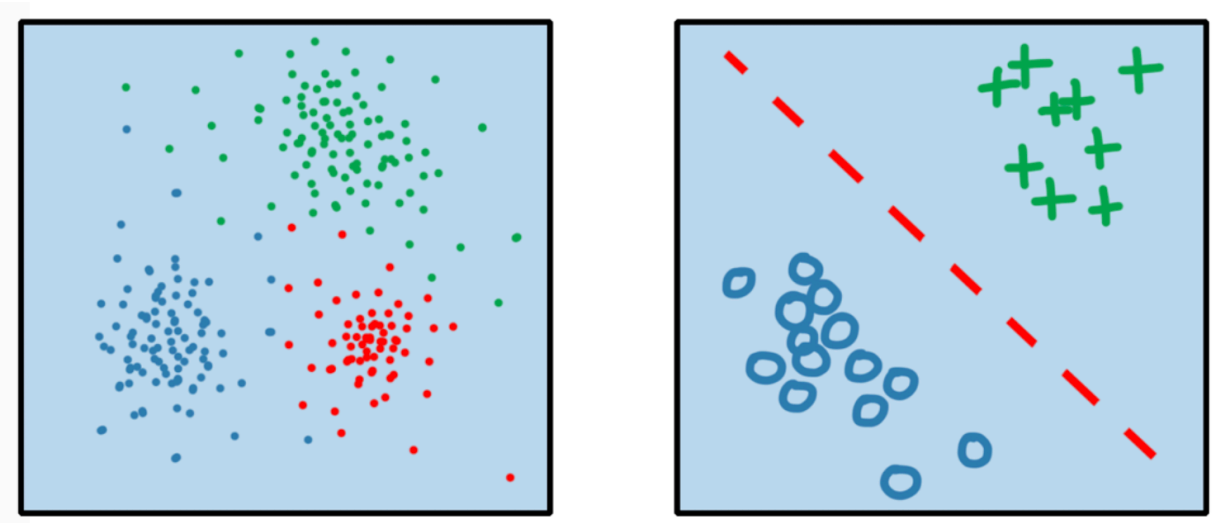


Supervised



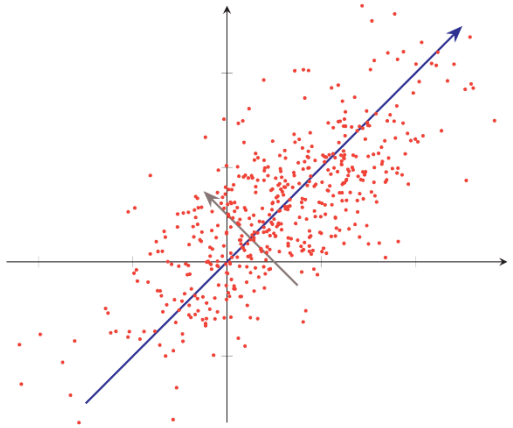
Unsupervised Learning vs. Supervised Learning

- 3. The unsupervised learning method is looking for the regularity in the data set, and the supervised learning is to train an optimal model through the existing labeled data set

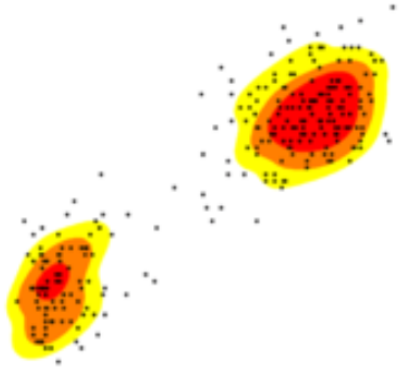


Sort by shape as dimension, or by color as dimension

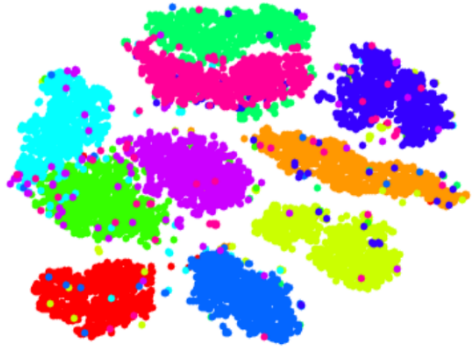
Typical unsupervised learning problem



Unsupervised feature learning



Density estimation



Clustering

Why we need unsupervised learning?

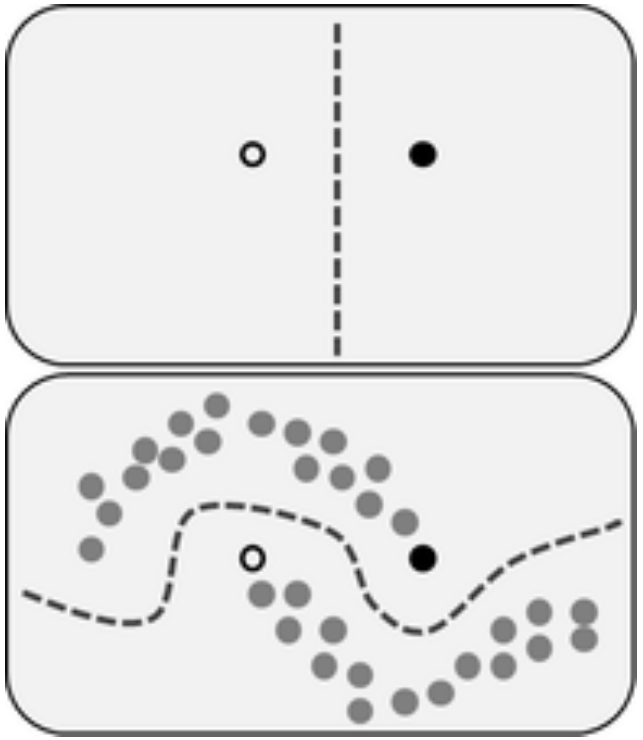
- Two aspects limit the potential of supervised learning:

  - Algorithms are constrained by the ***bias of supervised information***. Yes, the algorithm is how to accomplish this task. However, the algorithm cannot consider other possible situations when solving the problem.

  - Since learning is performed under supervision, ***it takes a huge amount of manpower to create labels for the algorithm***. The fewer labels you create manually, the less data the algorithm can use for training. Most real data is unlabeled.

Semi-supervised learning

- Supervised learning: train models from labeled training data
- Unsupervised learning task: training the model with unlabeled training data
- Semi-supervised learning task: use a large amount of unlabeled training data and a small amount of labeled data to train the model

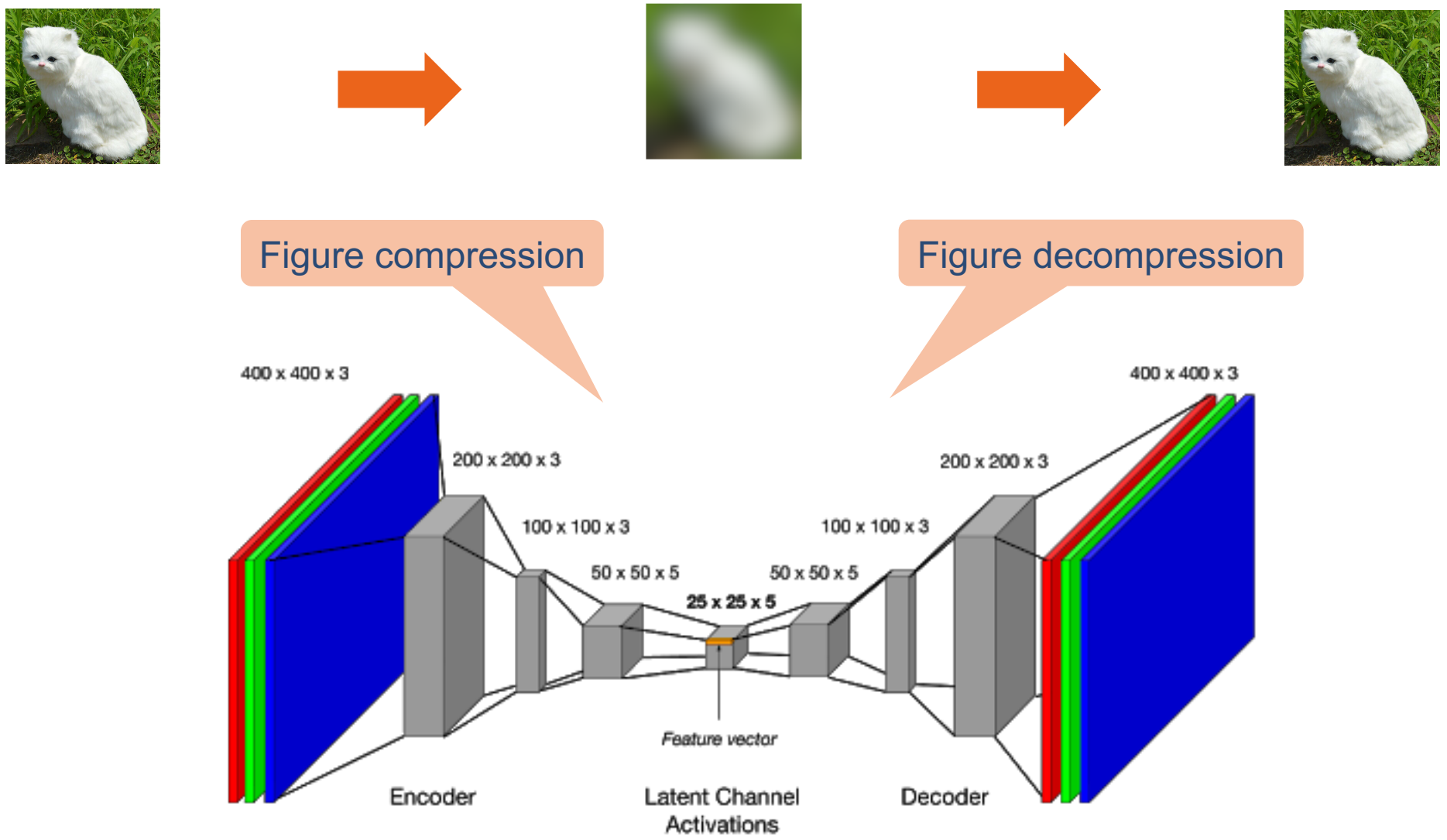


Auto Encoder

Autoencoder

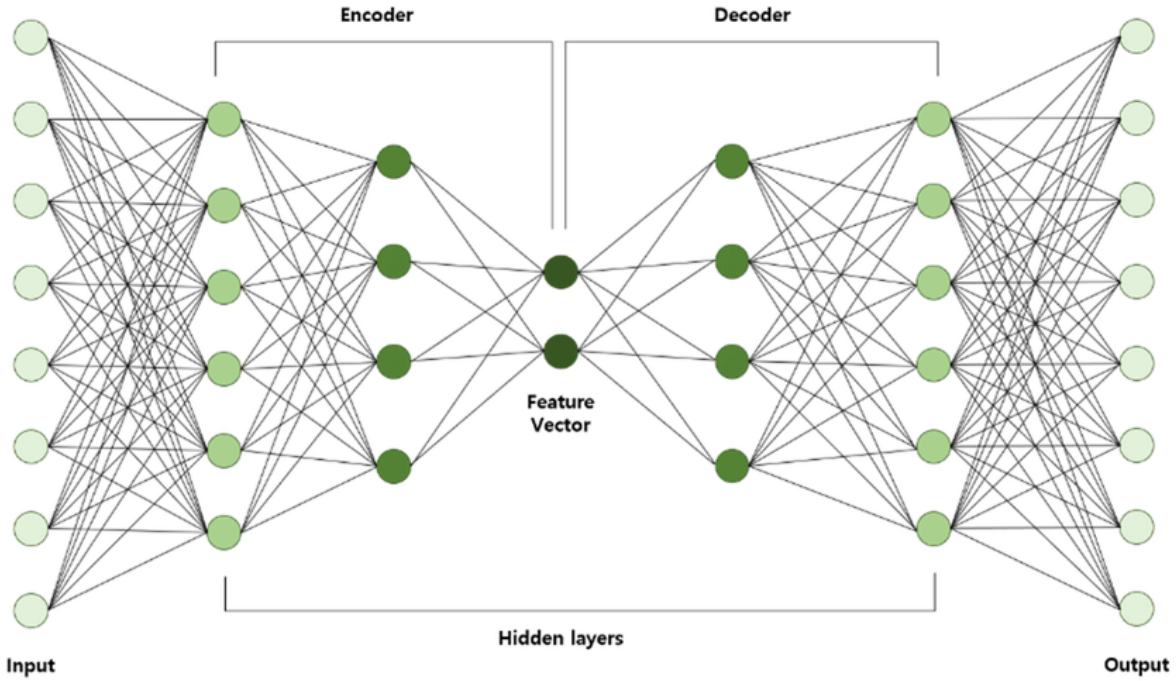
- **Autoencoders** are an unsupervised learning technique in which we leverage neural networks to learn efficient data codings in an unsupervised manner

Autoencoder



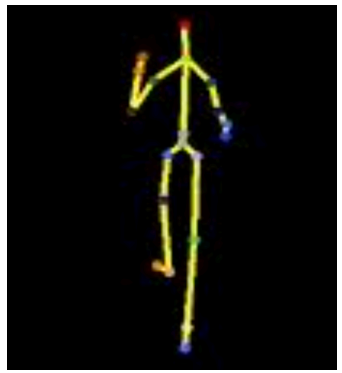
Why we need encoder? E.g., Figure compression

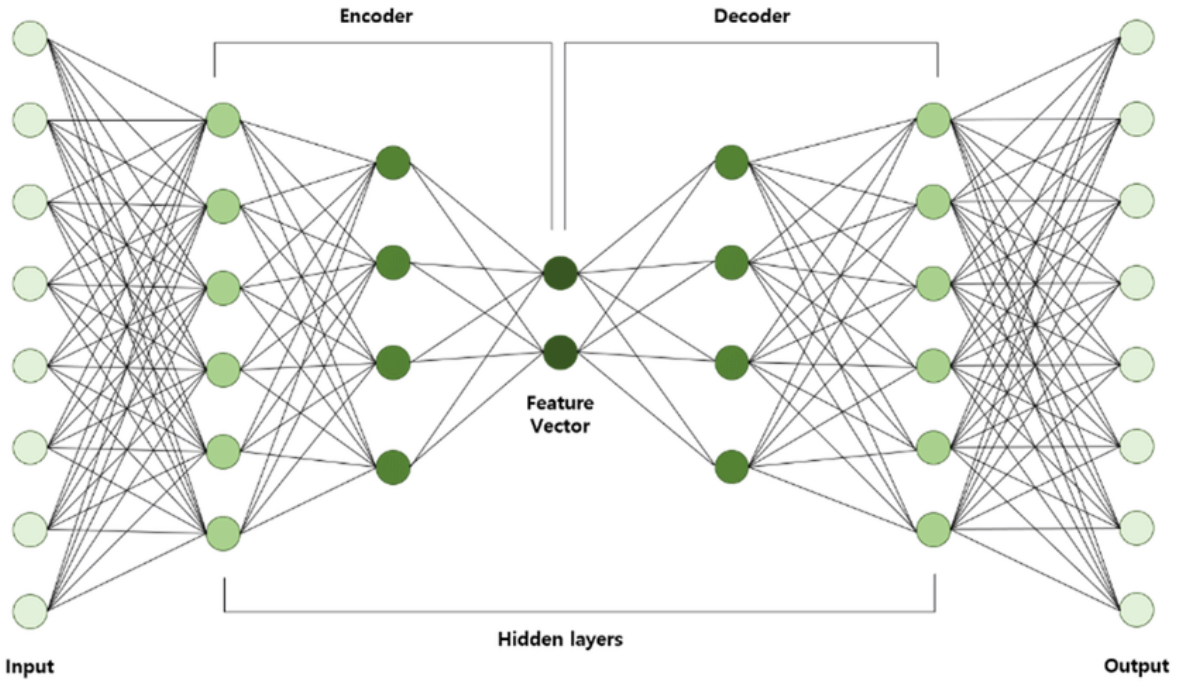
- Sometimes the neural network has to accept a large amount of input information. For example, when the input information is a high-definition picture, the amount of input information may reach tens of millions. It is a very heavy task for the neural network to learn directly from tens of millions of information sources.



Why we need encoder? E.g., Figure compression

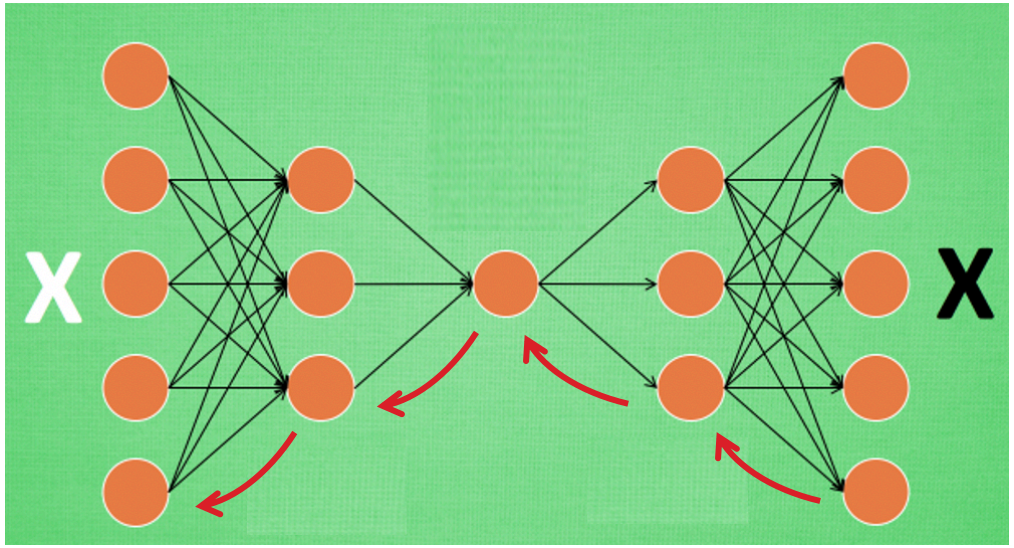
- Encoder is to compress the input and extract the most representative information in the original picture, reduce the amount of input information, and then put the reduced information into the neural network for learning





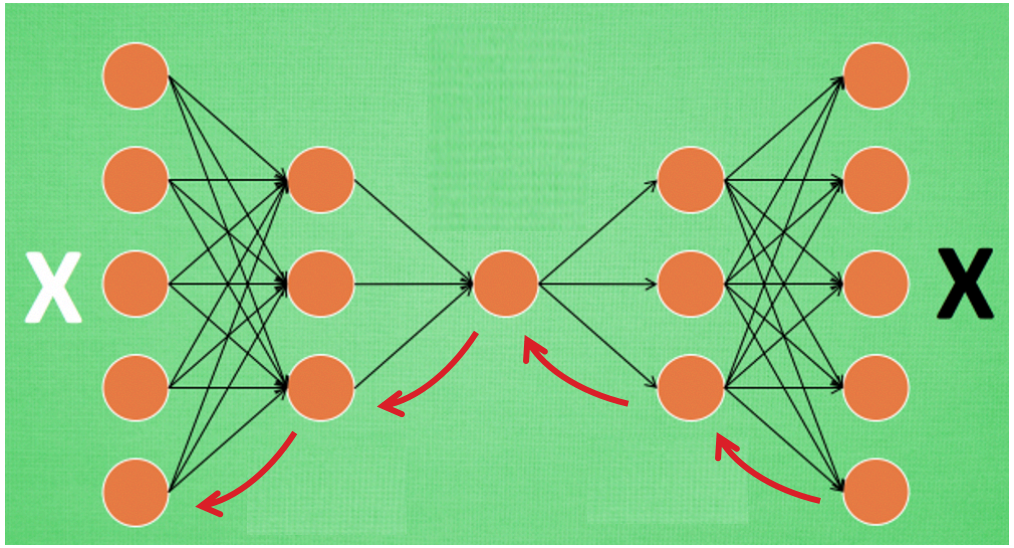
How to build an auto-encoder?

- By compressing the white X of the original data, decompressing it into black X, then by comparing the black and white X, the prediction error is calculated, and the reverse transmission is performed to gradually improve the accuracy of the auto-encoder



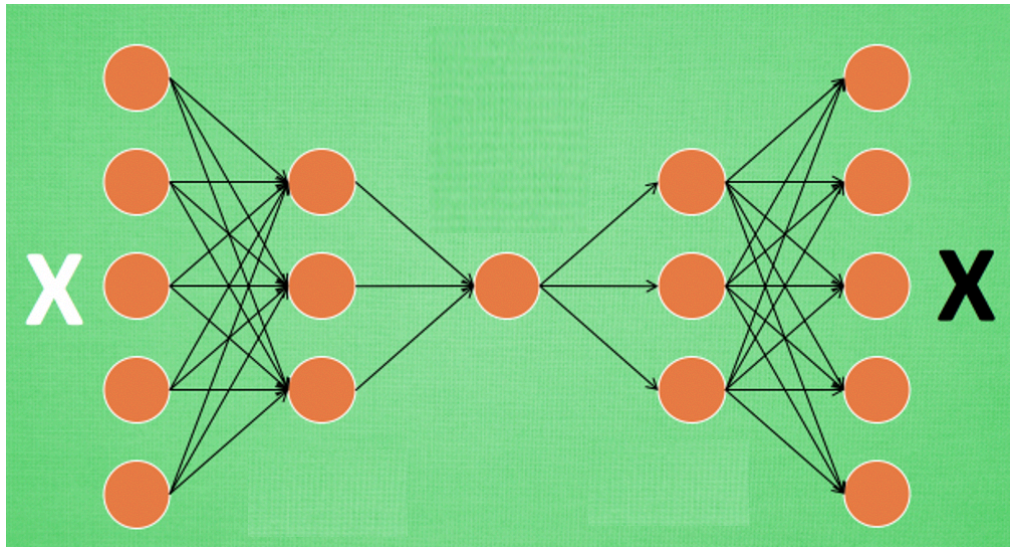
How to build an auto-encoder?

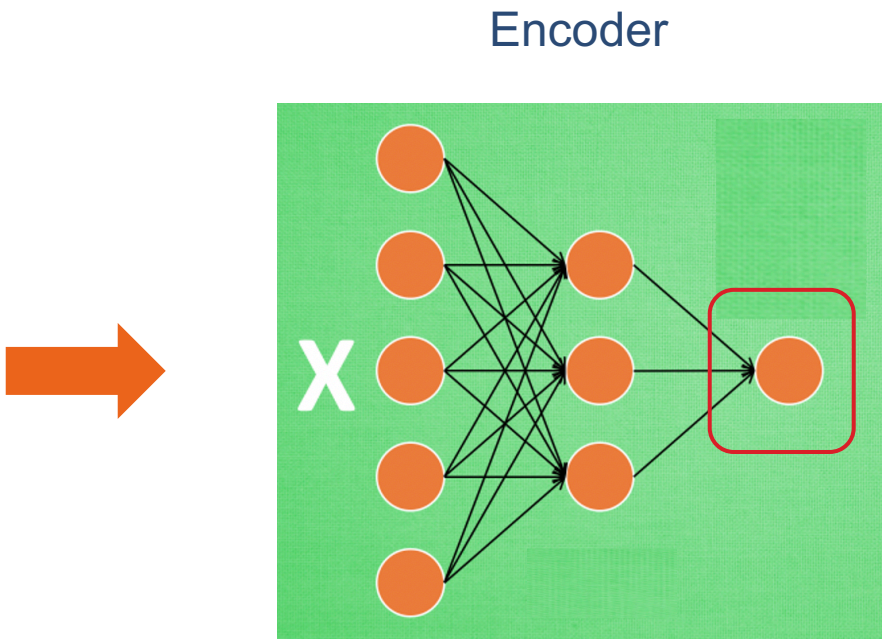
- We only used the input data X, and did not use the data label corresponding to X, so it can also be said that auto-encoder is a kind of unsupervised learning



Encoder

- The encoder can get the essence of the original data, and then we only need to create a small neural network to learn the essence of the data, which not only reduces the burden of the neural network, but also achieves good results.



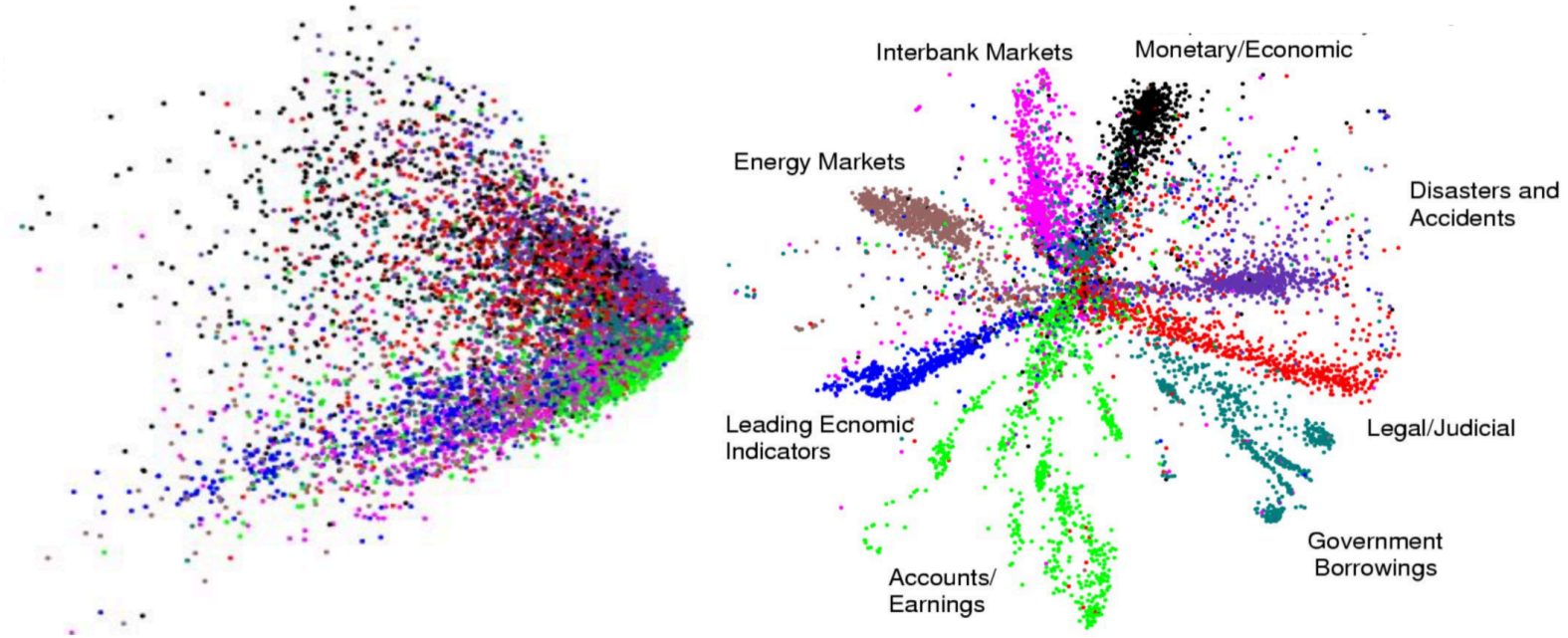


essence of the original data

Auto-encoder

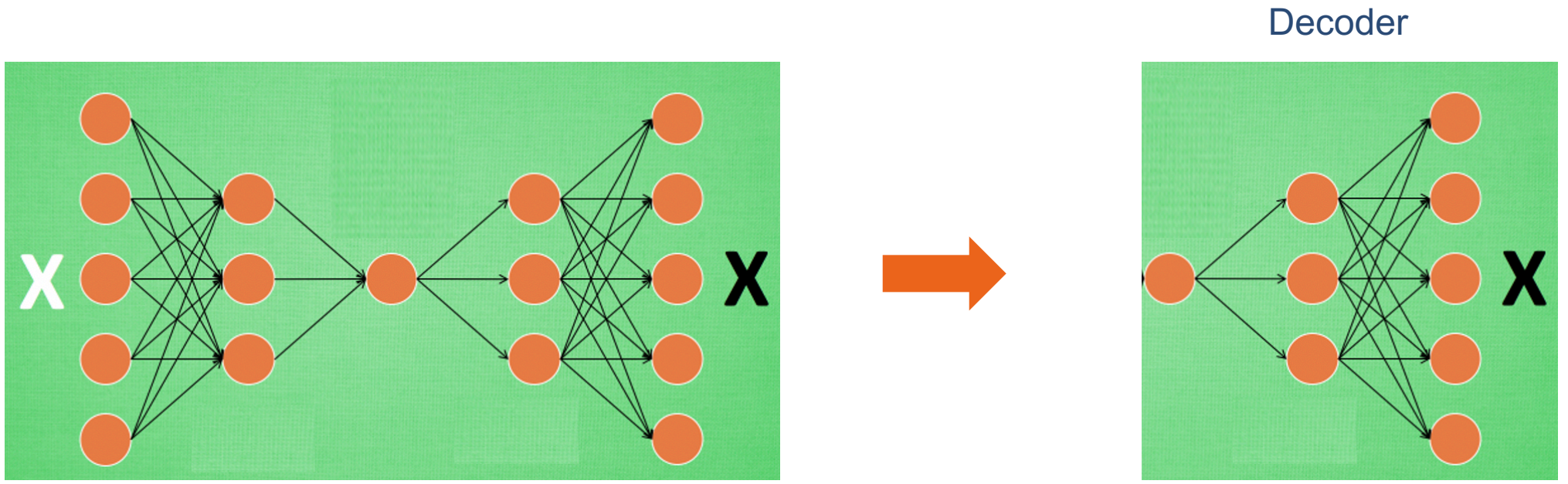
- This is a data sorted out through auto-encoder, which can summarize the characteristics of each type of data from the original data

- Auto-encoder can reduce the dimension of feature attributes



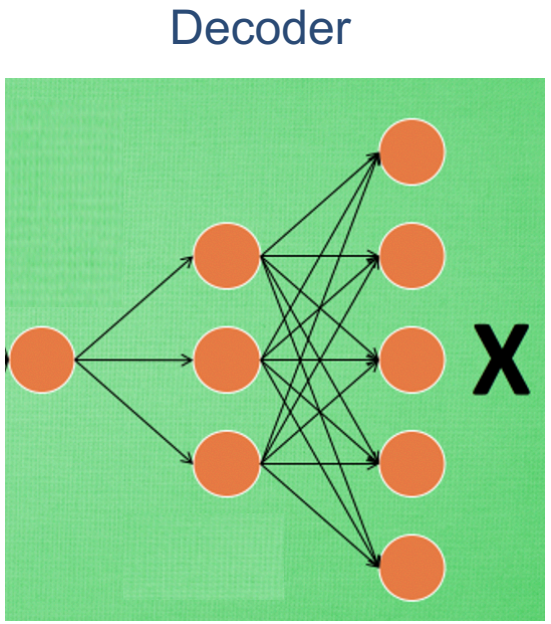
Decoder

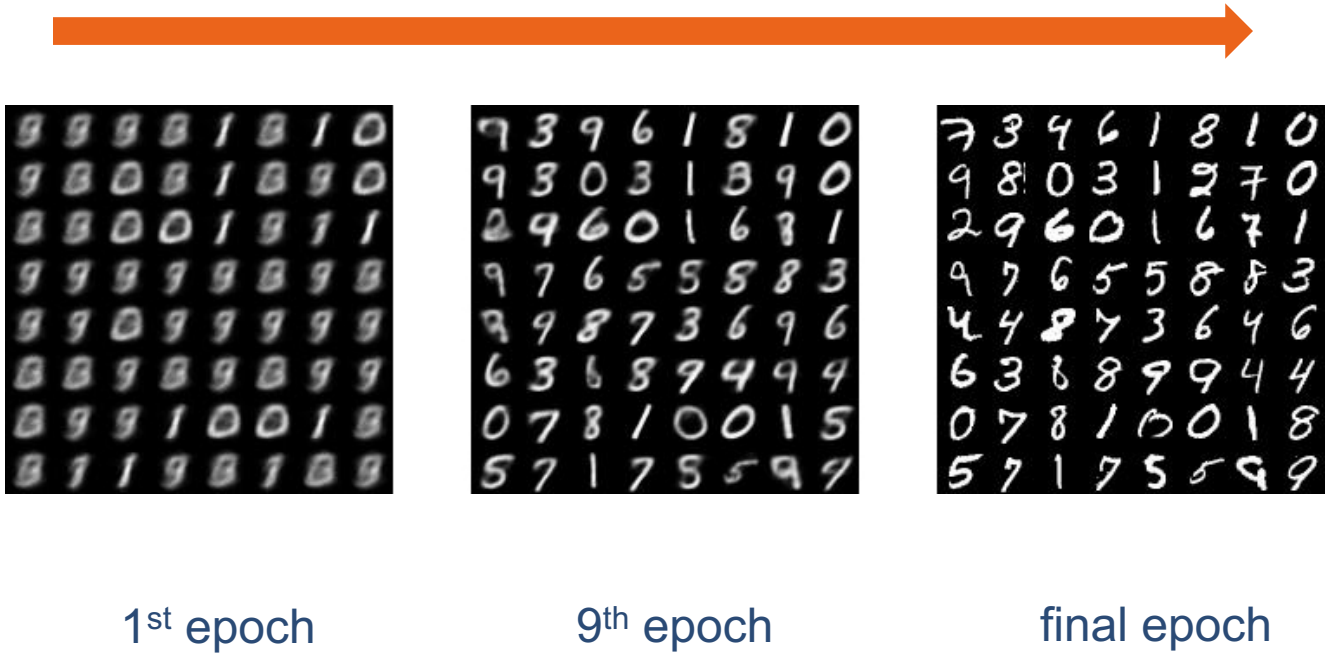
- When the decoder is trained, it is to decompress the essence information into the original information, so this provides a decompressor



Decoder

- Example: imitate and generate handwritten numbers





1st epoch

9th epoch

final epoch

Example: Auto-encoder

https://github.com/kevinsuo/CS7357/blob/master/usl/usl_1.py

- Insteall tensorflow
  - sudo apt install python3-pip –y
  - pip3 install tensorflow==1.14

- Check the version of tensorflow
  - python3 -c 'import tensorflow as tf; print(tf.__version__)'

Example: Auto-encoder

- Install older version of numpy
  - pip3 uninstall numpy
  - pip3 install numpy==1.16.4
  - pip3 install matplotlib
- Download MNIST dataset
  - wget https://raw.githubusercontent.com/tensorflow/tensorflow/r0.7/tensorflow/examples/tutorials/mnist/input_data.py
  - python3 input_data.py

MNIST dataset

Example: Auto-encoder

```
1   from __future__ import division, print_function, absolute_import
2
3   import tensorflow as tf
4   import numpy as np
5   import matplotlib.pyplot as plt
6
7   # Import MNIST data
8   from tensorflow.examples.tutorials.mnist import input_data
9   mnist = input_data.read_data_sets("/tmp/data/", one_hot=False)
10
11
12  # Visualize decoder setting
13  # Parameters
14  learning_rate = 0.01
15  training_epochs = 5
16  batch_size = 256
17  display_step = 1
18  examples_to_show = 10
19
20  # Network Parameters
21  n_input = 784  # MNIST data input (img shape: 28*28)
```

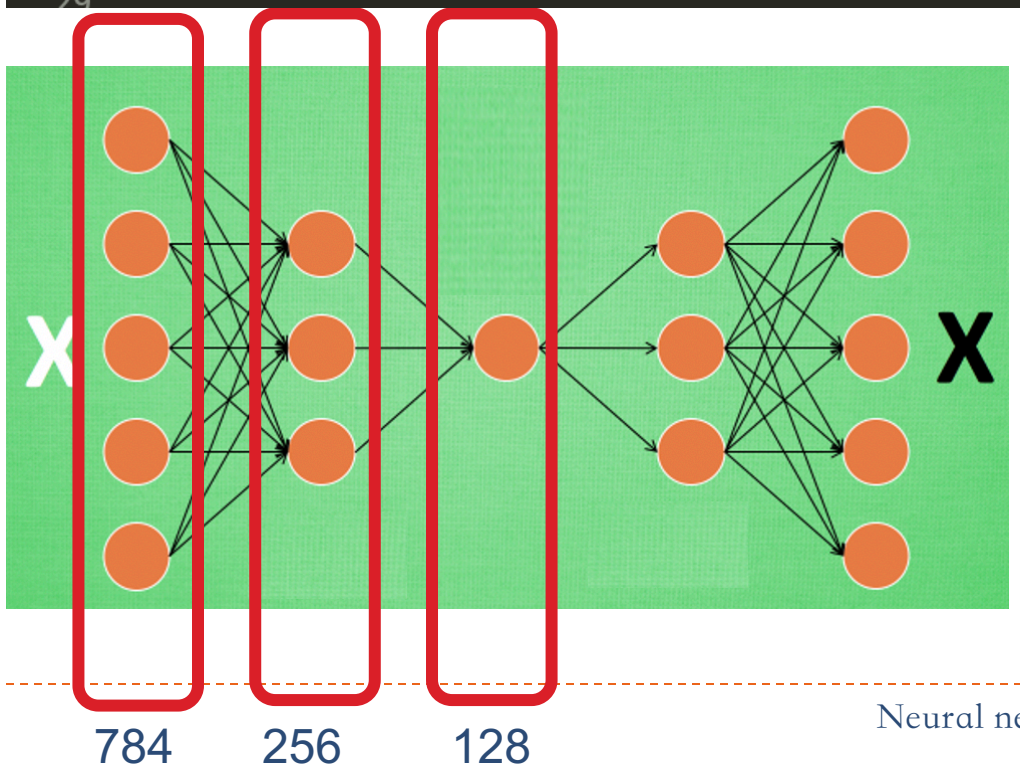
Input dataset

Train parameters

Input data size;
Input data number;

Example: Auto-encoder

```
19
20    # Network Parameters
21    n_input = 784  # MNIST data input (img shape: 28*28)
22
23    # tf Graph input (only pictures)
24    X = tf.placeholder("float", [None, n_input])
25
26    # hidden layer settings
27    n_hidden_1 = 256 # 1st layer num features
28    n_hidden_2 = 128 # 2nd layer num features
29
```



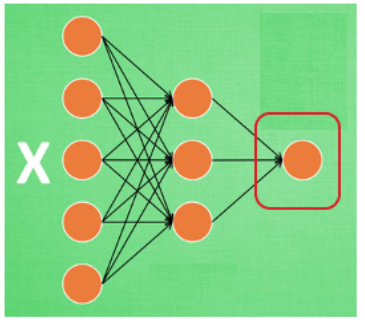
Example: Auto-encoder

The first layer is compressed into 256 layers

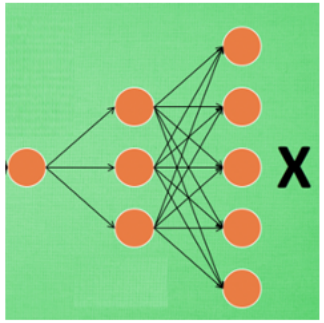
The second layer is compressed into 128 layers

```
44  # Building the encoder
45  def encoder(x):
46      # Encoder Hidden layer with sigmoid activation #1
47      layer_1 = tf.nn.sigmoid(tf.add(tf.matmul(x, weights['encoder_h1']),
48                                     biases['encoder_b1']))
49      # Decoder Hidden layer with sigmoid activation #2
50      layer_2 = tf.nn.sigmoid(tf.add(tf.matmul(layer_1, weights['encoder_h2']),
51                                     biases['encoder_b2']))
52      return layer_2
53
54
55  # Building the decoder
56  def decoder(x):
57      # Encoder Hidden layer with sigmoid activation #1
58      layer_1 = tf.nn.sigmoid(tf.add(tf.matmul(x, weights['decoder_h1']),
59                                     biases['decoder_b1']))
60      # Decoder Hidden layer with sigmoid activation #2
61      layer_2 = tf.nn.sigmoid(tf.add(tf.matmul(layer_1, weights['decoder_h2']),
62                                     biases['decoder_b2']))
63      return layer_2
64
65
66
67  # Construct model
68  encoder_op = encoder(X)
69  decoder_op = decoder(encoder_op)
70
71  # Prediction
72  y_pred = decoder_op
73  # Targets (Labels) are the input data.
74  y_true = X
75
```

Encoder



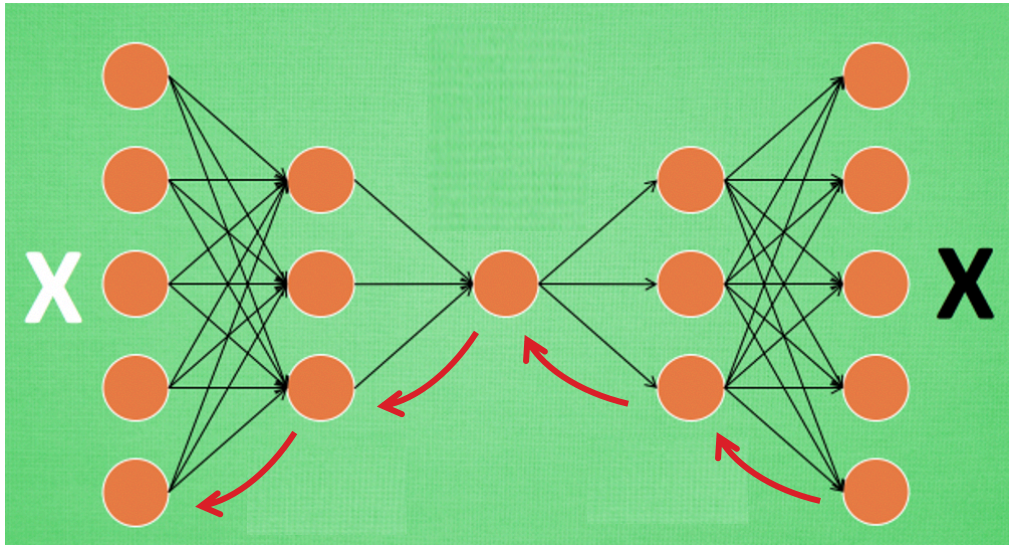
Decoder



Example: Auto-encoder

```
76    # Define loss and optimizer, minimize the squared error
77    cost = tf.reduce_mean(tf.pow(y_true - y_pred, 2))
78    optimizer = tf.train.AdamOptimizer(learning_rate).minimize(cost)
79
```

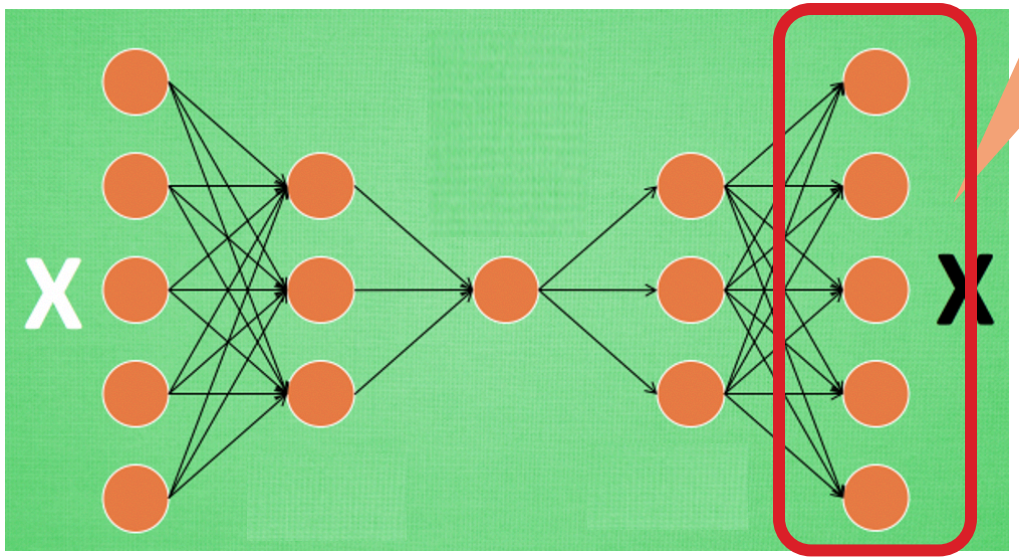
Keep learning to minimize cost



Example: Auto-encoder

```
105     # # Applying encode and decode over test set
106     encode_decode = sess.run(
107         y_pred, feed_dict={X: mnist.test.images[:examples_to_show]})
108     # Compare original images with their reconstructions
109     f, a = plt.subplots(2, 10, figsize=(10, 2))
110     for i in range(examples_to_show):
111         a[0][i].imshow(np.reshape(mnist.test.images[i], (28, 28)))
112         a[1][i].imshow(np.reshape(encode_decode[i], (28, 28)))
113     plt.show()
```

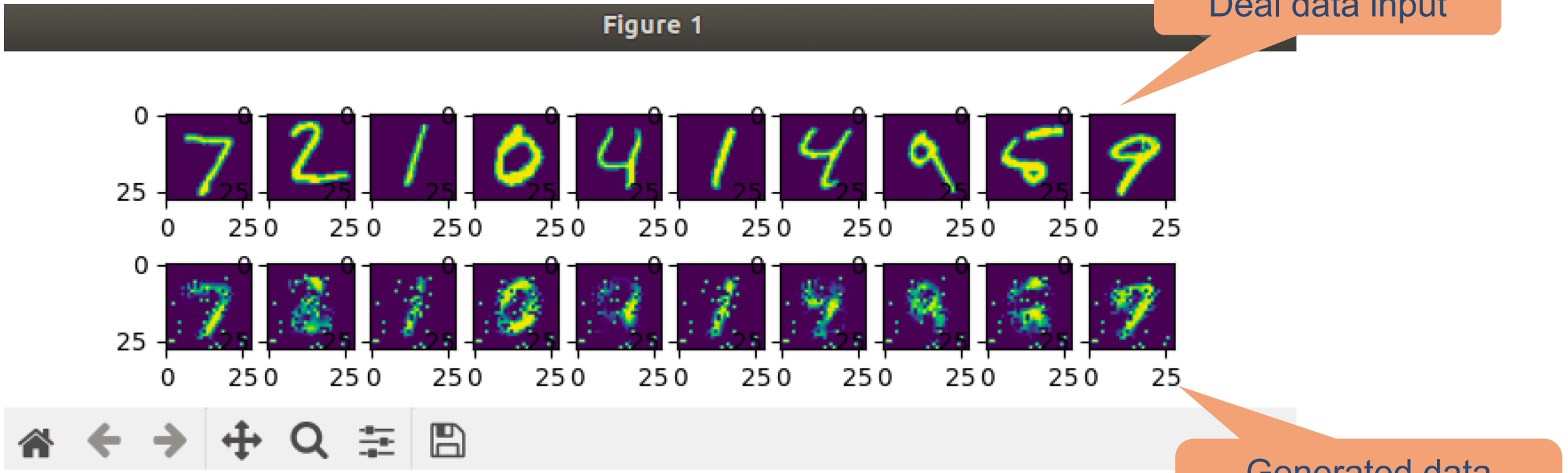
The output is all figures after encode-and-decode



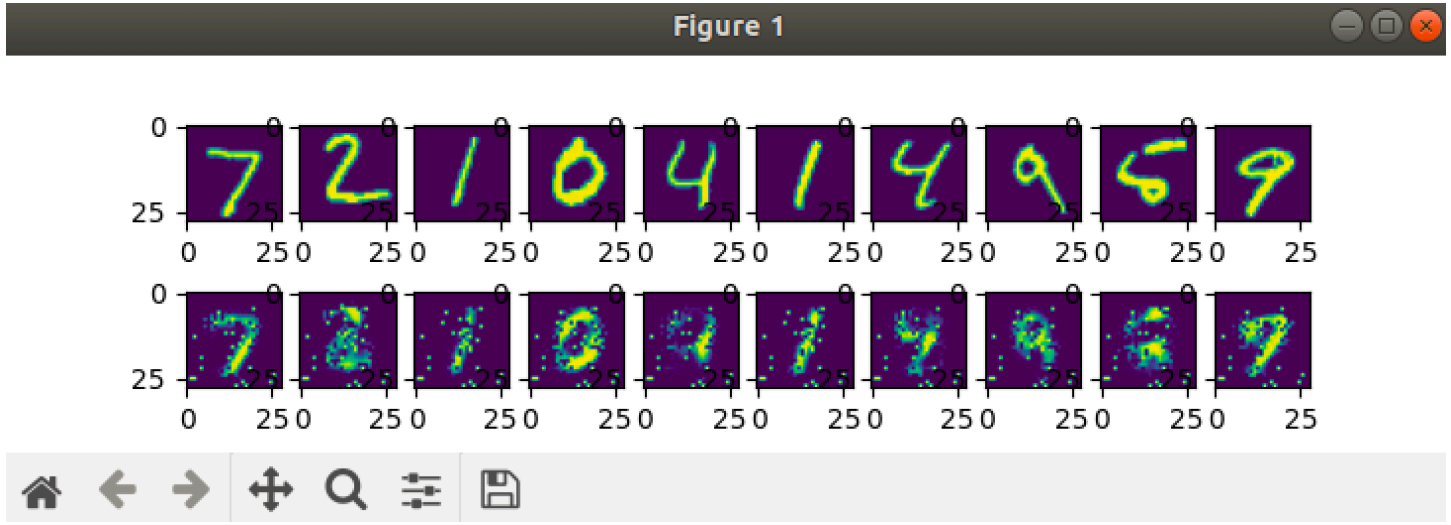
Example: Auto-encoder

```
Epoch: 0001 cost= 0.078412846
Epoch: 0002 cost= 0.069994360
Epoch: 0003 cost= 0.061541460
Epoch: 0004 cost= 0.059287664
Epoch: 0005 cost= 0.052562051
Optimization Finished!
```

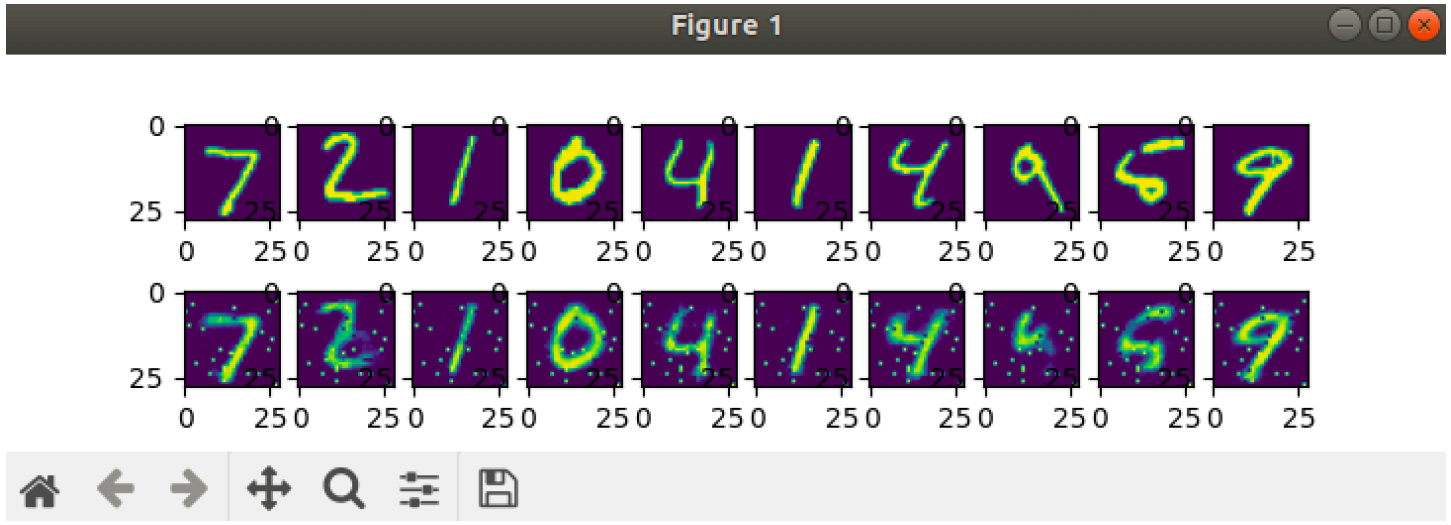
learning_rate = 0.01
training_epochs = 5



Example: Auto-encoder



learning_rate = 0.01
training_epochs = 5



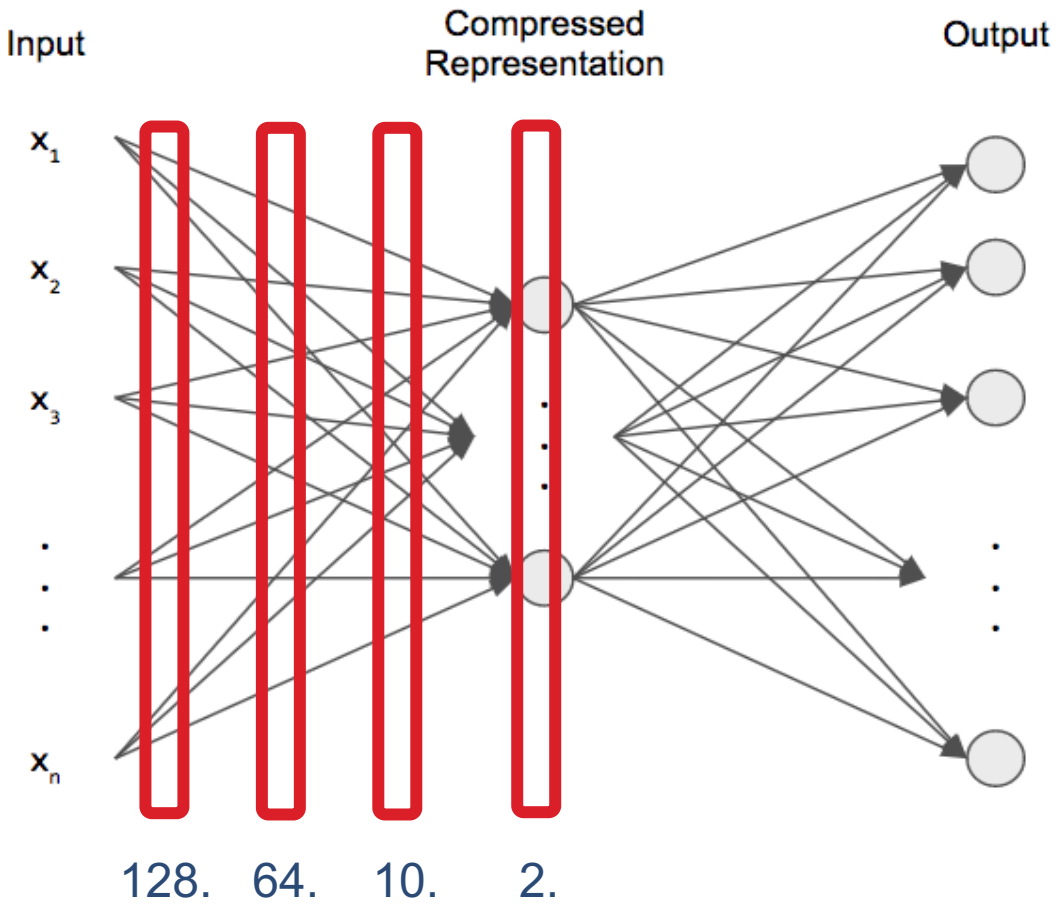
learning_rate = 0.001
training_epochs = 50

Example: Auto-encoder

https://github.com/kevinsuo/CS7357/blob/master/usl/usl_2.py

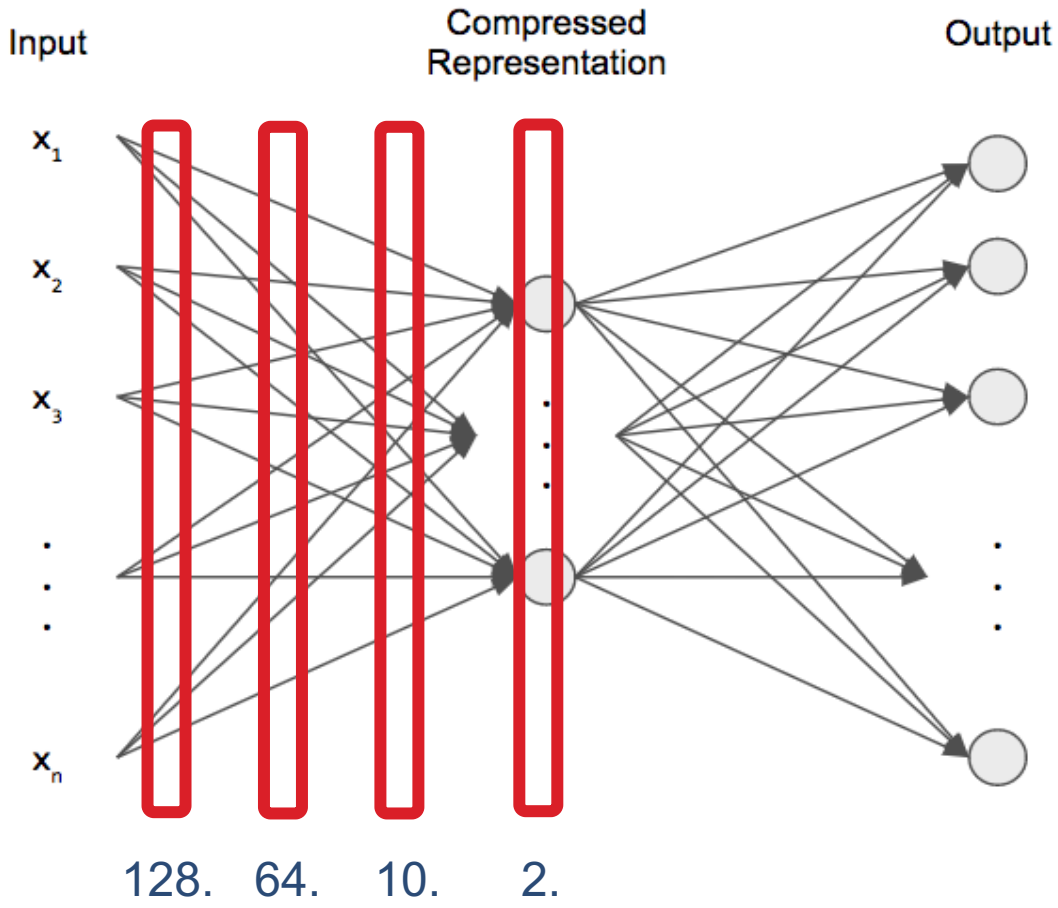
```
15    # Visualize encoder setting
16    # Parameters
17    learning_rate = 0.001    # 0.01 this learning rate will be better! Tested
18    training_epochs = 20
19    batch_size = 256
20    display_step = 1
21
```

```
28    # hidden layer settings
29    n_hidden_1 = 128
30    n_hidden_2 = 64
31    n_hidden_3 = 10
32    n_hidden_4 = 2
```



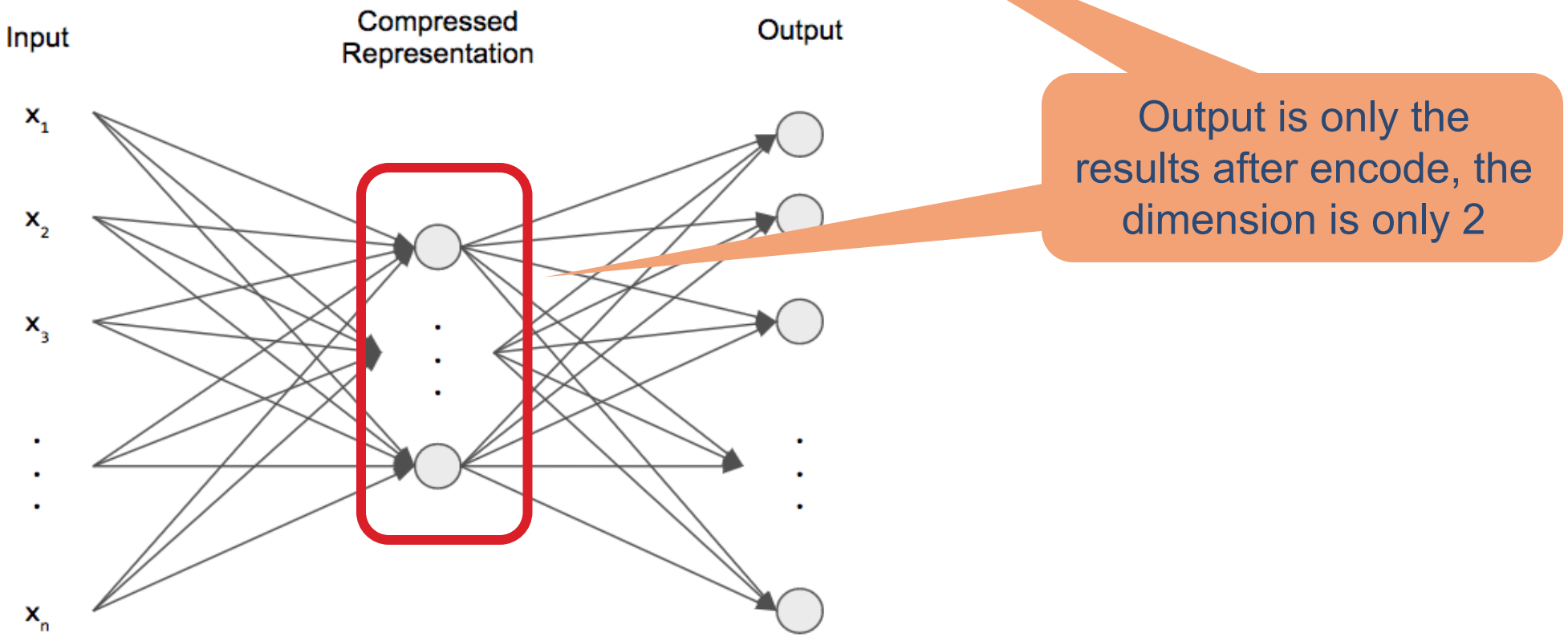
Example: Auto-encoder

```
58  def encoder(x):
59      layer_1 = tf.nn.sigmoid(tf.add(tf.matmul(x, weights['encoder_h1']),
60                                     biases['encoder_b1']))
61      layer_2 = tf.nn.sigmoid(tf.add(tf.matmul(layer_1, weights['encoder_h2']),
62                                     biases['encoder_b2']))
63      layer_3 = tf.nn.sigmoid(tf.add(tf.matmul(layer_2, weights['encoder_h3']),
64                                     biases['encoder_b3']))
65      layer_4 = tf.add(tf.matmul(layer_3, weights['encoder_h4']),
66                                      biases['encoder_b4'])
67      return layer_4
68
69
70  def decoder(x):
71      layer_1 = tf.nn.sigmoid(tf.add(tf.matmul(x, weights['decoder_h1']),
72                                     biases['decoder_b1']))
73      layer_2 = tf.nn.sigmoid(tf.add(tf.matmul(layer_1, weights['decoder_h2']),
74                                     biases['decoder_b2']))
75      layer_3 = tf.nn.sigmoid(tf.add(tf.matmul(layer_2, weights['decoder_h3']),
76                                    biases['decoder_b3']))
77      layer_4 = tf.nn.sigmoid(tf.add(tf.matmul(layer_3, weights['decoder_h4']),
78                                    biases['decoder_b4']))
79      return layer_4
80
```



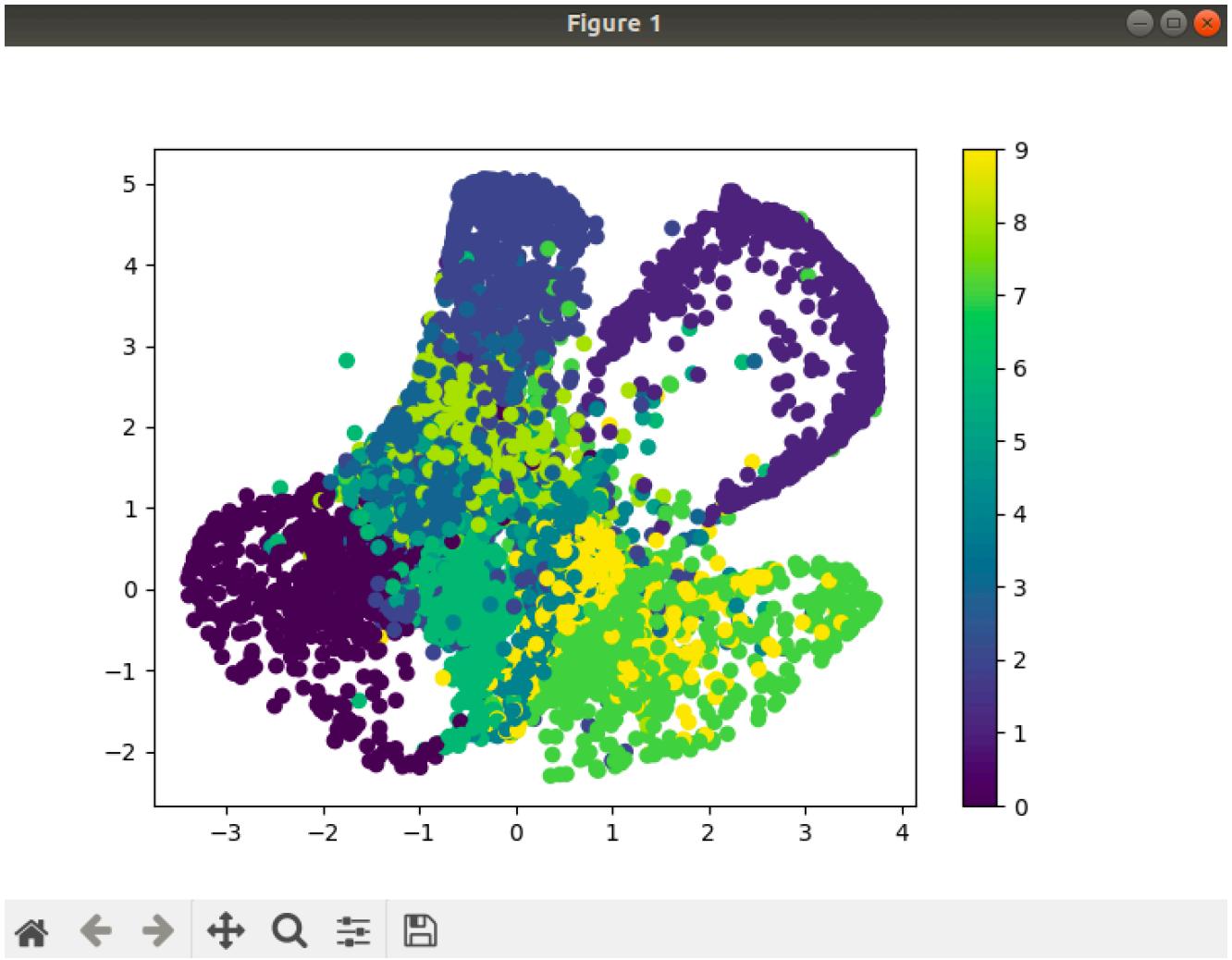
Example: Auto-encoder

```
121     encoder_result = sess.run(encoder_op, feed_dict={X: mnist.test.images})
122     plt.scatter(encoder_result[:, 0], encoder_result[:, 1], c=mnist.test.labels)
123     plt.colorbar()
124     plt.show()
```



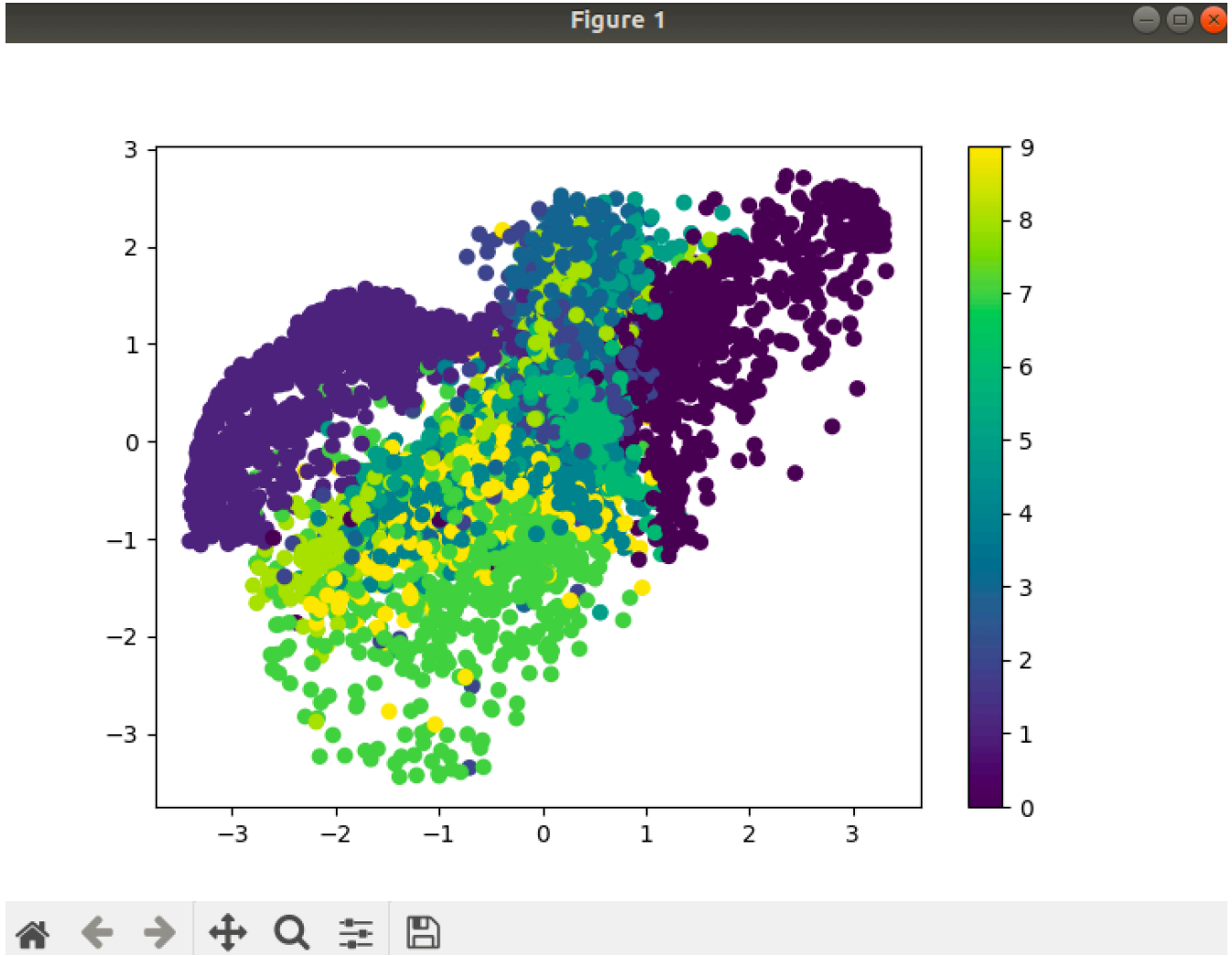
Output is only the results after encode, the dimension is only 2

Example: Auto-encoder



learning_rate = 0.01
training_epochs = 20

Example: Auto-encoder



learning_rate = 0.001
training_epochs = 50

Probability density estimation

Probability distribution function

- A **probability distribution function** refers to function that may be used to define a particular probability distribution

Probability distribution function F(x)

Outcome of die roll	**1**	**2**	**3**	**4**	**5**	**6**
Probability	1/6	1/6	1/6	1/6	1/6	1/6

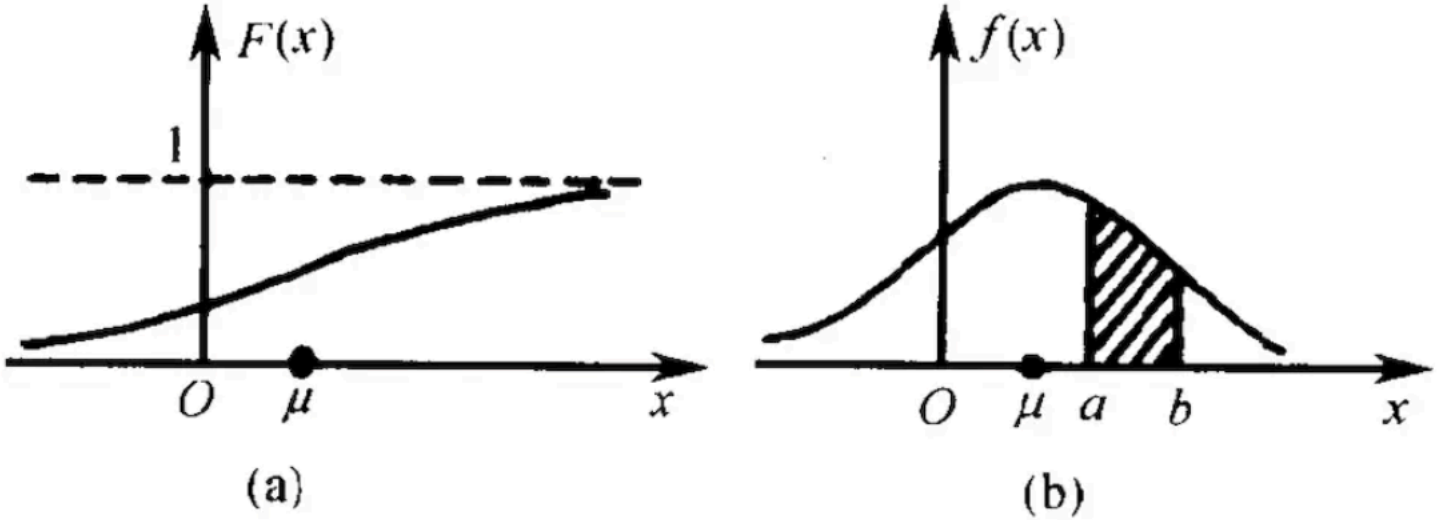
Dice

Probability density function

- The probability density function expressed by mathematical formula is a function of definite integral

$$P(a \leqslant X \leqslant b) = F(b) - F(a) = \int_a^b (x)\mathrm{d}x$$

- F(x) is the distribution function, the probability density function f(x) is the derivative function of the distribution function



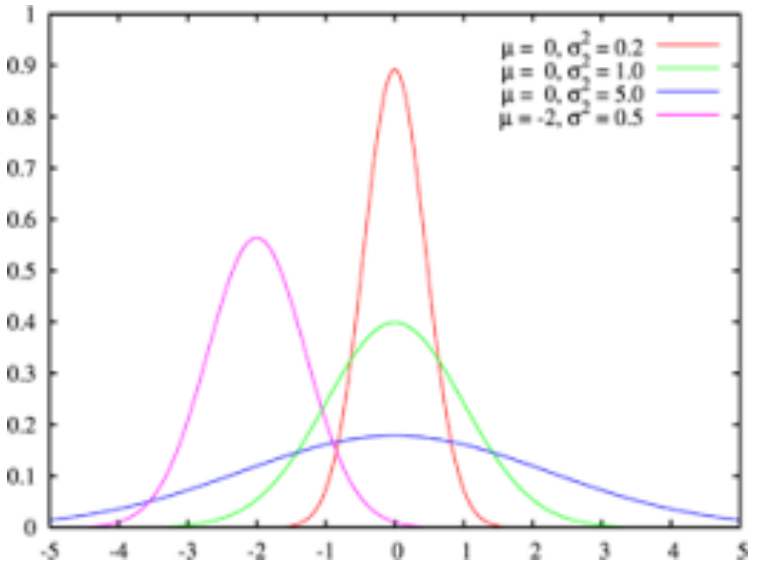
Probability density function

- A probability density function (PDF), is a function whose value at any given sample in the sample space can be interpreted as providing a relative likelihood that the value of the random variable would equal that sample.

Probability density function

$$f(x) \geq 0$$

$$\int_{-\infty}^{+\infty} f(x)dx = 1$$



Probability density function

- Gaussian distribution:
  - If the random variable X obeys a normal distribution with position parameter μ and scale parameter σ

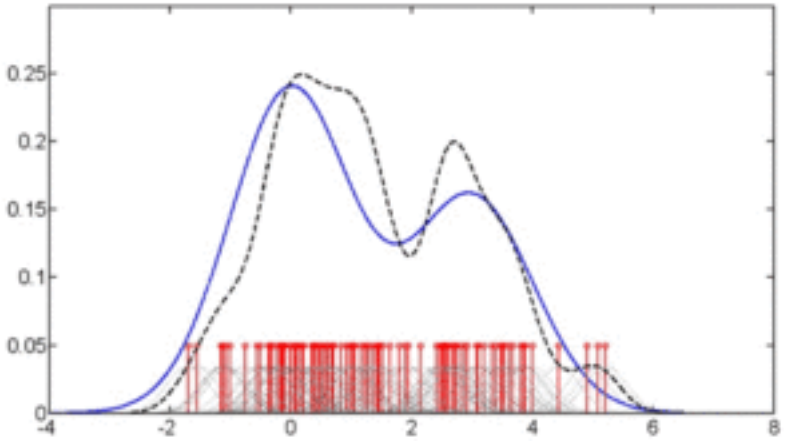
$$X \sim N(\mu, \sigma^2)$$

  - And its probability density function is

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Why we need probability density function (PDF)

- So what is the use of the probability density function?
  - Very useful! For example, we can use PDF to determine the credibility of a sample, and then determine whether the sample is an outlier.
  - In addition, sometimes we need to use PDF if our input data obey a certain distribution.
- But usually we don't know the PDF of a random variable, so we need to continue to approximate the PDF, and the process of approximation is ***probability density estimation (PDE)***.



Histogram Method

- A **histogram** is a **method** that uses bars to display count or frequency data. The independent variable consists of interval- or ratio-level data and is usually displayed on the abscissa (x-axis), and the frequency data on the ordinate (y-axis), with the height of the bar proportional to the count.

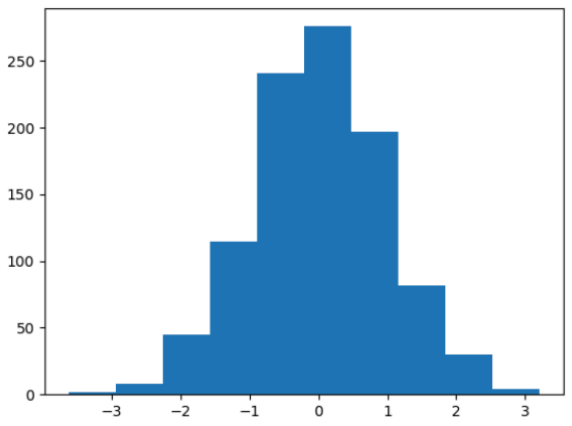
- Example: Group the observations into bins, and then count the number of events in each bin. The count or observation frequency in each box is then represented by a bar graph, with the box on the x-axis and the frequency on the y-axis.

Histogram Method

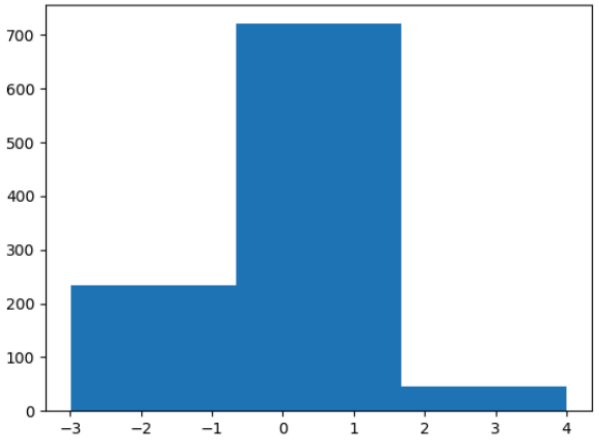
- Suppose the range of the observation value is 1 to 100, then we can divide into the following two ways:
  - 3 boxes (1-33,34-66,67-100): the division is relatively coarse-grained
  - 10 boxes (1-10,11-20,...,91-100): The division is more fine-grained, which can better extract density information, but the amount of calculation will be larger

```
# example of plotting a histogram of a random sample
from matplotlib import pyplot
from numpy.random import normal

# generate a sample
sample = normal(size=1000)
# plot a histogram of the sample
pyplot.hist(sample, bins=10)
pyplot.show()

pyplot.hist(sample, bins=3)
pyplot.show()
```

bin=10



bin=3



Probability density estimation

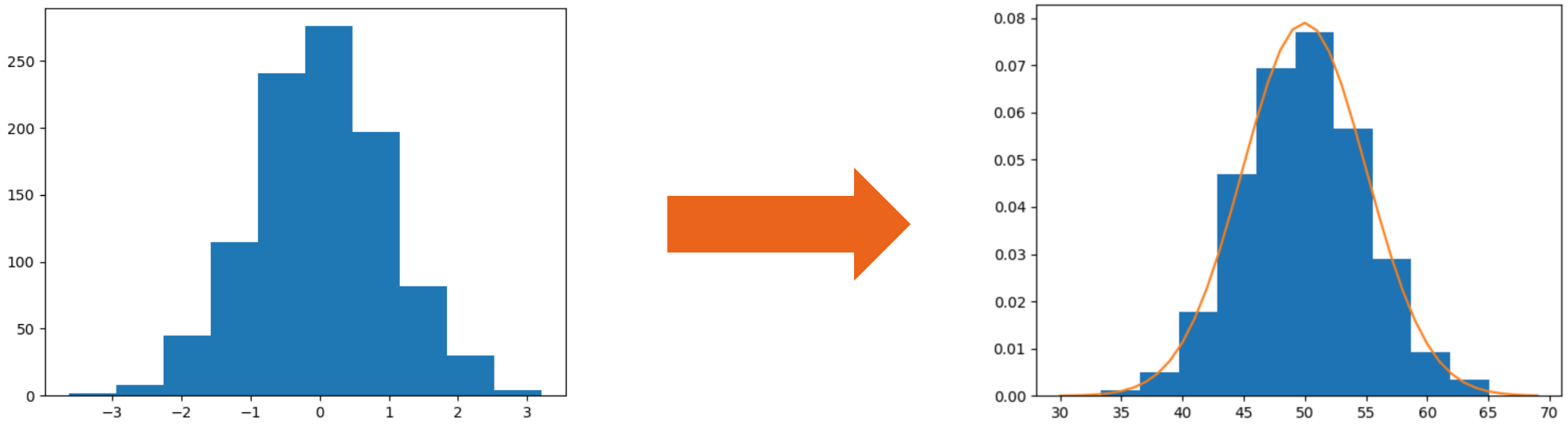
- Parametric Density Estimation
  - According to prior knowledge, it is assumed that random variables obey a certain distribution, and then the parameters of the distribution are estimated through training samples.
  - Estimation method: maximum likelihood estimation

$$\log p(\mathcal{D};\theta) = \sum_{n=1}^{N} \log p(\boldsymbol{x}^{(n)};\theta).$$

- Nonparametric Density Estimation
  - Without assuming that the data obey a certain distribution, the probability density function of the data is approximated by dividing the sample space into different regions and estimating the probability of each region.

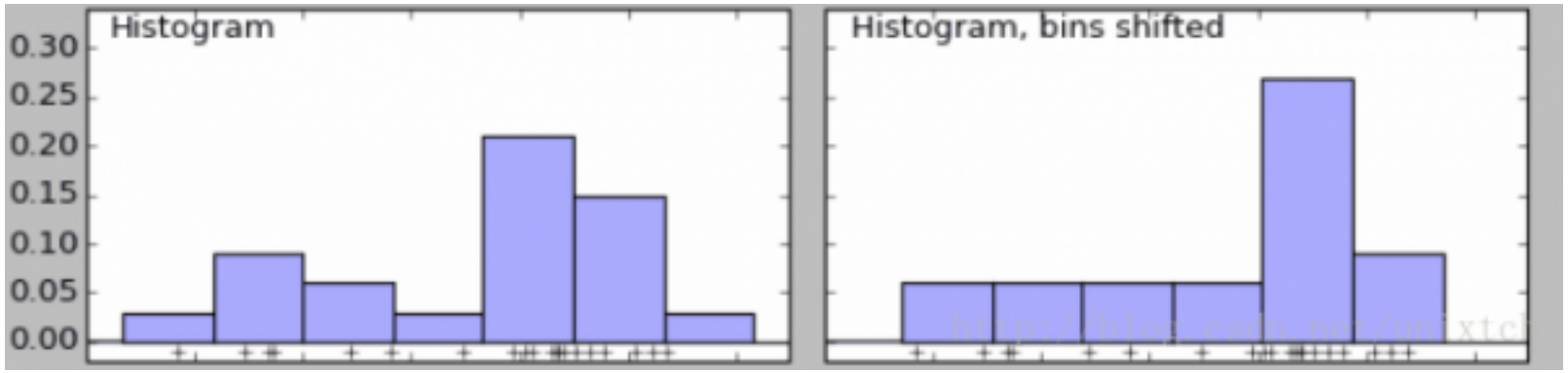
Parametric density estimation example

- Guessing the histogram can roughly guess that it obeys a normal distribution, so only the parameters (μ, σ) of this normal distribution are required later.



Nonparametric density estimation example

- In some cases, a data sample may not look like a common probability distribution, or it is not easy to fit a certain distribution. Especially when the data has two peaks (bimodal distribution) or multiple peaks (multimodal distribution), this situation often occurs

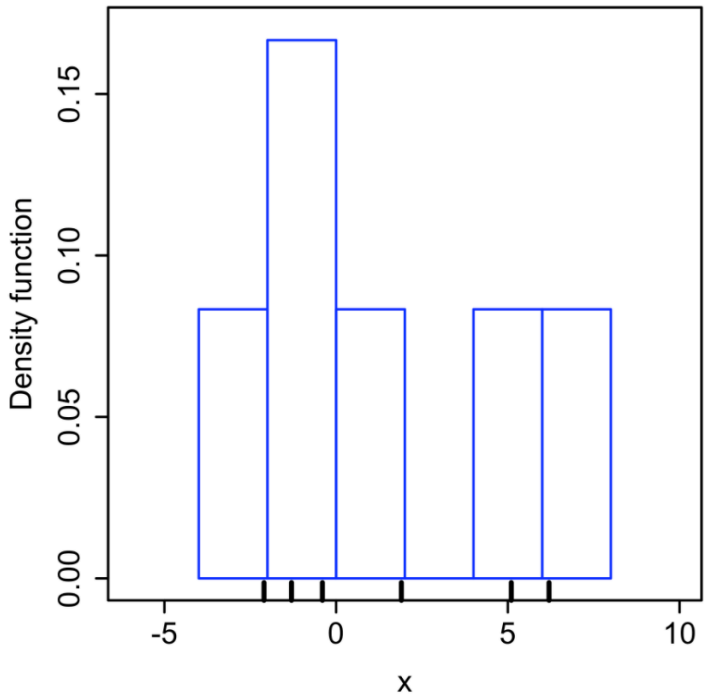


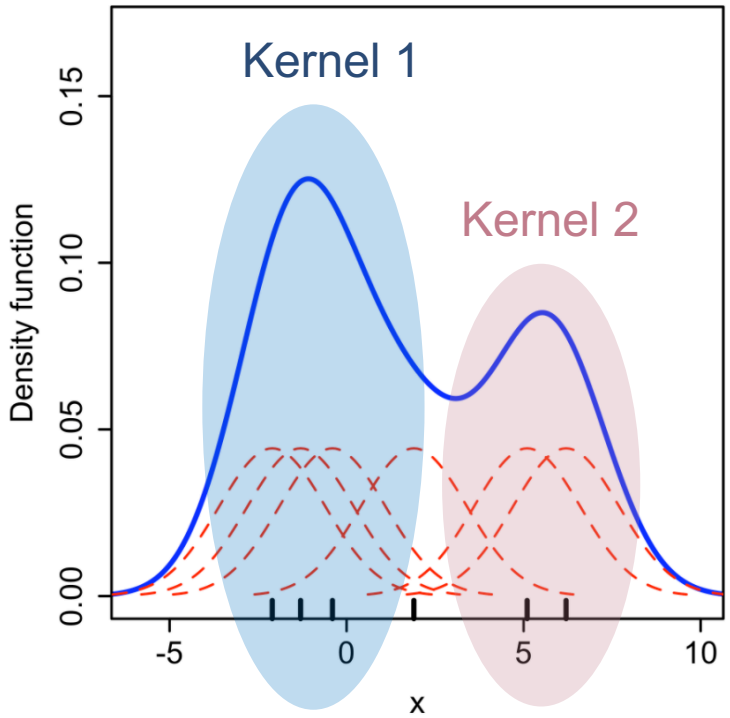
Nonparametric density estimation example

Weight

$$\hat{f}_h(x) = \frac{1}{n}\sum_{i=1}^{n} K_h(x - x_i) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right)$$

Kernel function





Parametric Density Estimation vs. Non-parametric Density Estimation

- Parameter density estimation does not need to retain the entire training set, so it is more efficient in storage and calculation.

- Non-parametric density estimation needs to retain the entire training set.

Parameter density estimation generally has the following problems

- **Model selection problem**
  - How to choose the density function of the data distribution?
  - The distribution of actual data is often very complex, rather than a simple normal distribution or multinomial distribution.

- **Unobservable variable problem**
  - That is, the samples we use for training only contain part of the observable variables, and some very critical variables are unobservable, which makes it difficult for us to accurately estimate the true distribution of the data.

- **Dimensional catastrophe**
  - It is very difficult to estimate the parameters of high-dimensional data
  - As the dimensionality increases, the number of samples required to estimate the parameters increases exponentially. Overfitting occurs when the sample is insufficient.

K-means

K-Means

- In the natural sciences and social sciences, there are a lot of classification problems. The so-called class, refers to a collection of similar elements.
- Cluster analysis originated from taxonomy. In ancient taxonomy, people mainly rely on experience and professional knowledge to achieve classification
- With the development of human science and technology, the requirements for classification are getting higher, so that sometimes it is difficult to classify exactly based on experience and professional knowledge.
- Clustering algorithm is the basic algorithm in unsupervised learning

The difference between clustering and classification

- Clustering:
  - refers to the division of similar data together.
  - The specific division does not care about this type of label.
  - The goal is to aggregate similar data together.
  - Clustering is a kind of unsupervised learning method.

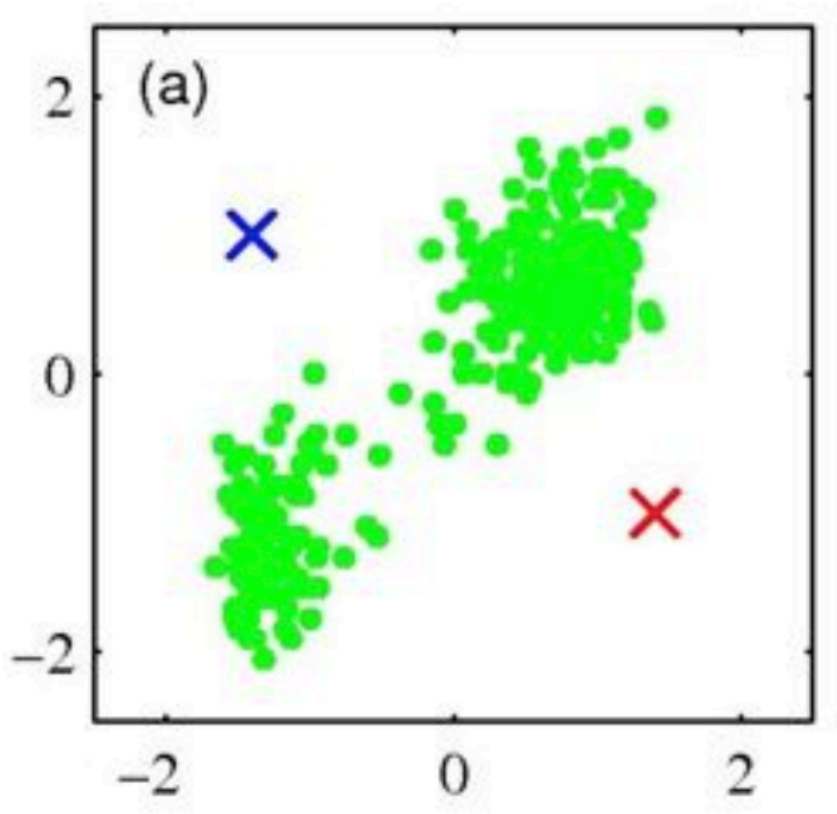
- Classification:
  - Dividing different data.
  - The process is to obtain a classifier through the training data set, and then use the classifier to predict unknown data.
  - Classification is a supervised learning method.

K-Means steps

1. Assuming that we want to cluster N sample observations, and require clustering into K categories, first select K points as the initial center points;
2. Next, according to the principle of the smallest distance from the initial center point, all observations are classified into the class where each center point is located;
3. There are several observations in each category, calculate the mean value of all sample points in the K categories, as the K center points of the second iteration;
4. Then repeat steps 2 and 3 according to this center until convergence (the center point no longer changes or reaches the specified number of iterations), and the clustering process ends.

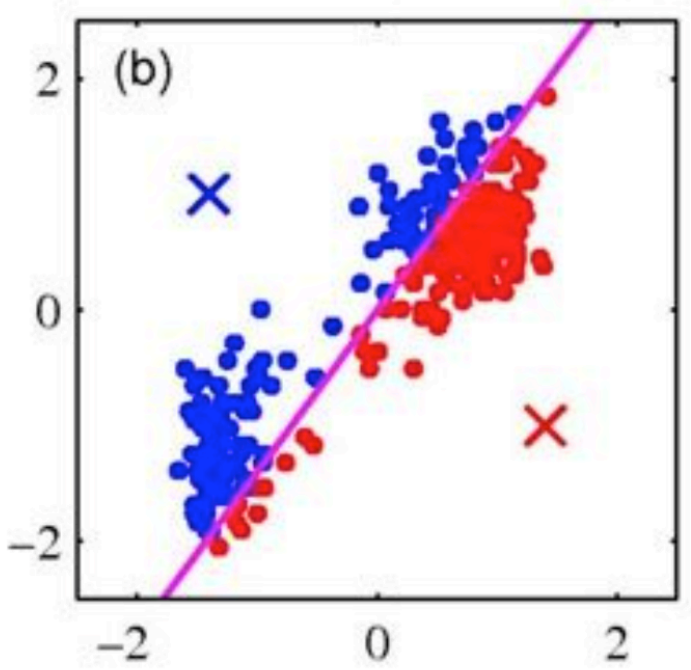
K-Means example

- Now we want to cluster the n green points in (a) into 2 categories, first randomly select the blue cross and the red cross as the initial center points;



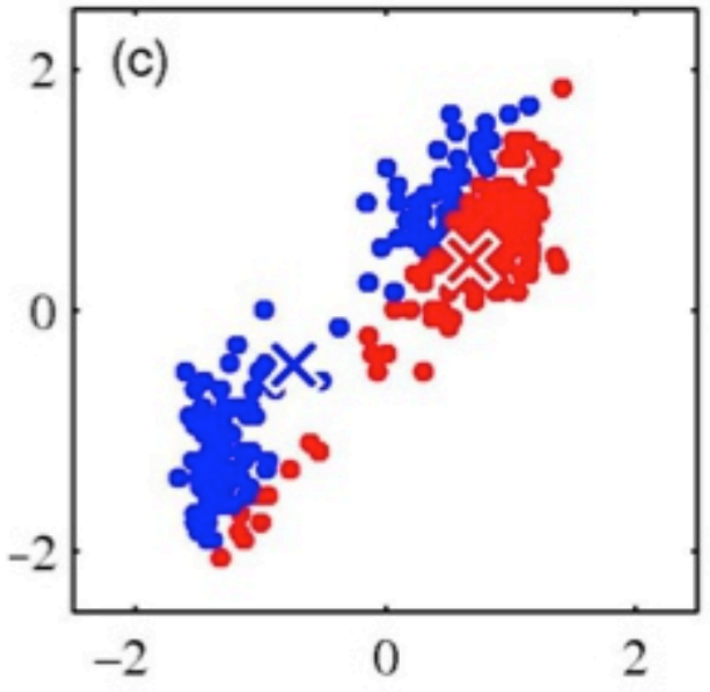
K-Means example

- Calculate the distances from all the points to the initial blue cross and the initial red cross respectively.
- Paint blue when it is closer to the blue cross and red when it is closer to the red cross. Traverse all the points until they are all dyed, such as Figure (b);



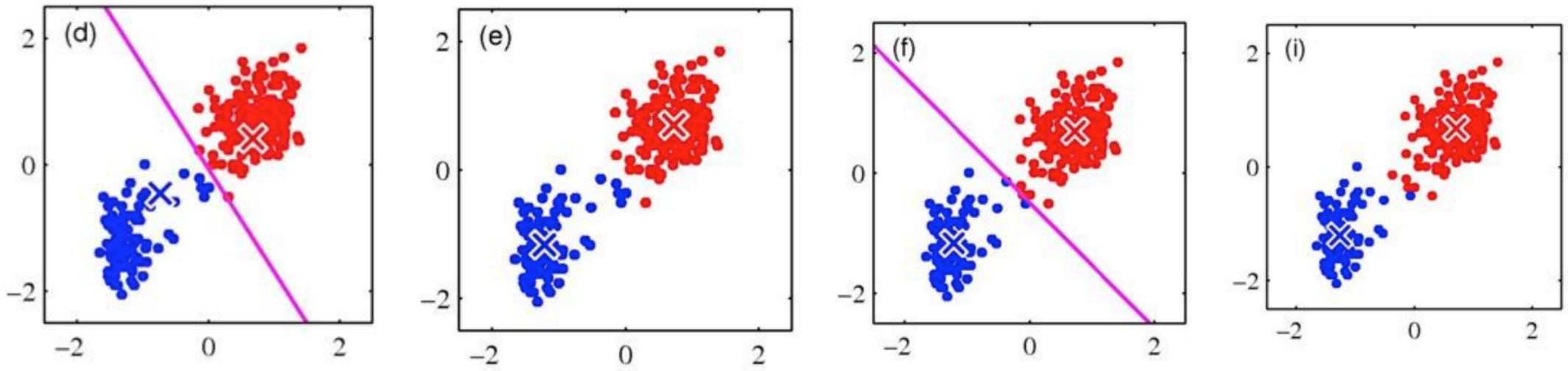
K-Means example

- Now we don't care about the initial blue cross and the initial red cross, calculate the red center of the dyed red point, and the blue point is also the same to get the center of the second iteration, as shown in Figure (c);



K-Means example

- Repeat steps 2 and 3 until convergence and the clustering process ends.



Question: What defines the distance between points in step 2?

- Clustering is to group similar objects together.
- What is the similarity (or distance) used to measure?

- Euclidean Distance

$$X = (x_1, x_2, \ldots, x_n) \qquad Y = (y_1, y_2, \ldots, y_n)$$

n is the number of features

$$dist_{ed}(X, Y) = ||X - Y||2 = \sqrt[2]{(x_1 - y_1)^2 + \ldots + (x_n - y_n)^2}$$

Question: How to calculate the mean (new center point) of all points in the third step?

- We generally use the error sum of squares as the objective function (similar to the loss function in linear regression)

$$SSE = \sum_{i=1}^{K} \sum_{x \in C_i} (C_i - x)^2$$

Calculate the Euclidean distance between the two points

Calculate the distance from all sample points to their center point and add them up

Question: How to calculate the mean (new center point) of all points in the third step?

- Find the extreme value of a function by derivation

$$\begin{aligned}\frac{\partial}{\partial C_k} SSE &= \frac{\partial}{\partial C_k} \sum_{i=1}^{K} \sum_{x \in C_i} (C_i - x)^2 \\ &= \sum_{i=1}^{K} \sum_{x \in C_i} \frac{\partial}{\partial C_k} (C_i - x)^2 \\ &= \sum_{x \in C_i} 2\,(C_i - x) = 0\end{aligned}$$

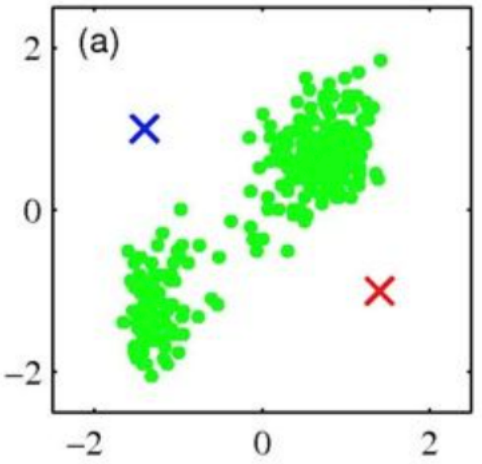
$$\sum_{x \in C_i} 2\,(C_i - x) = 0 \Rightarrow m_i C_i = \sum_{x \in C_i} x \Rightarrow C_i = \frac{1}{m_i} \sum_{x \in C_i} x$$

The solution of C is the mean point of X

$$X_1, \ldots, X_m \ , \ X_i = (x_{i1}, x_{i2}, \ldots, x_{in})$$

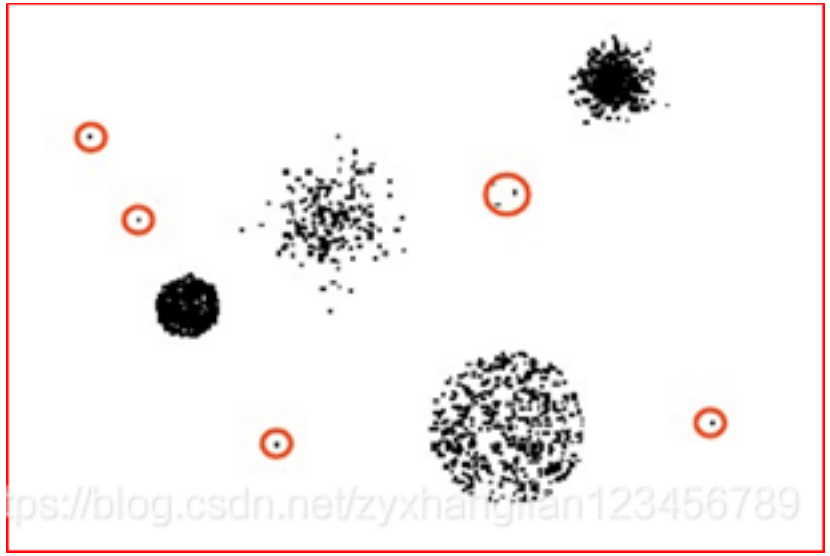
$$C = \left(\frac{x_{11} + \ldots + x_{1n}}{m}, \ldots, \frac{x_{m1} + \ldots + x_{mn}}{m}\right)$$

Question: How to determine the initial center point in the first step?



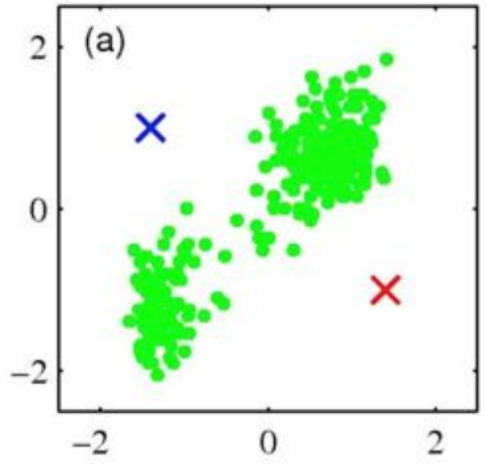
- Method 1:
  - Choose any point as the first initial center C1
  - Calculate the distance between all sample points and C1, the largest distance is selected as the next center C2, until K centers are selected

- Problem of method 1:
  - Outlier problem
  - In statistics, an outlier is a data point that differs significantly from other observations.



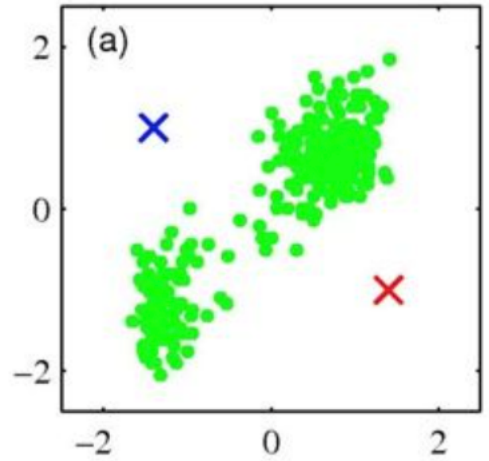
Question: How to determine the initial center point in the first step?

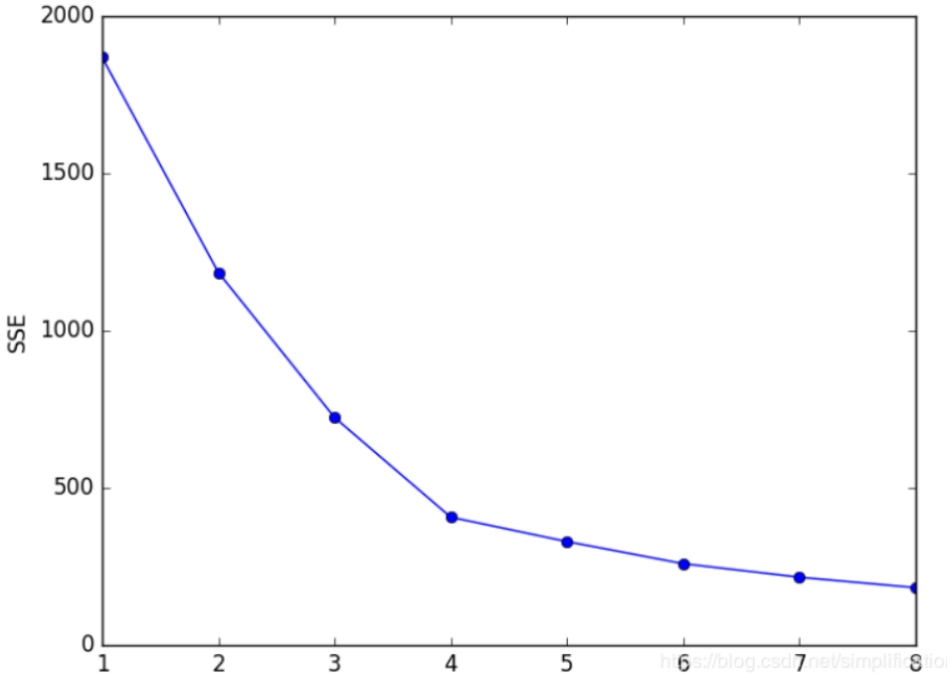
- Method 2:
  - Try a few more times, that is, set a few different initial points, and choose the best, that is, the group with the smallest SSE value as the final cluster



Question: How to determine the K value?

- The larger the K setting, the finer the sample is divided, the higher the degree of aggregation of each cluster, and the smaller the error squared sum SSE will be





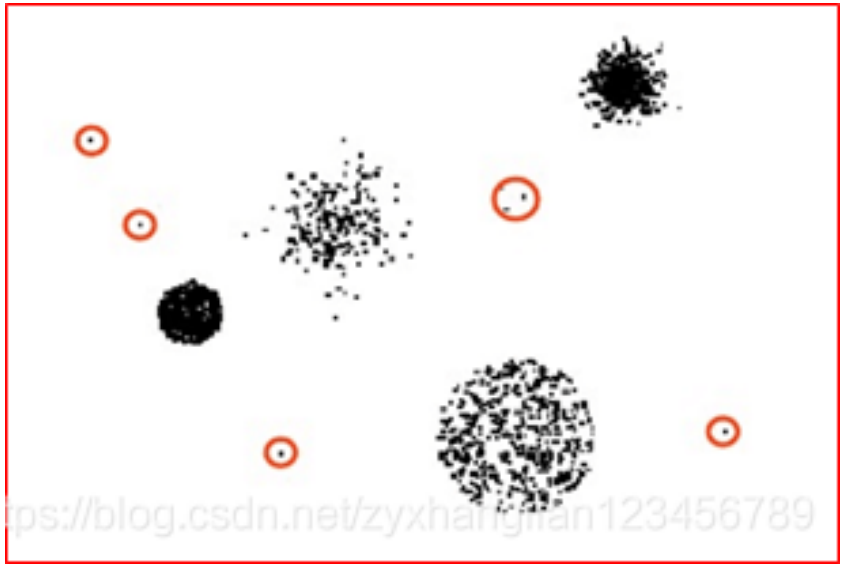
We started with K=2 and found that when K=3, the SSE dropped sharply. When K=4, the SSE declined slightly. When K=5, the decline shrank rapidly, and then it became more and more gentle. So we can set K to 4

Advantages of K-Means

- The principle is simple
- Easy to implement
- The clustering effect is good

Disadvantages of K-Means

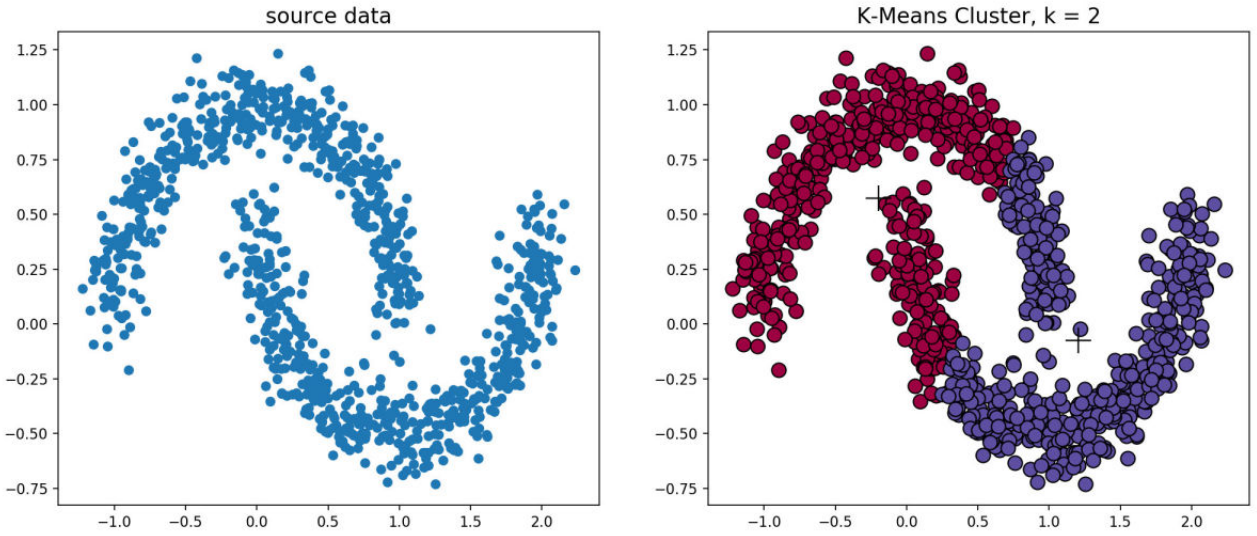
- The selection of K value and initial point is difficult to determine;
- The result obtained is only a local optimum;
- Affected by outliers

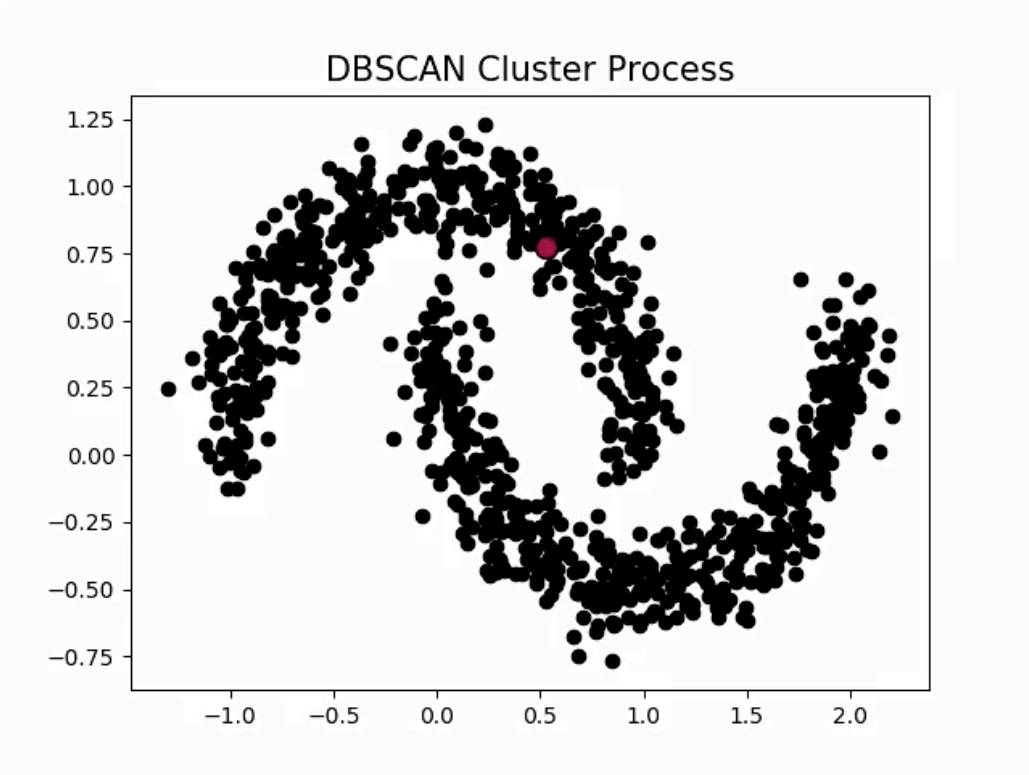


Other clustering methods

- Density-based clustering
- Hierarchical clustering
- …

kmeans clustering produces wrong results





K-means example

- Normalized dataset X

```
1   import random
2   from sklearn import datasets
3   import numpy as np
4   import matplotlib.pyplot as plt
5   from mpl_toolkits.mplot3d import Axes3D
6
7
8   # Normalized dataset X
9   def normalize(X, axis=-1, p=2):
10      lp_norm = np.atleast_1d(np.linalg.norm(X, p, axis))
11      lp_norm[lp_norm == 0] = 1
12      return X / np.expand_dims(lp_norm, axis)
13
```

K-means example

- Calculate the square of the Euclidean distance between a sample and all samples in the data set

```
14
15  # Calculate the square of the Euclidean distance between a sample and all samples in the data set
16  def euclidean_distance(one_sample, X):
17      one_sample = one_sample.reshape(1, -1)
18      X = X.reshape(X.shape[0], -1)
19      distances = np.power(np.tile(one_sample, (X.shape[0], 1)) - X, 2).sum(axis=1)
20      return distances
21
```

K-means example

- K-means class function

```
42    # Randomly select self.k samples from all samples as the initial cluster center
43        def init_random_centroids(self, X):
44            n_samples, n_features = np.shape(X)
45            centroids = np.zeros((self.k, n_features))
46            for i in range(self.k):
47                centroid = X[np.random.choice(range(n_samples))]
48                centroids[i] = centroid
49            return centroids
50
51    # Return to the nearest center index of the sample [0, self.k)
52        def _closest_centroid(self, sample, centroids):
53            distances = euclidean_distance(sample, centroids)
54            closest_i = np.argmin(distances)
55            return closest_i
```

K-means example

K-means class function

```
57    # Classify all samples, the classification rule is to classify the sample to the nearest center
58        def create_clusters(self, centroids, X):
59            n_samples = np.shape(X)[0]
60            clusters = [[] for _ in range(self.k)]
61            for sample_i, sample in enumerate(X):
62                centroid_i = self._closest_centroid(sample, centroids)
63                clusters[centroid_i].append(sample_i)
64            return clusters
65
66    # Update the center
67        def update_centroids(self, clusters, X):
68            n_features = np.shape(X)[1]
69            centroids = np.zeros((self.k, n_features))
70            for i, cluster in enumerate(clusters):
71                centroid = np.mean(X[cluster], axis=0)
72                centroids[i] = centroid
73            return centroids
74
75    # Categorize all samples, the index of their category is their category label
76        def get_cluster_labels(self, clusters, X):
77            y_pred = np.zeros(np.shape(X)[0])
78            for cluster_i, cluster in enumerate(clusters):
79                for sample_i in cluster:
80                    y_pred[sample_i] = cluster_i
81            return y_pred
```

K-means example

- K-means class function

```
    # Perform Kmeans clustering on the entire data set X and return its cluster label
    def predict(self, X):
        # Randomly select self.k samples from all samples as the initial cluster center
        centroids = self.init_random_centroids(X)

        # Iterate until the algorithm converges (the cluster center of the previous time and
        #the cluster center of this time almost coincide) or the maximum number of iterations is reached
        for _ in range(self.max_iterations):
            # All are classified, the classification rule is to classify the sample to the nearest center
            clusters = self.create_clusters(centroids, X)
            former_centroids = centroids

            # Calculate new cluster centers
            centroids = self.update_centroids(clusters, X)

            # If there is almost no change in the cluster center, the algorithm has converged, and the iteration is exited
            diff = centroids - former_centroids
            if diff.any() < self.varepsilon:
                break

        return self.get_cluster_labels(clusters, X)
```

K-means example

```
106  def main():
107      # Load the dataset
108      X, y = datasets.make_blobs(n_samples=10000,
109                                 n_features=3,
110                                 centers=[[3,3, 3], [0,0,0], [1,1,1], [2,2,2]],
111                                 cluster_std=[0.2, 0.1, 0.2, 0.2],
112                                 random_state =9)
113
114      # Clustering with Kmeans algorithm
115      clf = Kmeans(k=4)
116      y_pred = clf.predict(X)
117
118
119      # Visual clustering effect
120      fig = plt.figure(figsize=(12, 8))
121      ax = Axes3D(fig, rect=[0, 0, 1, 1], elev=30, azim=20)
122      plt.scatter(X[y==0][:, 0], X[y==0][:, 1], X[y==0][:, 2])
123      plt.scatter(X[y==1][:, 0], X[y==1][:, 1], X[y==1][:, 2])
124      plt.scatter(X[y==2][:, 0], X[y==2][:, 1], X[y==2][:, 2])
125      plt.scatter(X[y==3][:, 0], X[y==3][:, 1], X[y==3][:, 2])
126      plt.show()
127
128
129  if __name__ == "__main__":
130      main()
```

K-means example