初始化

```
#@markdown - **挂载**
from google.colab import drive
drive.mount('GoogleDrive')
```

- **挂载**

```
# #@markdown - **卸载**
# !fusermount -u GoogleDrive
```

代码区

```
#@title Autoencoder-kmeans clustering {display-mode: "both"}
# 该程序主要实现能够提取有效特征表示的自编码器（特征映射维数为3）的训练
#@markdown [参考程序](https://github.com/jswanglp/MyML/blob/master/codes/Neural_network_models/t
%tensorflow_version 1.x
import os, sys
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from scipy.cluster.vq import kmeans2
from mpl_toolkits.mplot3d import Axes3D
from tensorflow.examples.tutorials.mnist import input_data

tf.logging.set_verbosity(tf.logging.ERROR)
```

Autoencoder-kmeans clustering

[参考程序](https://github.com/jswanglp/MyML/blob/master/codes/Neural_network_models/t)

```
print(tf.__version__)
from tensorflow.python.client import device_lib
os.environ["TF_CPP_MIN_LOG_LEVEL"] = "99"
print(device_lib.list_local_devices())
```

```
1.15.0
[name: "/device:CPU:0"
device_type: "CPU"
memory_limit: 268435456
locality {
}
incarnation: 6955788614704154176
, name: "/device:XLA_CPU:0"
device_type: "XLA_CPU"
memory_limit: 17179869184
locality {
}
incarnation: 5437888317619334637
physical_device_desc: "device: XLA_CPU device"
, name: "/device:XLA_GPU:0"
device_type: "XLA_GPU"
memory_limit: 17179869184
locality {
}
incarnation: 13179324886281098758
physical_device_desc: "device: XLA_GPU device"
, name: "/device:GPU:0"
device_type: "GPU"
memory_limit: 7470045594
locality {
  bus_id: 1
  links {
  }
}
incarnation: 7722507684483214406
```

```
#@markdown - 相关函数的定义

# 定义打印进度函数
def print_progress(progress, epoch_num, loss):

        barLength = 30

        assert type(progress) is float, "id is not a float: %r" % id
        assert 0 <= progress <= 1, "variable should be between zero and one!"

        status = ""

        if progress >= 1:
                progress = 1
                status = "\r\n"

        indicator = int(round(barLength*progress))

        list = [str(epoch_num), "#"*indicator , ">"*(barLength-indicator), progress*100, l
        text = "\rEpoch {0[0]} {0[1]} {0[2]} {0[3]:.2f}% completed, total reconstructior
        sys.stdout.write(text)
        sys.stdout.flush()

# 提取 MNIST 数据集中的 0, 1 图像的函数
def extraction_fn(data):
        index_list = []
        for idx in range(data.shape[0]):
                if data[idx] == 0 or data[idx] == 1:
                        index_list.append(idx)
        return index_list

# 将数字标签转换为颜色符号的函数
def index_to_color(idx):
        color_list = []
        for i in idx:
                if i == 0:
                        color_list += 'b'
                else: color_list += 'g'
        return color_list

# Xavier Glorot 参数初始化
def glorot_init(shape, name):
        initial = tf.truncated_normal(shape=shape, stddev=1. / tf.sqrt(shape[0] / 2.))
```

- 相关函数的定义

```
        return tf.Variable(initial, name=name)

# bias 参数初始化
def bias_init(shape, name):
        initial = tf.constant(0.1, shape=shape)
        return tf.Variable(initial, name=name)
```

```
#@markdown - 模型超参定义

learning_rate = 3e-3 #@param {type:"number"}
num_epochs = 150 #@param {type:"integer"}
batch_size = 48 #@param {type:"integer"}

# 隐层参数
num_hidden1 = 128
num_hidden2 = 3
num_input = 784

display_step = 10
event_path = './GoogleDrive/My Drive/Colab Notebooks/Tensorboard'
checkpoint_path = './Checkpoints'
```

- 模型超参定义

learning_rate: 3e-3

num_epochs: 150

batch_size: 48

```
#@markdown - 图像预处理

# 提取包含 0, 1 图像及其标签
mnist = input_data.read_data_sets("MNIST_data", one_hot=False)
data = {}
index_list_train = extraction_fn(mnist.train.labels)
index_list_test = extraction_fn(mnist.test.labels)
data['train_imgs'], data['train_lbs'] = mnist.train.images[index_list_train], mnist.train.lab
data['test_imgs'], data['test_lbs'] = mnist.test.images[index_list_test], mnist.test.labels[i
data['train_imgs_lbs'] = np.c_[data['train_imgs'], data['train_lbs']]
num_samples, num_features = data['train_imgs'].shape
```

- 图像预处理

```
Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
```

```
#@markdown - 构建网络图
graph = tf.Graph()
with graph.as_default():

        # 权重参数及偏置
        with tf.name_scope('Weights_and_bisaes'):
                weights = {
                        'encoder_w1': glorot_init([num_features, num_hidden1], name='encoder_w
                        'encoder_w2': glorot_init([num_hidden1, num_hidden2], name='encoder_w2
                        'decoder_w1': glorot_init([num_hidden2, num_hidden1], name='decoder_w1
                        'decoder_w2': glorot_init([num_hidden1, num_features], name='decoder_w
                }
                biases = {
                        'encoder_b1': bias_init([num_hidden1], name='encoder_b1'),
                        'encoder_b2': bias_init([num_hidden2], name='encoder_b2'),
                        'decoder_b1': bias_init([num_hidden1], name='decoder_b1'),
                        'decoder_b2': bias_init([num_features], name='decoder_b2')
                }

        # 编码器和解码器函数
        with tf.name_scope('Encoder_and_decoder'):
                # 编码器函数
                def encoder(x):
                        layer_1 = tf.nn.sigmoid(tf.add(tf.matmul(x, weights['encoder_w1']),
                                                       biases['e
                        layer_2 = tf.nn.sigmoid(tf.add(tf.matmul(layer_1, weights['encoder_w2'
                                                       biases['e
                        return layer_2
                # 解码器函数
                def decoder(x):
                        layer_1 = tf.nn.sigmoid(tf.add(tf.matmul(x, weights['decoder_w1']),
                                                       biases['d
                        layer_2 = tf.nn.sigmoid(tf.add(tf.matmul(layer_1, weights['decoder_w2'
                                                       biases['d
                        return layer_2
        # 主网络结构
        with tf.name_scope('Main_structure'):
                with tf.name_scope('Input'):
                        X = tf.placeholder("float", [None, num_features], name='input_images
                        encoder_op = encoder(X)
                with tf.name_scope('Output'):
                        y_pred = decoder(encoder_op)
                with tf.name_scope('Loss'):
                        # 重构误差（平方差）
                        loss = tf.reduce_mean(tf.pow(X - y_pred, 2))
                        # 重构误差（交叉熵）
                        # loss = -tf.reduce_mean(X * tf.log(1e-10 + y_pred) + (1 - X)
                        # loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(lab
                with tf.name_scope('Train'):
                        train_op = tf.train.AdamOptimizer(learning_rate).minimize(loss)
        # summaries 的定义
        tf.summary.image('input_imgs', tf.reshape(X, [-1, 28, 28, 1]), max_outputs=3, coll
        tf.summary.image('reconstructed_imgs', tf.reshape(y_pred, [-1, 28, 28, 1]), max_out
        tf.summary.scalar('loss', loss, collections=['train'])
        summ_train = tf.summary.merge_all('train')
```

- 构建网络图

```
#@markdown - 模型的训练
with tf.Session(graph=graph) as sess:
  sess.run(tf.global_variables_initializer())

  summ_writer = tf.summary.FileWriter(event_path)
  summ_writer.add_graph(sess.graph)

  max_batch = num_samples // batch_size
  for epoch_num in range(num_epochs):
          np.random.shuffle(data['train_imgs'])
          for batch_num in range(max_batch):
                  index_start = batch_num * batch_size
                  index_end = (batch_num + 1) * batch_size
                  imgs_batch = data['train_imgs'][index_start:index_end, :]
```

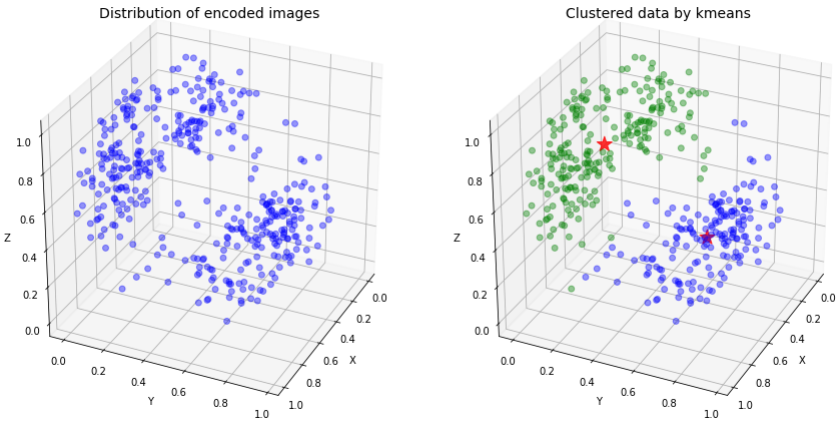
- 模型的训练

```
                _, batch_loss = sess.run([train_op, loss], feed_dict={X: imgs_bat

            total_loss, rs = sess.run([loss, summ_train], feed_dict={X: data['train_i
            summ_writer.add_summary(rs, global_step=epoch_num)

            progress = float(epoch_num % display_step + 1) / display_step
            print_progress(progress, epoch_num + 1, total_loss)

    print('Training completed.')

    # 编码需要显示的 400 幅图像
    encoder_imgs = sess.run(encoder_op, feed_dict={X: data['train_imgs'][:400]})
```

```
Epoch 10 ##############################  100.00% completed, total reconstruction loss: 0.0242.
Epoch 20 ##############################  100.00% completed, total reconstruction loss: 0.0214.
Epoch 30 ##############################  100.00% completed, total reconstruction loss: 0.0198.
Epoch 40 ##############################  100.00% completed, total reconstruction loss: 0.0187.
Epoch 50 ##############################  100.00% completed, total reconstruction loss: 0.0178.
Epoch 60 ##############################  100.00% completed, total reconstruction loss: 0.0174.
Epoch 70 ##############################  100.00% completed, total reconstruction loss: 0.0171.
Epoch 80 ##############################  100.00% completed, total reconstruction loss: 0.0169.
Epoch 90 ##############################  100.00% completed, total reconstruction loss: 0.0167.
Epoch 100 ##############################  100.00% completed, total reconstruction loss: 0.0168.
Epoch 110 ##############################  100.00% completed, total reconstruction loss: 0.0165.
Epoch 120 ##############################  100.00% completed, total reconstruction loss: 0.0164.
Epoch 130 ##############################  100.00% completed, total reconstruction loss: 0.0164.
Epoch 140 ##############################  100.00% completed, total reconstruction loss: 0.0162.
Epoch 150 ##############################  100.00% completed, total reconstruction loss: 0.0162.
Training completed
```

```
#@markdown - 通过 k-means 函数对特征表示进行聚类
mu, label = kmeans2(encoder_imgs, k=2, iter=10)

# 结果显示
titles = ['Distribution of encoded images', 'Clustered data by kmeans']
index_list = [np.zeros((400,), dtype=int), label]
fig = plt.figure(1, figsize=(16, 8))
fig.subplots_adjust(wspace=0.01, hspace=0.02)
for i, title, idx in zip([1, 2], titles, index_list):
        ax = fig.add_subplot(1, 2, i, projection='3d')
        color = index_to_color(idx)
        ax.scatter(encoder_imgs[:, 0], encoder_imgs[:, 1], encoder_imgs[:, 2], c=color, s=
        ax.set_xlabel('X')
        ax.set_ylabel('Y')
        ax.set_zlabel('Z')
        ax.view_init(elev=30., azim=25)
        ax.set_title(title, fontsize=14)
ax.scatter(mu[:, 0], mu[:, 1], mu[:, 2], c='r', s=250, alpha=0.8, marker='*')
plt.show()
```

- 通过 k-means 函数对特征表示进行聚类




```
# !kill 1268
```

```
# %load_ext tensorboard
%tensorboard --logdir='./GoogleDrive/My Drive/Colab Notebooks/Tensorboard'
```

TensorBoard SCALARS IMAGES GRAPHS INACTIVE

Show data download links

Ignore outliers in chart scaling

Tooltip sorting method: default

Smoothing

0.6

Horizontal Axis

STEP RELATIVE WALL

Runs

Write a regex to filter runs

TOGGLE ALL RUNS

./GoogleDrive/My Drive/Colab Notebooks/Tensorboard

Filter tags (regular expressions supported)

loss 1