姓名：向建宇

学号：1801210687

**利用循环神经网络预测Cos函数。**

一、循环神经网络RNN

#### 它的网络结构如下：

#### https://img-blog.csdn.net/20171129184524844?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvcXFfMzk0MjI2NDI=/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast

#### 其中每个圆圈可以看作是一个单元，而且每个单元做的事情也是一样的，因此可以折叠呈左半图的样子。用一句话解释RNN，就是一个单元结构重复使用。

#### 

#### RNN是一个序列到序列的模型，假设xt−1,xt,xt+1xt−1,xt,xt+1是一个输入：“我是中国“，那么ot−1,otot−1,ot就应该对应”是”，”中国”这两个，预测下一个词最有可能是什么？就是ot+1ot+1应该是”人”的概率比较大。

#### 

#### 因此，我们可以做这样的定义：Xt:表示t时刻的输入，ot:表示t时刻的输出，St:表示t时刻的记忆Xt:表示t时刻的输入，ot:表示t时刻的输出，St:表示t时刻的记忆。因为我们当前时刻的输出是由记忆和当前时刻的输入决定的，就像你现在大四，你的知识是由大四学到的知识（当前输入）和大三以及大三以前学到的东西的（记忆）的结合，RNN在这点上也类似，神经网络最擅长做的就是通过一系列参数把很多内容整合到一起，然后学习这个参数，因此就定义了RNN的基础：St=f(U∗Xt+W∗St−1)St=f(U∗Xt+W∗St−1)

#### 二、LSTM

#### Long Short-Term Memory Neural Network—— 一般就叫做 LSTM ，是一种 RNN 特殊的类型，可以学习长期依赖信息。LSTM 由Hochreiter & Schmidhuber (1997)提出，并在近期被Alex Graves进行了改良和推广。在很多问题，LSTM 都取得相当巨大的成功，并得到了广泛的使用。LSTM 通过刻意的设计来避免长期依赖问题。记住长期的信息在实践中是 LSTM 的默认行为，而非需要付出很大代价才能获得的能力！

#### 所有 RNN 都具有一种重复神经网络模块的链式的形式。在标准的 RNN 中，这个重复的模块只有一个非常简单的结构，例如一个 tanh 层。

#### https://img-blog.csdn.net/20170221171225361?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvdTAxMDA4OTQ0NA==/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast

#### LSTM 同样是这样的结构，但是重复的模块拥有一个不同的结构。不同于 单一神经网络层，这里是有四个，以一种非常特殊的方式进行交互

#### 

#### 我们先来熟悉一下图中使用的各种元素的图标：在上面的图例中，每一条黑线传输着一整个向量，从一个节点的输出到其他节点的输入。粉色的圈代表 pointwise 的操作，诸如向量的和，而黄色的矩阵就是学习到的神经网络层。合在一起的线表示向量的连接，分开的线表示内容被复制，然后分发到不同的位置。

#### 

#### LSTM 的关键就是细胞状态，水平线在图上方贯穿运行。

#### 细胞状态类似于传送带。直接在整个链上运行，只有一些少量的线性交互。信息在上面流传保持不变会很容易。

#### 

#### LSTM 有通过精心设计的称作为“门”的结构来去除或者增加信息到细胞状态的能力。门是一种让信息选择式通过的方法。他们包含一个 sigmoid 神经网络层和一个 pointwise 乘法操作。

#### https://img-blog.csdn.net/20170221172159708?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvdTAxMDA4OTQ0NA==/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast

#### Sigmoid 层输出 0 到 1 之间的数值，描述每个部分有多少量可以通过。0 代表“不许任何量通过”，1 就指“允许任意量通过”！

#### LSTM 拥有三个门（输入门，遗忘门，输出门），来保护和控制细胞状态。

#### 三、预测余旋函数

#### 1. 定义RNN的参数。

|  |
| --- |
| HIDDEN\_SIZE = 30 # LSTM中隐藏节点的个数。NUM\_LAYERS = 2 # LSTM的层数。TIMESTEPS = 10 # 循环神经网络的训练序列长度。TRAINING\_STEPS = 10000 # 训练轮数。BATCH\_SIZE = 32 # batch大小。TRAINING\_EXAMPLES = 10000 # 训练数据个数。TESTING\_EXAMPLES = 1000 # 测试数据个数。SAMPLE\_GAP = 0.01 # 采样间隔。 |

#### 2. 产生正弦数据。

|  |
| --- |
| def generate\_data(seq):X = []y = []# 序列的第i项和后面的TIMESTEPS-1项合在一起作为输入；第i + TIMESTEPS项作为输# 出。即用sin函数前面的TIMESTEPS个点的信息，预测第i + TIMESTEPS个点的函数值。for i in range(len(seq) - TIMESTEPS):X.append([seq[i: i + TIMESTEPS]])y.append([seq[i + TIMESTEPS]])return np.array(X, dtype=np.float32), np.array(y, dtype=np.float32)# 用正弦函数生成训练和测试数据集合。test\_start = (TRAINING\_EXAMPLES + TIMESTEPS) \* SAMPLE\_GAPtest\_end = test\_start + (TESTING\_EXAMPLES + TIMESTEPS) \* SAMPLE\_GAPtrain\_X, train\_y = generate\_data(np.sin(np.linspace(0, test\_start, TRAINING\_EXAMPLES + TIMESTEPS, dtype=np.float32)))test\_X, test\_y = generate\_data(np.sin(np.linspace(test\_start, test\_end, TESTING\_EXAMPLES + TIMESTEPS, dtype=np.float32))) |

#### 3. 定义网络结构和优化步骤。

|  |
| --- |
| def lstm\_model(X, y, is\_training):# 使用多层的LSTM结构。cell = tf.nn.rnn\_cell.MultiRNNCell([tf.nn.rnn\_cell.BasicLSTMCell(HIDDEN\_SIZE)for \_ in range(NUM\_LAYERS)])# 使用TensorFlow接口将多层的LSTM结构连接成RNN网络并计算其前向传播结果。outputs, \_ = tf.nn.dynamic\_rnn(cell, X, dtype=tf.float32)output = outputs[:, -1, :]# 对LSTM网络的输出再做加一层全链接层并计算损失。注意这里默认的损失为平均# 平方差损失函数。predictions = tf.contrib.layers.fully\_connected(output, 1, activation\_fn=None)# 只在训练时计算损失函数和优化步骤。测试时直接返回预测结果。if not is\_training:return predictions, None, None# 计算损失函数。loss = tf.losses.mean\_squared\_error(labels=y, predictions=predictions)# 创建模型优化器并得到优化步骤。train\_op = tf.contrib.layers.optimize\_loss(loss, tf.train.get\_global\_step(),optimizer="Adagrad", learning\_rate=0.1)return predictions, loss, train\_op |

#### 4. 定义测试方法。

|  |
| --- |
| def run\_eval(sess, test\_X, test\_y):# 将测试数据以数据集的方式提供给计算图。ds = tf.data.Dataset.from\_tensor\_slices((test\_X, test\_y))ds = ds.batch(1)X, y = ds.make\_one\_shot\_iterator().get\_next()# 调用模型得到计算结果。这里不需要输入真实的y值。with tf.variable\_scope("model", reuse=True):prediction, \_, \_ = lstm\_model(X, [0.0], False)# 将预测结果存入一个数组。predictions = []labels = []for i in range(TESTING\_EXAMPLES):p, l = sess.run([prediction, y])predictions.append(p)labels.append(l)# 计算rmse作为评价指标。predictions = np.array(predictions).squeeze()labels = np.array(labels).squeeze()rmse = np.sqrt(((predictions - labels) \*\* 2).mean(axis=0))print("Root Mean Square Error is: %f" % rmse)#对预测的sin函数曲线进行绘图。plt.figure()plt.plot(predictions, label='predictions')plt.plot(labels, label='real\_sin')plt.legend()plt.show() |

#### 5. 执行训练和测试。

|  |
| --- |
| # 将训练数据以数据集的方式提供给计算图。ds = tf.data.Dataset.from\_tensor\_slices((train\_X, train\_y))ds = ds.repeat().shuffle(1000).batch(BATCH\_SIZE)X, y = ds.make\_one\_shot\_iterator().get\_next()# 定义模型，得到预测结果、损失函数，和训练操作。with tf.variable\_scope("model"):\_, loss, train\_op = lstm\_model(X, y, True)with tf.Session() as sess:sess.run(tf.global\_variables\_initializer())# 测试在训练之前的模型效果。print "Evaluate model before training."run\_eval(sess, test\_X, test\_y)# 训练模型。for i in range(TRAINING\_STEPS):\_, l = sess.run([train\_op, loss])if i % 1000 == 0:print("train step: " + str(i) + ", loss: " + str(l))# 使用训练好的模型对测试数据进行预测。print "Evaluate model after training."run\_eval(sess, test\_X, test\_y) |

#### 四、结果截图

#### 

#### 预测基本正确