第三次作业

作业要求：

1、基于RNN实现文本分类任务，数据使用搜狐新闻数据(SogouCS, 网址：http://www.sogou.com/labs/resource/cs.php)。任务重点在于搭建并训练RNN网络来提取特征，最后通过一个全连接层实现分类目标。
可以参考https://zhuanlan.zhihu.com/p/26729228
2、基于CIFAR-10数据集（https://www.cs.toronto.edu/~kriz/cifar.html）使用CNN完成图像分类任务。
3、基于MNIST数据集（http://yann.lecun.com/exdb/mnist/）使用GAN实现手写图像生成的任务。
在以上实验中可能因为数据集较大但同学们资源不足的情况，大家可以根据自己的资源情况筛选数据量来进行训练，核心在于练习基于不同任务训练cnn、rnn和gan模型。
实验结果在6.14号前上传到git上，新建一个第三次作业文件夹，将相关代码和报告放到里面。

Task 1： RNN

思路：

1. 将文本用jieba 分词

```
# 分词
datalist = list(data['text'])
length = len(datalist)
# 分词
# 定义分词函数
sentences = []
for i in range(length):
    split_text = list(jieba.cut(datalist[i]))
    sentences.append(split_text)# list 的list
sentences[:2]
```

2. 用word2vec 处理分词后的数据

```
# 训练Word2vec模型

model = Word2Vec(
                 size=100,  # 词向量维度
                 min_count=5,  # 词频阈值
                 window=5)  # 窗口大小
model.build_vocab(sentences)
model.train(sentences,total_examples = model.corpus_count,epochs = model.iter)
model.save('souhu.model')# 保存模型
```

3. 得到一个权重矩阵
4. 搭建模型结构： 用了LSTM

```
def train_lstm(p_n_symbols, p_embedding_weights, p_X_train, p_y_train, p_X_test, p_y_test):
    print (u'创建模型...')
    model = Sequential()
    model.add(Embedding(output_dim=vocab_dim,
                        input_dim=p_n_symbols,
                        mask_zero=True,
                        weights=[p_embedding_weights],
                        input_length=input_length))

    model.add(LSTM(output_dim=50,
                   activation='sigmoid',
                   inner_activation='hard_sigmoid'))
    model.add(Dropout(0.5))
    model.add(Dense(1))
    model.add(Activation('sigmoid'))
```

```
    print (u'编译模型...')
    model.compile(loss='binary_crossentropy',
                  optimizer='adam',
                  metrics=['accuracy'])

    print (u"训练...")
    print(model.summary())
    model.fit(p_X_train, p_y_train, batch_size=batch_size, nb_epoch=n_epoch,
              validation_data=(p_X_test, p_y_test),
             callbacks=[reduce_lr,earlystop_lr])

    print (u"评估...")
    score, acc = model.evaluate(p_X_test, p_y_test, batch_size=batch_size)
    print ('Test score:', score)
    print ('Test accuracy:', acc)
    return model
```

* 首先用EMbedding将文本数据映射到140 维
* 然后调用SLTM
* 用分词后的数据训练,用验证集提升精度

Task 2： Cnn

数据介绍

1. data -- a 10000x3072 numpy array of uint8s.
    - Each **row** of the array stores a 32x32 colour image.
    - The first 1024 entries contain the **red channel values**,
    - the next 1024 the **green**, and the final 1024 the **blue**.
    - The image is stored in row-major order,so that the first 32 entries of the array are the red channel values of the first row of the image.
2. 处理数据：
    - 图片是三维的：每张图片是高*宽为32*32的矩阵，每个矩阵点上有三个值（r,g,b）（高）
    - 目的：将数据变成3*232*（r,g,b) 的样子，然后再用imshow 画出来
    - 1.将每个图片的每一列取出来，分别为r,g,b
    - 2.将图片reshape（32，32，3）

模型：

- 两个卷积层

```
  # 卷积1
model.add(Convolution2D(
    input_shape=(32, 32, 3),
    filters=32,
    kernel_size=5,
    strides=1,
    padding='same',      # Padding method
    data_format='channels_last',
))
model.add(Activation('relu'))
# 池化层
model.add(MaxPooling2D(
    pool_size=(2,2),# 取样的时候，考虑图片的长宽
    strides=(2,2),# 取图片的间隔
    padding='same',    # Padding method,
    data_format='channels_last',
))
```

- 三个全连接层

```
#Fully connected layer 1 input shape(64*7*7)=(3136),output shape (1024)
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.3))
```

Task 3：GAN

思路：

1. 载入数据

```
def load_minst_data():
    # load the data
    (x_train, y_train), (x_test, y_test) = mnist.load_data()
    # normalize our inputs to be in the range[-1, 1]
    x_train = (x_train.astype(np.float32) - 127.5)/127.5
    # convert x_train with a shape of (60000, 28, 28) to (60000, 784) so we have
    # 784 columns per row
    x_train = x_train.reshape(60000, 784)
    return (x_train, y_train, x_test, y_test)
```

2. 构建生成器： 三层全连接，然后生成图片

```
# 创建生成器`
def get_generator(optimizer):
    generator = Sequential()

    generator.add(Dense(256,input_dim = random_dim,kernel_initializer = initializers.RandomNorma
    generator.add(LeakyReLU(0.2))

    generator.add(Dense(512))
    generator.add(LeakyReLU(0.2))

    generator.add(Dense(1024))
    generator.add(LeakyReLU(0.2))

    generator.add(Dense(784,activation = "tanh"))#

    generator.compile(loss='binary_crossentropy',optimizer = optimizer)
    return generator
```

3. 构建鉴别器

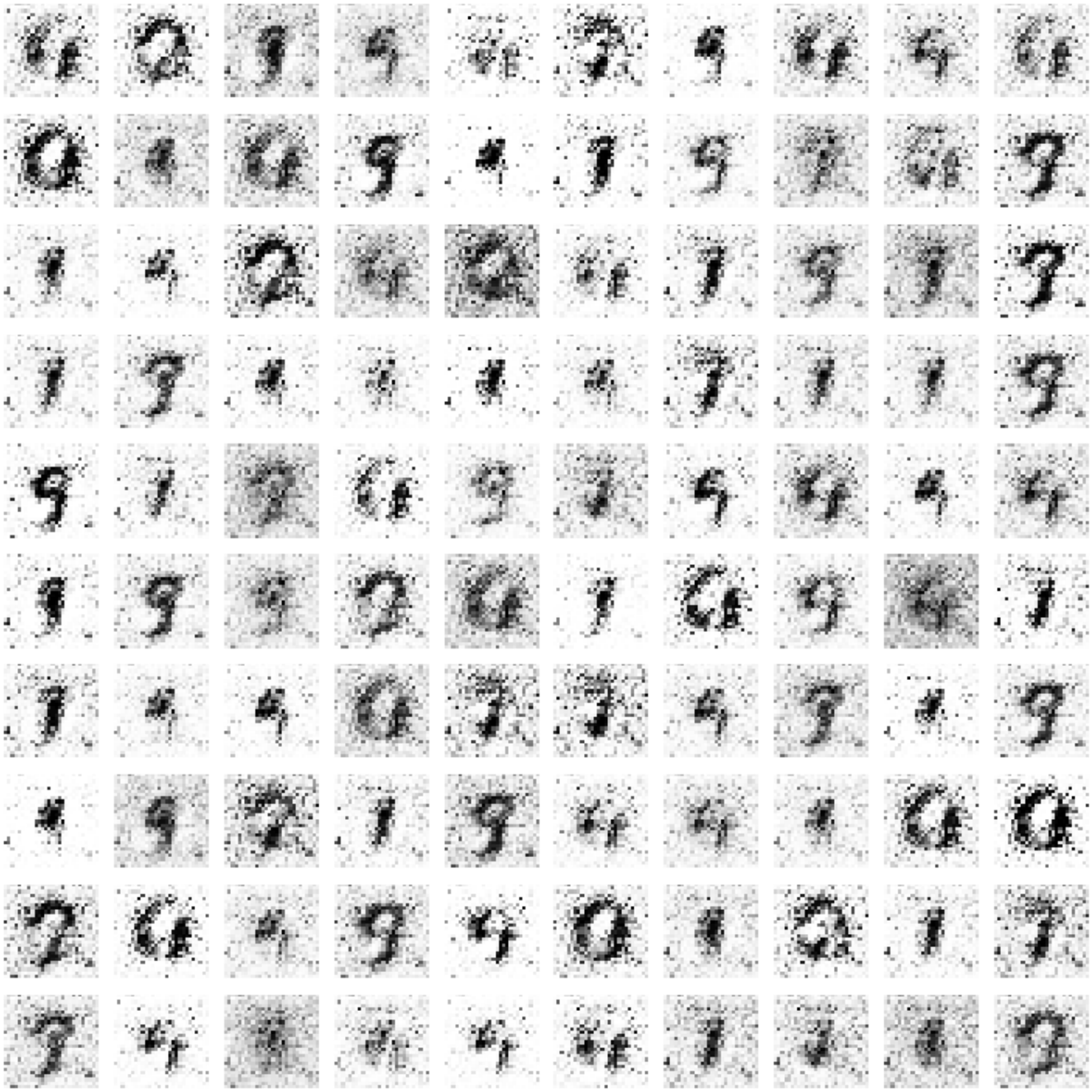
```
    discriminator = Sequential()

    discriminator.add(Dense(1024,input_dim=784,kernel_initializer = initializers.RandomNormal(st
    discriminator.add(LeakyReLU(0.2))
    discriminator.add(Dropout(0.3))#

    discriminator.add(Dense(512))
    discriminator.add(LeakyReLU(0.2))
    discriminator.add(Dropout(0.3))

    discriminator.add(Dense(256))
    discriminator.add(LeakyReLU(0.2))
    discriminator.add(Dropout(0.3))

    discriminator.add(Dense(1,activation='sigmoid'))
    discriminator.compile(loss = "binary_crossentropy",optimizer = optimizer)
    return discriminator
```

4. 然后开始对抗训练

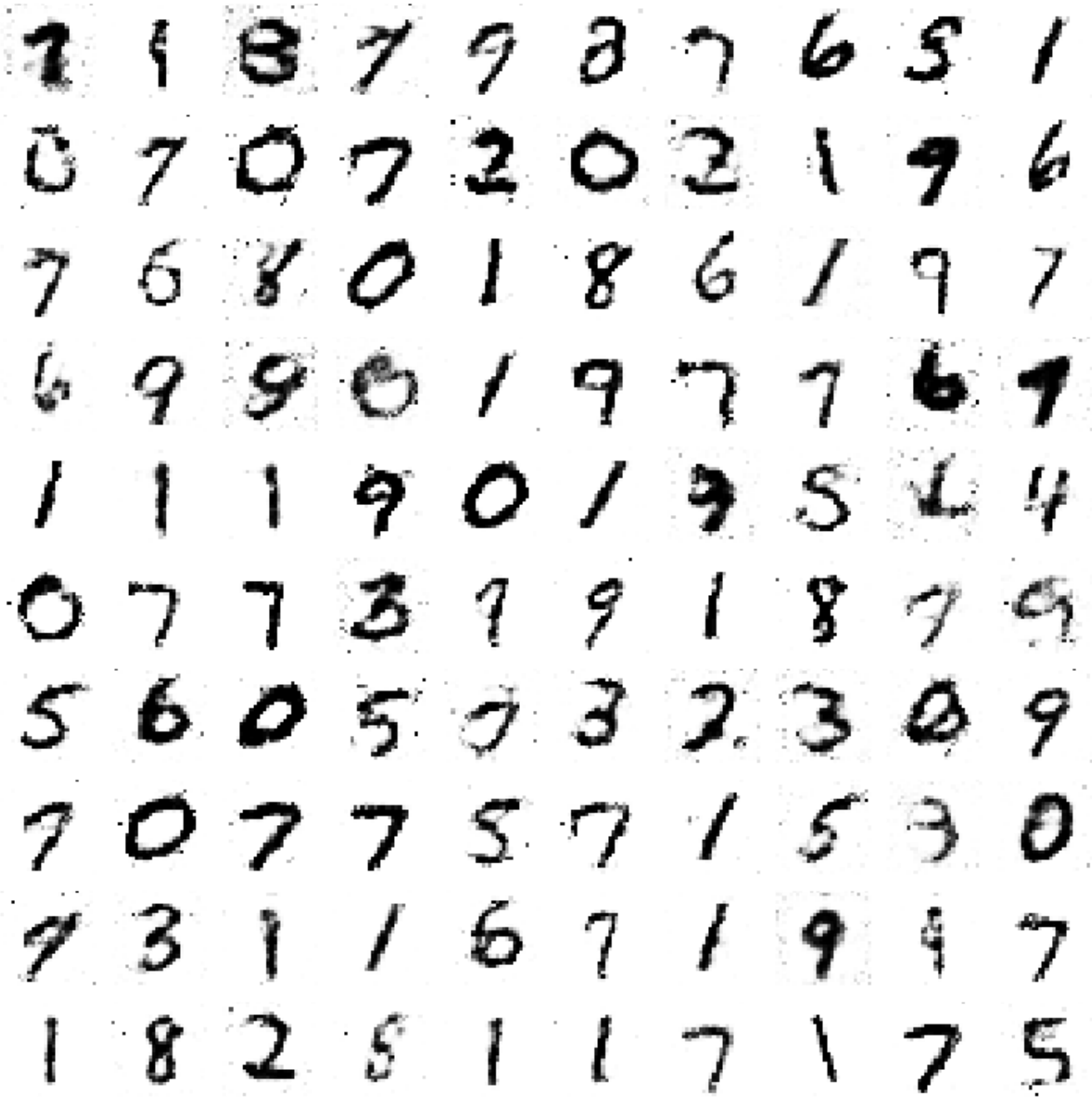
```
def train(epochs=1, batch_size=128):
    # Get the training and testing data
    x_train, y_train, x_test, y_test = load_minst_data()
    # Split the training data into batches of size 128
    batch_count = x_train.shape[0] / batch_size

    # Build our GAN netowrk
    adam = get_optimizer()
    generator = get_generator(adam)
    discriminator = get_discriminator(adam)
    gan = get_gan_network(discriminator, random_dim, generator, adam)

    for e in range(epochs):
        print('-'*15, 'Epoch %d' % e, '-'*15)
        for j in tqdm(range(int(batch_count))):
            # Get a random set of input noise and images
            noise = np.random.normal(0, 1, size=[batch_size, random_dim])
            image_batch = x_train[np.random.randint(0, x_train.shape[0], size=batch_size)]

            # Generate fake MNIST images
            generated_images = generator.predict(noise)
            X = np.concatenate([image_batch, generated_images])

            # Labels for generated and real data
            y_dis = np.zeros(2*batch_size)
            # One-sided label smoothing
            y_dis[:batch_size] = 0.9

            # Train discriminator
            discriminator.trainable = True
            discriminator.train_on_batch(X, y_dis)

            # Train generator
            noise = np.random.normal(0, 1, size=[batch_size, random_dim])
            y_gen = np.ones(batch_size)
            discriminator.trainable = False
            gan.train_on_batch(noise, y_gen)

        if e == 1 or e % 20 == 0:
            plot_generated_images(e, generator)
        pass
```

5. 将生成的图片保存。
   图片结果:

第一张生成图片：



第40张生成图片：



第40张生成图片：

可见最后的生成的图片逐渐清晰，效果较好