“基于 RNN,CNN,GAN 算法实现”
实验报告

课程名称：人工智能导论

专业：数据科学与大数据技术

年级：2017 级

姓名：刘颖凡

学号：10172100123

目录

一、实验目的

1. 基于 RNN 实现文本分类任务，掌握搭建并训练 RNN 网络来提取特征，最后通过一个全连接层实现分类目标；
2. 基于 CIFAR-10 数据集，利用 CNN 完成图像分类任务；
3. 基于 MNIST 数据集，使用 GAN 实现手写图像生成任务。

二、实验前期准备

1. 数据集来源：所有数据集放在 data 文件夹下

   搜狐新闻数据：sohu.csv

   CIFA 数据集：https://www.cs.toronto.edu/~kriz/cifar.html

   MNIST 数据集：train-images.idx3-ubyte

   train-labels.idx1-ubyte

2. 开发环境：Python 3.6 (jupyter notebook)

   Keras 2.2.4

   Jieba 0.38

   Tensorflow 1.13.1

   Skimage 0.14.3

三、实验内容

(一) 基于 RNN 实现文本分类任务

1. 数据导入及预处理

本次实验中用到搜狐新闻数据集：sohu.csv

1.1 数据预处理

```
data['label'].value_counts()
```

```
news        2989
sports      1200
business    1051
pic          312
yule         185
mil           95
caipiao       45
cul           44
Name: label, dtype: int64
```

观察数据，发现 pic,yule,mil,caipiao,cul 五类数据量较少，考虑到实验的准确度要求，将这五类数据合并为新类 others

```
#将pic,yule,mil,caipiao,cul合并成一个新的标签：others
data['label'].replace('pic','others',inplace = True)
data['label'].replace('yule','others',inplace = True)
data['label'].replace('mil','others',inplace = True)
data['label'].replace('caipiao','others',inplace = True)
data['label'].replace('cul','others',inplace = True)
```

```
data['label'].value_counts()
```

```
news        2989
sports      1200
business    1051
others       681
Name: label, dtype: int64
```

1.2 利用 sklearn 封装模块，将新的 labels 映射成数字，便于模型的搭建

```
#将类型映射成数字，方便处理
from sklearn import preprocessing

le = preprocessing.LabelEncoder()
le.fit(data['label'])
data['label'] = le.transform(data['label'])
```

1.3 利用 jieba 分词对文本进行分词及去除停用词处理

下载停用词表 stopword.txt，将停用词构建成列表形式

```
#对文本进行分词并去除停用词
def drop_stopwords(sentences):
    words = jieba.cut(sentences,cut_all=False)
    out_text =[]
    for word in words:
        if word not in stopwords and len(word)>1:
            out_text.append(word)
    return out_text
```

```
data['text'] = data['text'].apply(lambda x:' '.join(drop_stopwords(x)))
```

```
data.head(10)
```

	label	text
0	2	高清 彭帅 谢淑薇 遗憾 逆转 出局 击掌 鼓励 责编 马克 日期 2013 10 彭帅 谢...
1	1	河南 周口 路边 秸秆 燃烧 幼儿 10 下午 崔先生 驾车 周口 郸城县 汲冢镇 走亲戚 ...
2	1	消息 138 中国 劳工 菲律宾 使馆 核实 使馆 核实 情况 中国 菲律宾 大使馆 发言人...
3	1	越南 军事 领导人 武元甲 大将 去世 享年 102 美国 媒体 10 报道 越南 抗法 抗...
4	1	西沙 搜救 发现 遇难 渔民 遗体 尚有 52 失踪 发现 遇难 渔民 遗体 尚有 52 失...
5	2	10 十佳 李娜 小德 球迷 抢眼 大威 文静 责编 马克 日期 2013 10 小德 支持...
6	2	高清 小德 速胜 高举 双臂 庆祝 鼓掌 球迷 致意 责编 马克 日期 2013 10 小德...
7	3	卡努特 伤愈 首发 战辽足 斯塔诺 辽足 不好 对付 高清 国安 踩场 卡努特 卖力 对抗赛...
8	1	宙斯 系统 成功 反导 试验 新华网 华盛顿 10 日电 记者 小春 美国国防部 成功 反导...
9	1	中国 建筑 俄罗斯 身亡 中国 建筑 俄罗斯 身亡 伊尔库茨克州 中国 建筑工人 在建 住宅...

2. 模型搭建

2.1 利用 keras 模块提取数据特征

```
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.utils import to_categorical

maxlen = 100
max_features = 10000
tokenizer = Tokenizer(max_features)
tokenizer.fit_on_texts(data['text'])
sequences = tokenizer.texts_to_sequences(data['text'])
x = pad_sequences(sequences, maxlen)

#将标签变成one-hot形式
y = to_categorical(data['label'])
```

把训练与测试数据放在一起提取特征，使用 keras 的 Tokenizer 来实现，将新闻文档处理成单词索引序列；将长度不足 100 的新闻用 0 填充（在前端填充），用 keras 的 pad_sequences 实现；最后将标签处理成 one-hot 向量，比如 6 变成了 [0,0,0,0,0,0,1,0,0,0,0,0,0]，用 keras 的 to_categorical 实现

2.2 划分训练集，测试集

```
# 划分训练集、测试集
from sklearn.cross_validation import train_test_split

x1_train, x1_test,y1_train,y1_test = train_test_split(x,y ,test_size=0.3)
```

2.3 模型搭建

本次模型使用 keras 封装 LSTM 实现，LSTM 通常用于处理较长序列

```
from keras.layers import LSTM

model = Sequential()
model.add(Embedding(max_features,32))
model.add(LSTM(32))
model.add(Dense(4,activation='sigmoid'))

model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['acc'])

history = model.fit(xl_train,yl_train,
                    epochs = 10,
                    batch_size=50,
                    validation_split=0.2)
```

Sequential:顺序模型

Embedding：词向量表示

Activation：激活层，通过激活函数对张量进行激活

Dense：全连接，因为有 4 个 labels，参数设置为 4

model.compile：进行编译

optimizer：随机梯度下降优化器，这里使用的是 rmsprop

loss='binary_crossentropy'：二分类用的 one-hot 交叉熵

model.fit：进行 10 轮，批次为 50 的训练，默认训练过程中是会加入正则化防止过拟合

3. 训练结果展示

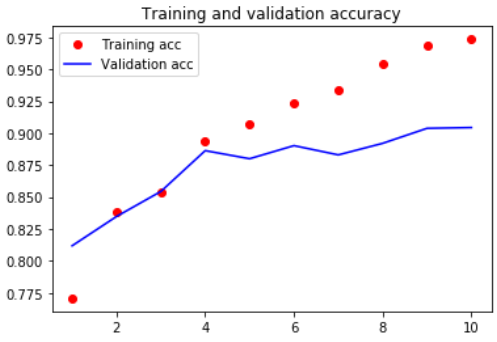
```
Epoch 1/10
3315/3315 [==============================] - 3s 1ms/step - loss: 0.5000 - acc: 0.7707 - val_loss: 0.4339 - val_acc: 0.8118
Epoch 2/10
3315/3315 [==============================] - 2s 656us/step - loss: 0.4125 - acc: 0.8381 - val_loss: 0.3978 - val_acc: 0.8347
Epoch 3/10
3315/3315 [==============================] - 2s 658us/step - loss: 0.3553 - acc: 0.8538 - val_loss: 0.3484 - val_acc: 0.8543
Epoch 4/10
3315/3315 [==============================] - 2s 667us/step - loss: 0.2845 - acc: 0.8942 - val_loss: 0.2959 - val_acc: 0.8863
Epoch 5/10
3315/3315 [==============================] - 2s 648us/step - loss: 0.2407 - acc: 0.9068 - val_loss: 0.3193 - val_acc: 0.8800
Epoch 6/10
3315/3315 [==============================] - 2s 675us/step - loss: 0.1997 - acc: 0.9231 - val_loss: 0.2701 - val_acc: 0.8902
Epoch 7/10
3315/3315 [==============================] - 2s 700us/step - loss: 0.1664 - acc: 0.9337 - val_loss: 0.3137 - val_acc: 0.8830
Epoch 8/10
3315/3315 [==============================] - 2s 722us/step - loss: 0.1374 - acc: 0.9541 - val_loss: 0.3280 - val_acc: 0.8920
Epoch 9/10
3315/3315 [==============================] - 2s 700us/step - loss: 0.1173 - acc: 0.9692 - val_loss: 0.2866 - val_acc: 0.9038
Epoch 10/10
3315/3315 [==============================] - 2s 690us/step - loss: 0.1041 - acc: 0.9735 - val_loss: 0.3009 - val_acc: 0.9044
```

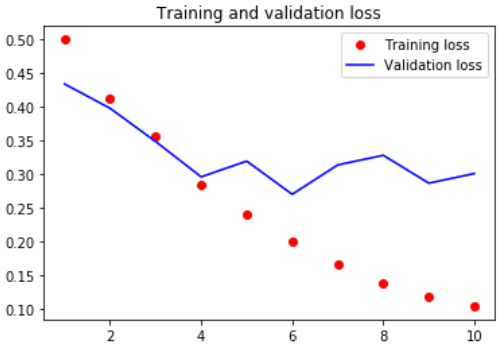
4. 测试结果展示

```
accuracy = model.evaluate(x1_test, y1_test, batch_size = 50)
print("test accuracy:{}".format(accuracy))
```

```
1777/1777 [==============================] - 0s 117us/step
test accuracy:[0.3126860215010506, 0.9040517700342362]
```

测试集准确度达到 90%





图为 accuracy 和 loss 曲线

（二）基于 CIFAR-10 数据集实现 CNN 完成图像分类任务

1. 数据集导入及数据预处理

1.1 设置模型用到的超参数

```
"""
x_train,y_train: 训练的样本的数据和labels
x_test,y_test: 测试的样本的数据和labels
dtype=int,0~255
"""

batch_size = 32
num_classes = 10
data_augmentation = True
```

1.2 数据集导入并划分训练集和测试集

本次实验中用到第 CIFAR-10 数据集：数据集采用 import 方式导入

```
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
```

```
x_train shape: (50000, 32, 32, 3)
50000 train samples
10000 test samples
```

```
#转成one-hot编码
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

2. 模型搭建

```
model = Sequential()

#第一,二层，32个（33）的卷积核，步长为1（默认也是1）
#input_shape：神经网络输入的张量的大小是多少
model.add(Conv2D(32, (3, 3), padding='same',input_shape=x_train.shape[1:]))
#激活函数选择 'relu'
model.add(Activation('relu'))
model.add(Dropout(0.25))
#Maxpooling:池化核的大小，22，步长strides默认和池化大小一致
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3), padding='same',input_shape=x_train.shape[1:]))
model.add(Activation('relu'))
model.add(Dropout(0.25))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3), padding='same',input_shape=x_train.shape[1:]))
model.add(Activation('relu'))
model.add(Dropout(0.25))
model.add(GlobalMaxPooling2D())

# Dense（100）表示把他压成和我们labels一样的维度100，通过softmax进行激活
model.add(Dense(500))
model.add(Activation('relu'))
model.add(Dense(100))
model.add(Activation('sigmoid'))
model.add(Dropout(0.25))
model.add(Dense(num_classes))
model.add(Activation('softmax'))

model.summary()
```

Sequential:顺序模型

Activation：激活层，通过激活函数对张量进行激活

Dense：全连接

Dropout:drouput 层，以一定概率不激活神经元，防止过拟合

Flatten：把多维输入一维化

Conv2D：2d 卷积

MaxPooling2D：2d 下采样,把一维的向量转换为 num_class 维的 One-hot 编码

plot_model：打印建好的模型，相当于可视化模型

3. 训练结果展示

Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 32, 32, 32)	896
activation_7 (Activation)	(None, 32, 32, 32)	0
dropout_5 (Dropout)	(None, 32, 32, 32)	0
max_pooling2d_3 (MaxPooling2	(None, 16, 16, 32)	0
conv2d_5 (Conv2D)	(None, 16, 16, 64)	18496
activation_8 (Activation)	(None, 16, 16, 64)	0
dropout_6 (Dropout)	(None, 16, 16, 64)	0
max_pooling2d_4 (MaxPooling2	(None, 8, 8, 64)	0
conv2d_6 (Conv2D)	(None, 8, 8, 64)	36928
activation_9 (Activation)	(None, 8, 8, 64)	0
dropout_7 (Dropout)	(None, 8, 8, 64)	0
global_max_pooling2d_2 (Glob	(None, 64)	0
dense_4 (Dense)	(None, 500)	32500
activation_10 (Activation)	(None, 500)	0
dense_5 (Dense)	(None, 100)	50100
activation_11 (Activation)	(None, 100)	0
dropout_8 (Dropout)	(None, 100)	0
dense_6 (Dense)	(None, 10)	1010
activation_12 (Activation)	(None, 10)	0

Total params: 139,930
Trainable params: 139,930
Non-trainable params: 0

4. 测试结果展示

```
opt = keras.optimizers.Adam(lr=0.0001)
model.compile(loss='categorical_crossentropy',optimizer=opt,metrics=['accuracy'])

print("train___________")
model.fit(x_train,y_train,epochs=300,batch_size=128,)
print("test____________")
```

model.compile：进行编译

optimizer：随机梯度下降优化器，这里使用的是 Adam，学习率是 0，0001

loss='categorical_crossentropy'：多分类用的 one-hot 交叉熵

model.fit(X_train,y_train,epochs=600,batch_size=128,)：进行 300 轮，批次为 128 的训练，默认训练过程中是会加入正则化防止过拟合

loss,acc = model.evaluate(x_test,y_test)：对样本进行测试，默认不使用正则化，返回损失值和正确率。

```
loss,acc=model.evaluate(x_test,y_test)
print("loss=",loss)
print("accuracy=",acc)
```

```
10000/10000 [==============================] - 5s 453us/step
loss= 0.7131511684417725
accuracy= 0.7541
```

设置 epoch 层为 300，准确率可达 75%

（三）基于 MNIST 数据集实现 GAN 手写图像生成任务

本次实验中用到第三个数据集：train-images.idx3-ubyte，train-labels.idx1-ubyte，提前解压好放在 data 文件夹下

1. 数据预处理.

1.1 设置超参数

```
#图像的size为（28,28,1）
image_width = 28
image_height = 28
image_size = image_width * image_height

#是否训练和存储设置
train = True
restore = False
output_path = "./output_image/"

# set hyperparameters
max_epoch = 300
batch_size = 256
z_size = 220            #生成器的传入参数
h1_size = 300     #第一隐藏层的size，即特征数
h2_size = 300     #第二隐藏层的size，即特征数
```

1.2 将下载好的数据解码成 numpy 可以识别的形式

```
#定义DataLoad()函数将文件数据转为numpy可以读取的格式
def DataLoad(data_path):
    file_data = open(os.path.join(data_path,'train-images.idx3-ubyte'))
    loaded_data = np.fromfile(file = file_data,dtype = np.uint8)
    #前16个字符为说明符，需要跳过
    train_data = loaded_data[16:].reshape((-1,784)).astype(np.float)

    file_label = open(os.path.join(data_path,'train-labels.idx1-ubyte'))
    loaded_label = np.fromfile(file = file_label,dtype = np.uint8)
    #前8个字符为说明符，需要跳过
    train_label = loaded_label[8:].reshape((-1)).astype(np.float)

    return train_data,train_label
```

2. 模型搭建

2.1 构建生成器

```
import tensorflow as tf

def Generator(z_input):
    #第一个链接层
    w1 = tf.Variable(tf.truncated_normal([z_size,h1_size],stddev = 0.1),name = "g_w1",dtype = tf.float32)
    b1 = tf.Variable(tf.zeros([h1_size]),name = "g_b1",dtype = tf.float32)
    h1 = tf.nn.relu(tf.matmul(z_input, w1) + b1)

    #第二个链接层
    w2 = tf.Variable(tf.truncated_normal([h1_size,h2_size],stddev = 0.1),name = "g_w2",dtype = tf.float32)
    b2 = tf.Variable(tf.zeros([h2_size]),name = "g_b2",dtype = tf.float32)
    h2 = tf.nn.relu(tf.matmul(h1, w2) + b2)

    #第三个链接层
    w3 = tf.Variable(tf.truncated_normal([h2_size,image_size],stddev = 0.1),name = "g_w3",dtype = tf.float32)
    b3 = tf.Variable(tf.zeros([image_size]),name = "g_b3",dtype = tf.float32)
    h3 = tf.nn.relu(tf.matmul(h2, w3) + b3)

    #输出：生成图像
    output_generate = tf.nn.tanh(h3)    #利用tanh激活函数，将h3传入输出层

    #输出：生成图像的所有参数
    g_parameters = [w1, w2, w3, b1, b2, b3]              #合并所有参数

    return output_generate,g_parameters
```

w：以 2 倍标准差 stddev 的截断的正态分布中生成大小为[size1， size2]的随机值，权值 weight 初始化

b：生成大小为[size2]的 0 值矩阵，偏置 bias 初始化

h ： 通过矩阵运算，将输入 z_input 传入隐含层 h1。激活函数为 relu

2.2 构建判别器

```
def Discriminator(true_data, generated_data, dropout_rate):

    # 合并输入数据，包括真实数据true_data和生成器生成的假数据generated_data
    sum_data = tf.concat([true_data, generated_data], 0)

    # 第一个链接层
    w1 = tf.Variable(tf.truncated_normal([image_size, h2_size], stddev=0.1), name="d_w1", dtype=tf.float32)
    b1 = tf.Variable(tf.zeros([h2_size]), name="d_b1", dtype=tf.float32)
    h1 = tf.nn.dropout(tf.nn.relu(tf.matmul(sum_data, w1) + b1), dropout_rate)

    # 第二个链接层
    w2 = tf.Variable(tf.truncated_normal([h2_size, h1_size], stddev=0.1), name="d_w2", dtype=tf.float32)
    b2 = tf.Variable(tf.zeros([h1_size]), name="d_b2", dtype=tf.float32)
    h2 = tf.nn.dropout(tf.nn.relu(tf.matmul(h1, w2) + b2), dropout_rate)

    # 第三个链接层
    w3 = tf.Variable(tf.truncated_normal([h1_size, 1], stddev=0.1), name="d_w3", dtype=tf.float32)
    b3 = tf.Variable(tf.zeros([1]), name="d_b3", dtype=tf.float32)
    h3 = tf.matmul(h2, w3) + b3

    #从h3中切出batch_size张图像
    slice_image = tf.nn.sigmoid(tf.slice(h3, [0, 0], [batch_size, -1], name=None))
    #从h3中切除余下的图像
    slice_left_image = tf.nn.sigmoid(tf.slice(h3, [batch_size, 0], [-1, -1], name=None))

    #合并参数
    d_parameters = [w1, w2, w3, b1, b2, b3]

    return slice_image, slice_left_image, d_parameters
```

w：以 2 倍标准差 stddev 的截断的正态分布中生成大小为[size1， size2]的随机值，权值 weight 初始化

b：生成大小为[size2]的 0 值矩阵，偏置 bias 初始化

h ： 通过矩阵运算，将输入 z_input 传入隐含层 h1。激活函数为 relu

2.3 构建显示结果的函数

```
def ShowResult(batch_res, filepath, grid_size=(8, 8), grid_pad=5):

    #将batch_res进行值[0, 1]归一化，同时将其reshape成（batch_size, image_height, image_width）
    batch_res = 0.5 * batch_res.reshape((batch_res.shape[0], image_height, image_width)) + 0.5

    #重构显示图像格网的参数
    re_image_height, re_image_width = batch_res.shape[1], batch_res.shape[2]
    grid_height = re_image_height * grid_size[0] + grid_pad * (grid_size[0] - 1)
    grid_width = re_image_width  * grid_size[1] + grid_pad * (grid_size[1] - 1)
    img_grid = np.zeros((grid_height, grid_width), dtype=np.uint8)
    for i, res in enumerate(batch_res):
        if i >= grid_size[0] * grid_size[1]:
            break
        img = (res) * 255.
        img = img.astype(np.uint8)
        row = (i // grid_size[0]) * (re_image_height + grid_pad)
        col = (i % grid_size[1]) * (re_image_width + grid_pad)
        img_grid[row:row + re_image_height, col:col + re_image_width] = img
    #保存图像
    imsave(filepath, img_grid)
```

2.4 开始训练

```
def StartTrain():

    # 加载数据
    train_data, train_label = DataLoad("./data/MNIST_data")
    size = train_data.shape[0]

    # 构建模型————————————————————————————————————————————————
    # 定义GAN网络的输入，其中x_data为[batch_size, image_size]，z_input为[batch_size, z_size]
    x_data = tf.placeholder(tf.float32, [batch_size, image_size], name="x_data") # (batch_size, image_size)
    z_input = tf.placeholder(tf.float32, [batch_size, z_size], name="z_input") # (batch_size, z_size)
    # 定义dropout率
    dropout_rate = tf.placeholder(tf.float32, name="dropout_rate")
    global_step = tf.Variable(0, name="global_step", trainable=False)

    # 利用生成器生成数据x_generated和参数g_params
    x_generated, g_params = Generator(z_input)
    # 利用判别器判别生成器的结果
    y_data, y_generated, d_params = Discriminator(x_data, x_generated, dropout_rate)

    # 定义判别器和生成器的loss函数
    d_loss = - (tf.log(y_data) + tf.log(1 - y_generated))
    g_loss = - tf.log(y_generated)

    # 设置学习率为0.0001，用AdamOptimizer进行优化
    optimizer = tf.train.AdamOptimizer(0.0001)

    # 判别器discriminator 和生成器 generator 对损失函数进行最小化处理
    d_trainer = optimizer.minimize(d_loss, var_list=d_params)
    g_trainer = optimizer.minimize(g_loss, var_list=g_params)
    # 模型构建完毕————————————————————————————————————————————————

    # 全局变量初始化
    init = tf.global_variables_initializer()

    # 启动会话sess
    saver = tf.train.Saver()
    sess = tf.Session()
    sess.run(init)
```

```
    # 判断是否需要存储
    if restore:
        #若是，将最近一次的checkpoint点存到outpath下
        chkpt_fname = tf.train.latest_checkpoint(output_path)
        saver.restore(sess, chkpt_fname)
    else:
        #若否，判断目录是存在，如果目录存在，则递归的删除目录下的所有内容，并重新建立目录
        if os.path.exists(output_path):
            shutil.rmtree(output_path)
        os.mkdir(output_path)

    # 利用随机正态分布产生噪声影像，尺寸为(batch_size, z_size)
    z_sample_val = np.random.normal(0, 1, size=(batch_size, z_size)).astype(np.float32)
```

```
    # 逐个epoch内训练
    for i in range(sess.run(global_step), max_epoch):
        # 图像每个epoch内可以放(size // batch_size)个size
        for j in range(size // batch_size):
            if j%20 == 0:
                print("epoch:%s, iter:%s" % (i, j))

            # 训练一个batch的数据
            batch_end = j * batch_size + batch_size
            if batch_end >= size:
                batch_end = size - 1
            x_value = train_data[ j * batch_size : batch_end ]
            # 将数据归一化到[-1, 1]
            x_value = x_value / 255.
            x_value = 2 * x_value - 1

            # 以正太分布的形式产生随机噪声
            z_value = np.random.normal(0, 1, size=(batch_size, z_size)).astype(np.float32)
            # 每个batch下，输入数据运行GAN，训练判别器
            sess.run(d_trainer,
                     feed_dict={x_data: x_value, z_input: z_value, dropout_rate: np.sum(0.7).astype(np.float32)})
            # 每个batch下，输入数据运行GAN，训练生成器
            if j % 1 == 0:
                sess.run(g_trainer,
                         feed_dict={x_data: x_value,z_input: z_value, dropout_rate: np.sum(0.7).astype(np.float32)})
        # 每一个epoch中的所有batch训练完后，利用z_sample测试训练后的生成器
        x_gen_val = sess.run(x_generated, feed_dict={z_input: z_sample_val})
        # 每一个epoch中的所有batch训练完后，显示生成器的结果，并打印生成结果的值
        ShowResult(x_gen_val, os.path.join(output_path, "sample%s.jpg" % i))
        print(x_gen_val)
        # 每一个epoch中，生成随机分布以重置z_random_sample_val
        z_random_sample_val = np.random.normal(0, 1, size=(batch_size, z_size)).astype(np.float32)
        # 每一个epoch中，利用z_random_sample_val生成手写数字图像，并显示结果
        x_gen_val = sess.run(x_generated, feed_dict={z_input: z_random_sample_val})
        ShowResult(x_gen_val, os.path.join(output_path, "random_sample%s.jpg" % i))
        # 保存会话
        sess.run(tf.assign(global_step, i + 1))
        saver.save(sess, os.path.join(output_path, "model"), global_step=global_step)
```

3. 结果展示

设置 epoch 为 300，将结果图片存到 output-image 文件夹中。

观察图像，发现图像随着 epoch 的增大越来越清晰。

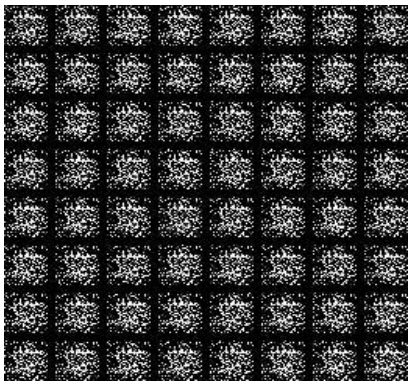


图 1：epoch 为 1

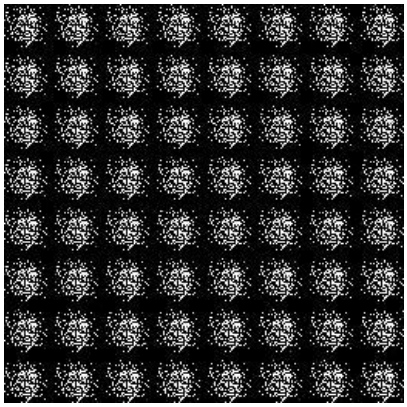


图 2：epoch 为 10

图 3：epoch 为 50

图 4：epoch 为 100

图 5：epoch 为 300

四、总结与体会

1. 整个实验主要通过理论学习机器学习相关算法，了解原理机制，在此基础上通过调用 keras 库函数以直接实现相关算法，以辅助对理论学习的理解；

2. 安装配置一些库时也遇到了一些环境配置的问题，积累了更多解决相关问题的经验；

3. 在进行实验的过程中，巩固了对 python 语言的掌握。