**实验三 神经语言模型**

1. 实验目的
2. 掌握循环神经网络结构，包括LSTM等；
3. 掌握使用Tensorflow构建循环神经网络模型的方法；
4. 了解Tensorflow在GPU上的使用方法。
5. 实验要求
6. 建立一个规范的LSTM网络；
7. 在PTB（Penn Treebank）语料库上进行语言模型的训练和评估。
8. 实验原理
9. 语言模型：

语言模型是一个为某一段词序列分配概率的模型。它对多种自然语言处理任务都有帮助。例如，在机器翻译任务中，需要由语言模型为系统输出打分，以提高输出目标语言输出的流畅性。在语音识别任务中，语言模型与声学模型一起预测下一个词。语言模型用来计算一个具有N个词的词序列概率，即：

******

上式可知，它也可以分解成对给定前缀（一般叫做上下文）的下一词出现概率的乘积。神经语言模型就是用于估计每一个词出现的条件概率的。

1. 循环神经网络：

循环神经网络对序列数据的处理有先天优势，它的结构使网络可以接受变长输入，当网络输入窗口被移位时，不需要重复计算。其网络结构如图1所示。

我们关注某一时刻的损失，它就等于此时刻之前所有时间步的损失之和。例如，对于时间时刻的损失，它的损失为

这种在时间步上展开进行反向传播的算法，称为基于时间的反向传播（Back-Propagation Through Time，BPTT）。循环神经网络的训练就是通过BPTT算法进行的。



图1 循环神经网络（RNN）结构

下面以图1为例，我们将通过BPTT算法来计算RNN的参数梯度。对于每一个节点N，我们需要基于N后面的节点的梯度，递归地计算梯度。我们从最后一个节点的损失开始递归：

对于时间步t输出的梯度的第i个元素为：

从序列的最后时间步开始反向计算梯度。对于最后时间步，只由得到，则梯度为：

根据这个梯度我们就可以依次计算时间步到时间步1的隐层节点的梯度。由于是同时有和两个后续节点。因此，对于每个时间步t的隐层节点的梯度为：

在得到了隐层节点的梯度后，我们就可以计算对于参数的梯度：

其中，**、、**表示时刻t时，**W**、**U**、**b**的副本，则表示时刻t的时的梯度贡献，对于**U**和**b**是类似的。

长短期记忆网络（LSTM）是一种具有门结构的特殊循环神经网络。它是为了应对长期依赖的挑战而提出的。LSTM网络结构如图2所示。



图2 LSTM网络结构

LSTM引入了“门”机制对细胞状态信息进行添加或删除，由此实现长程记忆。“门”机制由一个Sigmoid激活函数层和一个向量点乘操作组成，Sigmoid层的输出控制了信息传递的比例。每个LSTM基本单元包含遗忘门、输入门和输出门三个门结构。

1）遗忘门

LSTM通过遗忘门（forget gate）实现对细胞状态信息遗忘程度的控制，输出当前状态的遗忘权重，取决于和。

2）输入门

LSTM通过输入门（input gate）实现对细胞状态输入接收程度的控制，输出当前输入信息的接受权重，取决于和。

3）输出门

LSTM通过输出门（output gate）实现对细胞状态输出认可程度的控制，输出当前输出信息的认可权重，取决于和。

4）状态更新

1. 实验所用工具及数据集
2. 工具

Anaconda、TensorFlow（Tensorflow安装教程参考：Tensorflow官网、Tensorflow中文社区、<https://github.com/tensorflow/tensorflow>）

1. 数据集

Penn Treebank（PTB）语料库（下载地址：http://www.fit.vutbr.cz/~imikolov/rnnlm/simple-examples.tgz）

1. 实验步骤与方法
2. 下载语料库

数据来源于 Tomas Mikolov 网站上的 PTB 数据集。该数据集已经预先处理过并且包含了全部的 10000 个不同的词语，其中包括语句结束标记符，以及标记稀有词语的特殊符号 (<unk>) 。

1. 加载及准备数据

#从文件读入的语料库数据上迭代生成训练、验证、训练数据集

#其中raw\_data是需要将词转换成词序号，方便后续处理

def ptb\_producer(raw\_data, batch\_size, num\_steps, name=None):

with tf.name\_scope(name, "PTBProducer", [raw\_data, batch\_size, num\_steps]):

raw\_data = tf.convert\_to\_tensor(raw\_data, name="raw\_data", dtype=tf.int32)

#将数据进行切分，去掉多余数据

data\_len = tf.size(raw\_data)

batch\_len = data\_len // batch\_size

data = tf.reshape(raw\_data[0 : batch\_size \* batch\_len],

[batch\_size, batch\_len])

epoch\_size = (batch\_len - 1) // num\_steps

assertion = tf.assert\_positive(

epoch\_size,

message="epoch\_size == 0, decrease batch\_size or num\_steps")

with tf.control\_dependencies([assertion]):

epoch\_size = tf.identity(epoch\_size, name="epoch\_size")

#从数据中获得批数据集

i = tf.train.range\_input\_producer(epoch\_size, shuffle=False).dequeue()

x = tf.strided\_slice(data, [0, i \* num\_steps],

[batch\_size, (i + 1) \* num\_steps])

x.set\_shape([batch\_size, num\_steps])

y = tf.strided\_slice(data, [0, i \* num\_steps + 1],

[batch\_size, (i + 1) \* num\_steps + 1])

y.set\_shape([batch\_size, num\_steps])

return x, y

1. 构建计算图

**（1）构建LSTM**

with tf.device("/cpu:0"):

embedding = tf.get\_variable(

"embedding", [vocab\_size, size], dtype=data\_type())

inputs = tf.nn.embedding\_lookup(embedding, input\_.input\_data)

#输入数据接dropout层

if is\_training and config.keep\_prob < 1:

inputs = tf.nn.dropout(inputs, config.keep\_prob)

#为每个LSTM单元加上dropout层

def make\_cell():

cell = self.\_get\_lstm\_cell(config, is\_training)

if is\_training and config.keep\_prob < 1:

cell = tf.contrib.rnn.DropoutWrapper(

cell, output\_keep\_prob=config.keep\_prob)

return cell

cell = tf.contrib.rnn.MultiRNNCell(

[make\_cell() for \_ in range(config.num\_layers)], state\_is\_tuple=True)

#初始单元状态为零状态

self.\_initial\_state = cell.zero\_state(config.batch\_size, data\_type())

state = self.\_initial\_state

#构建LSTM网络

outputs = []

with tf.variable\_scope("RNN"):

for time\_step in range(self.num\_steps):

#设置参数共享（重用）

if time\_step > 0: tf.get\_variable\_scope().reuse\_variables()

(cell\_output, state) = cell(inputs[:, time\_step, :], state)

outputs.append(cell\_output)

output = tf.reshape(tf.concat(outputs, 1), [-1, config.hidden\_size])

#softmax分类层

softmax\_w = tf.get\_variable(

"softmax\_w", [size, vocab\_size], dtype=data\_type())

softmax\_b = tf.get\_variable("softmax\_b", [vocab\_size], dtype=data\_type())

logits = tf.nn.xw\_plus\_b(output, softmax\_w, softmax\_b)

# Reshape logits to be a 3-D tensor for sequence loss

logits = tf.reshape(logits, [self.batch\_size, self.num\_steps, vocab\_size])

**（2）训练与评估：**

# 用在批上平均的序列损失求得损失

loss = tf.contrib.seq2seq.sequence\_loss(

logits,

input\_.targets,

tf.ones([self.batch\_size, self.num\_steps], dtype=data\_type()),

average\_across\_timesteps=False,

average\_across\_batch=True)

# 每一时间步损失求和

self.\_cost = tf.reduce\_sum(loss)

#求得梯度

self.\_lr = tf.Variable(0.0, trainable=False)

tvars = tf.trainable\_variables()

#使用全局梯度的范数进行梯度截断

grads, \_ = tf.clip\_by\_global\_norm(tf.gradients(self.\_cost, tvars),

config.max\_grad\_norm)

optimizer = tf.train.GradientDescentOptimizer(self.\_lr)

self.\_train\_op = optimizer.apply\_gradients(

zip(grads, tvars),

global\_step=tf.train.get\_or\_create\_global\_step())

#更新学习率

self.\_new\_lr = tf.placeholder(

tf.float32, shape=[], name="new\_learning\_rate")

self.\_lr\_update = tf.assign(self.\_lr, self.\_new\_lr)

1. 创建会话，进行模型的训练与评估

#在批数据集上运行模型

def run\_epoch(session, model, eval\_op=None, verbose=False):

start\_time = time.time()

costs = 0.0

iters = 0

state = session.run(model.initial\_state)

fetches = {

"cost": model.cost,

"final\_state": model.final\_state,

}

if eval\_op is not None:

fetches["eval\_op"] = eval\_op

for step in range(model.input.epoch\_size):

feed\_dict = {}

for i, (c, h) in enumerate(model.initial\_state):

feed\_dict[c] = state[i].c

feed\_dict[h] = state[i].h

vals = session.run(fetches, feed\_dict)

cost = vals["cost"]

state = vals["final\_state"]

costs += cost

iters += model.input.num\_steps

if verbose and step % (model.input.epoch\_size // 10) == 10:

print("%.3f perplexity: %.3f speed: %.0f wps" %

(step \* 1.0 / model.input.epoch\_size, np.exp(costs / iters),

iters \* model.input.batch\_size \* max(1, FLAGS.num\_gpus) /

(time.time() - start\_time)))

return np.exp(costs / iters)、

with tf.Graph().as\_default():

sv = tf.train.Supervisor(logdir=FLAGS.save\_path)

config\_proto = tf.ConfigProto(allow\_soft\_placement=soft\_placement)

with sv.managed\_session(config=config\_proto) as session:

for i in range(config.max\_max\_epoch):

lr\_decay = config.lr\_decay \*\* max(i + 1 - config.max\_epoch, 0.0)

m.assign\_lr(session, config.learning\_rate \* lr\_decay)

print("Epoch: %d Learning rate: %.3f" % (i + 1, session.run(m.lr)))

train\_perplexity = run\_epoch(session, m, eval\_op=m.train\_op,

verbose=True)

print("Epoch: %d Train Perplexity: %.3f" % (i + 1, train\_perplexity))

valid\_perplexity = run\_epoch(session, mvalid)

print("Epoch: %d Valid Perplexity: %.3f" % (i + 1, valid\_perplexity))

test\_perplexity = run\_epoch(session, mtest)

print("Test Perplexity: %.3f" % test\_perplexity)

if FLAGS.save\_path:

print("Saving model to %s." % FLAGS.save\_path)

sv.saver.save(session, FLAGS.save\_path, global\_step=sv.global\_step)