循环神经网络 RNN 实验

一、实验目的

学习掌握循环神经网络（RNN）的基本原理及 LSTM 的基本结构;
掌握利用 LSTM 神经元构造循环神经网络进行训练和预测时间序列。

二、实验内容

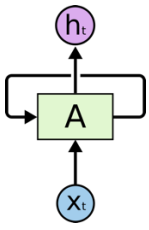
通过 PC 上位机连接服务器，登陆 SimpleAI 平台，利用 python 语言搭建基于 LSTM 的 RNN 模型。利用 RNN 模型对正线曲线或余弦曲线的数值变化进行预测。所用到的数据集由学生自己利用 python 编写代码采样生成。目标是利用正弦曲线或者余弦曲线的前 10 个点的数值预测下一个点的数值。

三、实验环境

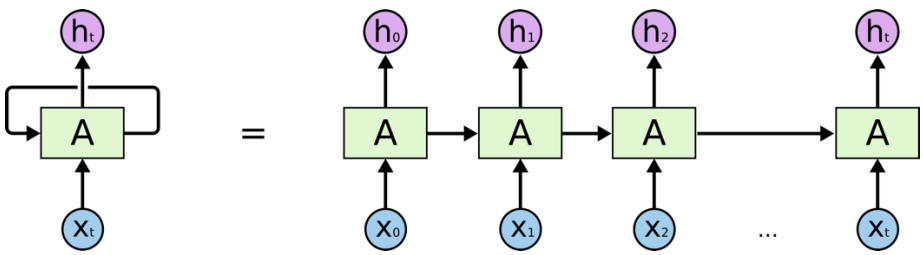
硬件：x86_64 Centos 3.10.0 服务器/GPU 服务器、GPU、PC 上位机
软件：SimpleAI 实验平台、Docker 下 Ubuntu16.04 镜像、python3.5、tensorflow1.7, numpy1.12.1

四、实验原理

RNN 是一种用于处理时序数据的神经网络模型。在传统神经网络中，模型不会关注上一时刻的处理会有什么信息可以用于下一时刻，每一次都只会关注当前时刻的处理。举个例子来说，我们想对一部影片中每一刻出现的事件进行分类，如果我们知道电影前面的事件信息，那么对当前时刻事件的分类就会非常容易。实际上，传统神经网络没有记忆功能，所以它对每一刻出现的事件进行分类时不会用到影片已经出现的信息。基于对这种时间序列关注的需求，循环神经网络应运而生。递归神经网络的结果与传统神经网络有一些不同，它带有一个指向自身的环，用来表示它可以传递当前时刻处理的信息给下一时刻使用，结构如下：

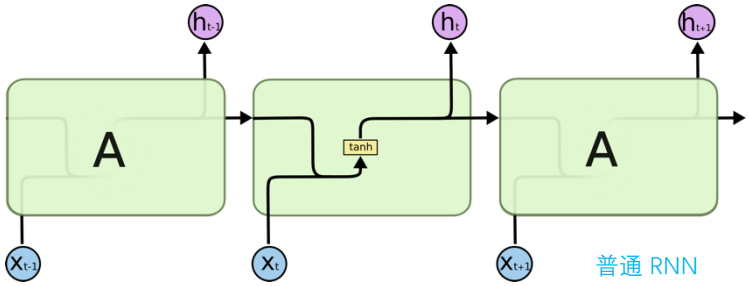


其中，X_t为t时间节点的输入，A为模型处理部分，h_t为t时间节点的输出。为了更好地说明 RNN 的工作原理，可以将上图的循环网络结构展开，得到：

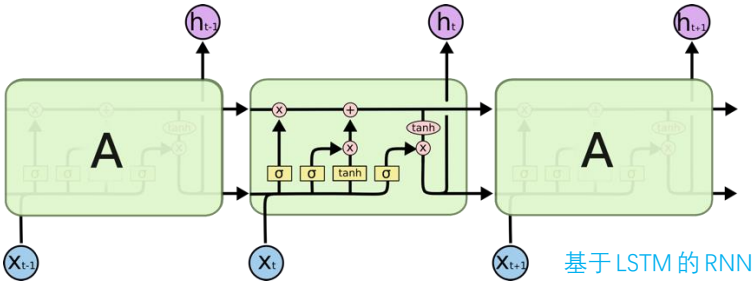


这样的一条链状神经网络代表了一个递归神经网络，可以认为它是对相同神经网络的多重复制，每一时刻的神经网络会传递信息给下一时刻。在本次试验中可以认为X_i为当前i时的正线曲线值，h_i为预测的$i+1$时刻的正弦曲线值。

不过 RNN 也有它自己的问题，即在当预测点与依赖的相关信息距离比较远的时候，就难以学到该相关信息。为了解决这类问题，人们发明了 Long Short Term Memory（LSTM）。其结构与普通的循环神经网络的神经元结构对比如下：

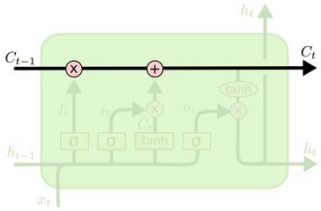


普通 RNN

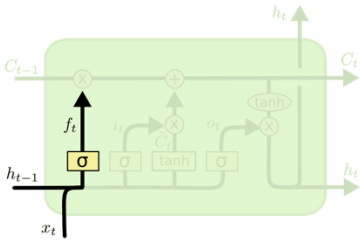


基于 LSTM 的 RNN

理解 LSTMs 的关键就是下面的矩形方框，被称为 memory block（记忆块），主要包含了三个门（forget gate、input gate、output gate）与一个记忆单元（cell）。方框内上方的那条水平线，被称为 cell state（单元状态），它就像一个传送带，可以控制信息传递给下一时刻。



下面来逐步了解 LSTM 的工作原理。LSTM 第一步是用来决定什么信息可以通过 cell state。这个决定由“forget gate”层通过 sigmoid 来控制，它会根据上一时刻的输出h_{t-1}和当前输入x_t来产生一个 0 到 1 的f_t值，来决定是否让上一时刻学到的信息C_{t-1}通过或部分通过。如下：

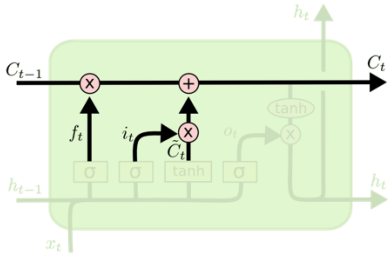


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

举个例子来说就是，我们在之前的句子中学到了很多东西，一些东西对当前来讲是没用的，可以对它进行选择性地过滤。

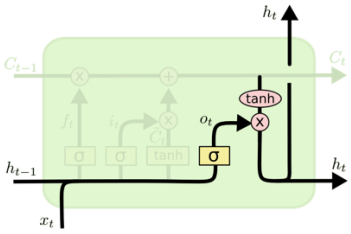
第二步是产生我们需要更新的新信息。这一步包含两部分，第一个是一个“input gate”层通过 sigmoid 来决定哪些值用来更新，第二个是一个 tanh 层用来生成新的候选值$\tilde{C}_t$，它作

为当前层产生的候选值可能会添加到 cell state 中。我们会把这两部分产生的值结合来进行更新。



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

最后一步是决定模型的输出，首先是通过 sigmoid 层来得到一个初始输出，然后使用 tanh 将C_t值缩放到-1 到 1 间，再与 sigmoid 得到的输出逐对相乘，从而得到模型的输出。



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

五、实验步骤

提示：实验步骤中给出的代码不是按照代码顺序给出，请结合代码截图中的行号进行实验。

1、启动 jupyter。

2、新建子目录 log/sin 文件夹和 log/cos 文件夹。这两个文件夹将用于存储训练好的 RNN 模型目录结构如下图所示：

3、新建 python 文件，rnnPredict.py 进行模型编写，引入需要的包

```
1  import os
2  import shutil
3  import tensorflow as tf
4  import numpy as np
5  from sklearn.utils import shuffle
6  import matplotlib.pyplot as plt
```

4、定义主函数，首先进行数据集的创建，首先利用 numpy 工具包进行正弦值或者余弦值序列的生成，并划分训练集和测试集的范围。

```
if __name__ == '__main__':
    tf.logging.set_verbosity(tf.logging.INFO)
    if FLAGS.function == 'sin':
        func = lambda x: np.sin(x)
        log_dir = 'log/sin/'
    else:
        func = lambda x: np.cos(x)
        log_dir = 'log/cos/'
    test_start = FLAGS.train_samples_num * FLAGS.sample_gap
    test_end = (FLAGS.train_samples_num + FLAGS.test_samples_num) * FLAGS.sample_gap
```

5、编写采样函数对正弦曲线值或者余弦曲线值进行采样。每 11 个采样点为一组数据，其中前 10 个为输入数据（x），最后一个为预测数据（label）。

```
def generate_data(seq):
    x = []
    y = []
    for i in range(len(seq) - FLAGS.max_len):
        x.append(seq[i:i + FLAGS.max_len])
        y.append(seq[i + FLAGS.max_len])

    return np.array(x, dtype=np.float32), np.array(y, dtype=np.float32)
```

同时定义辅助函数用于获取数据以及计算 MSE

```
# 获取一个batch_size大小的数据
def get_batches(X, y):
    batch_size = FLAGS.batch_size
    for i in range(0, len(X), batch_size):
        begin_i = i
        end_i = i + batch_size if (i + batch_size) < len(X) else len(X)

        yield X[begin_i:end_i], y[begin_i:end_i]


def average_mse(real, predict):
    predict = np.array(predict)
    mse = np.mean(np.square(real - predict))
    return mse
```

随后在主函数中调用采样函数生成训练集（x,y）对和测试集（x,y）对。

```
train_x, train_y = generate_data(func(np.linspace(0, test_start, FLAGS.train_samples_num, dtype=np.float32)))
tf.logging.info(
    'train dataset has been prepared. train_x shape: {}; train_y shape: {}'.format(train_x.shape, train_y.shape))

test_x, test_y = generate_data(np.sin(np.linspace(test_start, test_end, FLAGS.test_samples_num, dtype=np.float32)))
tf.logging.info(
    'test dataset has been prepared. test_x shape: {}; test_y shape: {}'.format(test_x.shape, test_y.shape))
```

6、编写 RNN 模型，首先对超参数进行定于。在三个参数分别代表超参数变量名称、默认值、参数含义描述。需要特别注意的两个参数是 model_state 和 debugging。model_state 控制模型是训练状态还是预测状态，训练状态将进行反向传播进行优化，预测状态只会前向传

播进行预测。debugging 参数用于控制是否删除当前保存的模型，debugging 为 True 即为重新训练，为 False 则读取原有模型继续训练。

```
FLAGS = tf.app.flags.FLAGS

tf.app.flags.DEFINE_integer('train_samples_num', 1000, 'number of points in the train dataset')
tf.app.flags.DEFINE_float('sample_gap', 0.01, 'the interval of sampling')
tf.app.flags.DEFINE_integer('layer_num', 1, 'number of lstm layer')
tf.app.flags.DEFINE_integer('test_samples_num', 10000, 'number of points in the test dataset')
tf.app.flags.DEFINE_integer('units_num', 128, 'number of hidden units of lstm')
tf.app.flags.DEFINE_integer('epoch', 50, 'epoch of training step')
tf.app.flags.DEFINE_integer('batch_size', 64, 'mini_batch size')
tf.app.flags.DEFINE_integer('max_len', 10, 'we will use ten point to predict the value of 11th')
tf.app.flags.DEFINE_enum('model_state', 'predict', ["train", "predict"], 'model state')
tf.app.flags.DEFINE_boolean('debugging', False, 'delete log or not')
tf.app.flags.DEFINE_float('lr', 0.01, 'learning rate')
tf.app.flags.DEFINE_enum('function', 'cos', ['sin', 'cos'], 'select sin funtion or cosin function')
```

7、创建模型类，名为 RNN，编写初始化函数。定义输入 x 和真实标签 y。

```
class RNN(object):
    def __init__(self):
        self.x = tf.placeholder(dtype=tf.float32, shape=[None, FLAGS.max_len])
        self.y_ = tf.placeholder(dtype=tf.float32, shape=[None])
        self.global_step = tf.train.create_global_step()

        self.input = tf.expand_dims(input=self.x, axis=-1)  # rnn标准的输入维度为 [batch_size, seq_len, dim_size]
```

8、编写 RNN 模型定义函数。

```
    def build_rnn(self):
        with tf.variable_scope('lstm_layer'):
            cells = tf.contrib.rnn.MultiRNNCell(
                [tf.contrib.rnn.BasicLSTMCell(FLAGS.units_num) for _ in range(FLAGS.layer_num)])

            outputs, final_states = tf.nn.dynamic_rnn(cell=cells, inputs=self.input, dtype=np.float32)

            self.outputs = outputs[:, -1]  # 最后一个time step的输出才是我们真正需要的，因为是预测第11个time step的值

        with tf.variable_scope('output_layer'):
            self.predicts = tf.contrib.layers.fully_connected(self.outputs, 1, activation_fn=None)
            self.predicts = tf.reshape(tensor=self.predicts, shape=[-1])
```

9、编写优化操作，用于模型训练。

```
    def build_train_op(self):
        with tf.variable_scope('train_op_layer'):
            self.loss = tf.reduce_mean(tf.square(self.y_ - self.predicts))
            # self.loss = tf.losses.mean_squared_error(self.predicts, self.y_)  # 与上一行效果等效，上一行为详细计算写法，可替换
            tf.summary.scalar(name='loss', tensor=self.loss)  # 配合tensorboard动态监控loss值下降情况

            optimizer = tf.train.AdamOptimizer(learning_rate=FLAGS.lr)
            self.train_op = optimizer.minimize(self.loss, self.global_step)
```

10、编写模型创建函数，该函数为 class 对外界的接口，用于创建模型图对象。

```
    def build_net(self):
        self.build_rnn()
        self.build_train_op()
        self.merged_summary = tf.summary.merge_all()
```

11、至此 RNN 模型定义完毕，回到主函数定义模型训练过程。首先创建 RNN 对象实体并调用 build_net 函数构建模型图。

```
    rnn_model = RNN()
    rnn_model.build_net()
```

12、创建 session 对象。

```
    if FLAGS.debugging:
        if os.path.exists(log_dir):
            print('remove: ' + log_dir)
            shutil.rmtree(log_dir)

    if FLAGS.model_state == 'train':
        if not os.path.exists(log_dir):
            os.makedirs(log_dir)
    saver = tf.train.Saver()
    sv = tf.train.Supervisor(logdir=log_dir,
                             is_chief=True,
                             saver=saver,
                             summary_op=None,
                             save_summaries_secs=None,  # save summaries for tensorboard every 60 secs
                             save_model_secs=60,  # checkpoint every 60 secs
                             global_step=rnn_model.global_step)

    tf.logging.info("Preparing or waiting for session...")
    sess_context_manager = sv.prepare_or_wait_for_session()
    tf.logging.info("Created session.")
    minLoss = 1000
```

13、定义训练过程和预测过程。主要区别是在 sess.run 函数中，预测阶段不需要调用 rnn_model.train_op 这个 operation，即无需优化损失函数。

```
with sess_context_manager as sess:
    if FLAGS.model_state == 'train':
        print('------------Enter train model!------------')
        summary_writer = tf.summary.FileWriter(log_dir)
        for e in range(FLAGS.epoch):
            train_x, train_y = shuffle(train_x, train_y)
            for xs, ys in get_batches(train_x, train_y):
                feed_dict = {
                    rnn_model.x: xs,
                    rnn_model.y_: ys
                }
                _, loss, step, merged_summary = sess.run(
                    [rnn_model.train_op, rnn_model.loss, rnn_model.global_step, rnn_model.merged_summary],
                    feed_dict=feed_dict)
                if step % 10 == 0:
                    tf.logging.info('epoch->{} step->{} loss: {}'.format(e, step, loss))

                    summary_writer.add_summary(merged_summary, step)
                    if loss < minLoss:
                        minLoss = loss
                        saver.save(sess=sess, save_path=log_dir, global_step=step)
    if FLAGS.model_state == 'predict':
        print('------------Enter train model!------------')
        results = []

        for xs, ys in get_batches(test_x, test_y):
            feed_dict = {
                rnn_model.x: xs,
                rnn_model.y_: ys
            }
            predicts = sess.run(rnn_model.predicts, feed_dict=feed_dict)  # 不添加train_op这一优化操作，即不进行反向传播

            results.extend(predicts.tolist())
```

将模式改为训练，运行程序

```
tf.app.flags.DEFINE_enum('model_state', 'train', ["train", "predict"], 'model state')
```

训练过程输出如下图所示，每 10 次优化输出一次当前损失函数值，当损失函数值不再下降时则应停止训练。

```
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Starting standard services.
INFO:tensorflow:Saving checkpoint to path log/cos/model.ckpt
INFO:tensorflow:Starting queue runners.
INFO:tensorflow:Created session.
-----------Enter train model!-----------
INFO:tensorflow:epoch->0 step->0 loss: 0.5745213031768799
INFO:tensorflow:epoch->0 step->10 loss: 0.008007414638996124
INFO:tensorflow:epoch->1 step->20 loss: 0.008686481043696404
INFO:tensorflow:epoch->1 step->30 loss: 0.010120172053575516
INFO:tensorflow:epoch->2 step->40 loss: 0.0016014904249459505
INFO:tensorflow:epoch->3 step->50 loss: 0.0027953218668699265
INFO:tensorflow:epoch->3 step->60 loss: 0.0010719469282776117
INFO:tensorflow:epoch->4 step->70 loss: 0.0011219447478652
INFO:tensorflow:epoch->5 step->80 loss: 0.0011419495567679405
INFO:tensorflow:epoch->5 step->90 loss: 0.0009486249764449894
INFO:tensorflow:epoch->6 step->100 loss: 0.0007039331831037998
INFO:tensorflow:epoch->6 step->110 loss: 0.0008630134398117661
INFO:tensorflow:epoch->7 step->120 loss: 0.000694805639795959
INFO:tensorflow:epoch->8 step->130 loss: 0.0007065759855322254
INFO:tensorflow:epoch->8 step->140 loss: 0.0006421316647902131
INFO:tensorflow:epoch->9 step->150 loss: 0.0005916758673265576
INFO:tensorflow:epoch->10 step->160 loss: 0.0004954956821165979
INFO:tensorflow:epoch->10 step->170 loss: 0.0008869168814271688
INFO:tensorflow:epoch->11 step->180 loss: 0.0006379132391884923
INFO:tensorflow:epoch->11 step->190 loss: 0.0004438956966623664
INFO:tensorflow:epoch->12 step->200 loss: 0.0005897346418350935
INFO:tensorflow:epoch->13 step->210 loss: 0.0005366588593460619
INFO:tensorflow:epoch->13 step->220 loss: 0.0005266557564027607
INFO:tensorflow:epoch->14 step->230 loss: 0.0003990195752501488
INFO:tensorflow:epoch->15 step->240 loss: 0.0004630267503671348
INFO:tensorflow:epoch->15 step->250 loss: 0.0006055147969163954
INFO:tensorflow:epoch->16 step->260 loss: 0.0004742182209156453 6
INFO:tensorflow:epoch->16 step->270 loss: 0.0002932628267444467
INFO:tensorflow:epoch->17 step->280 loss: 0.0004473477019928396
INFO:tensorflow:epoch->18 step->290 loss: 0.0004539597139228135 3
INFO:tensorflow:epoch->18 step->300 loss: 0.0003993560676462948 3
INFO:tensorflow:epoch->19 step->310 loss: 0.0004394424613565206 5
INFO:tensorflow:epoch->20 step->320 loss: 0.000855114427395165
INFO:tensorflow:epoch->20 step->330 loss: 0.0003508035151753574 6
```

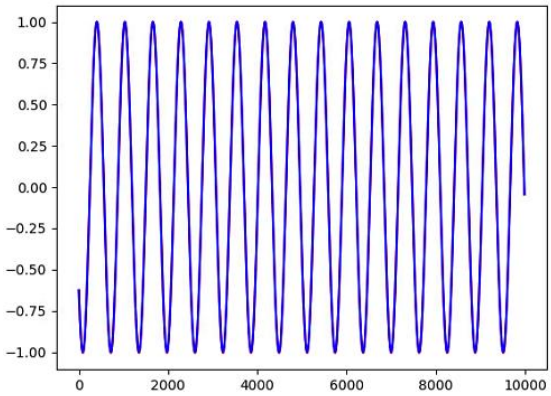
再将参数改为预测运行程序

```
18    tf.app.flags.DEFINE_enum('model_state', 'predict', ["train", "predict"], 'model state')
```

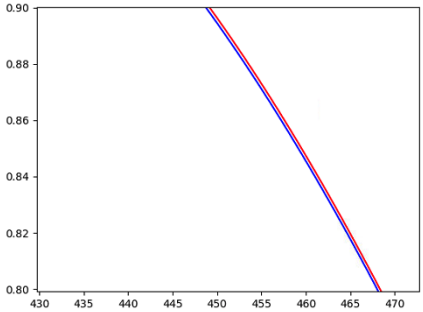
预测阶段输出如下图所示将显示预测结果的平均 MSE 值。

```
INFO:tensorflow:Restoring parameters from log/cos/-530
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Starting standard services.
INFO:tensorflow:Saving checkpoint to path log/cos/model.ckpt
INFO:tensorflow:Starting queue runners.
INFO:tensorflow:Created session.
------------Enter train model!------------
INFO:tensorflow:average of mse: 8.030989345558939e-06
```

最后也可以将预测结果可视化。查看预测效果。示范预测结果如下图所示，蓝色线为真实值，红色线为预测值。



为了真切的看到差别，将曲线图放大，如下所示，可以看到虽然预测的很接近，但还是存在一定的偏差。



六、扩展实验

以上是利用单层 RNN 使用 10 个连续的正弦函数值或者余弦函数值来预测第 11 个值的实验。在扩展实验中，（1）使用 GRU 网络与 LSTM 网络进行比较。（2）使用多层 RNN 来进行进行预测，将预测的 MSE 值结果与单层模型的 MSE 值进行对比分析。