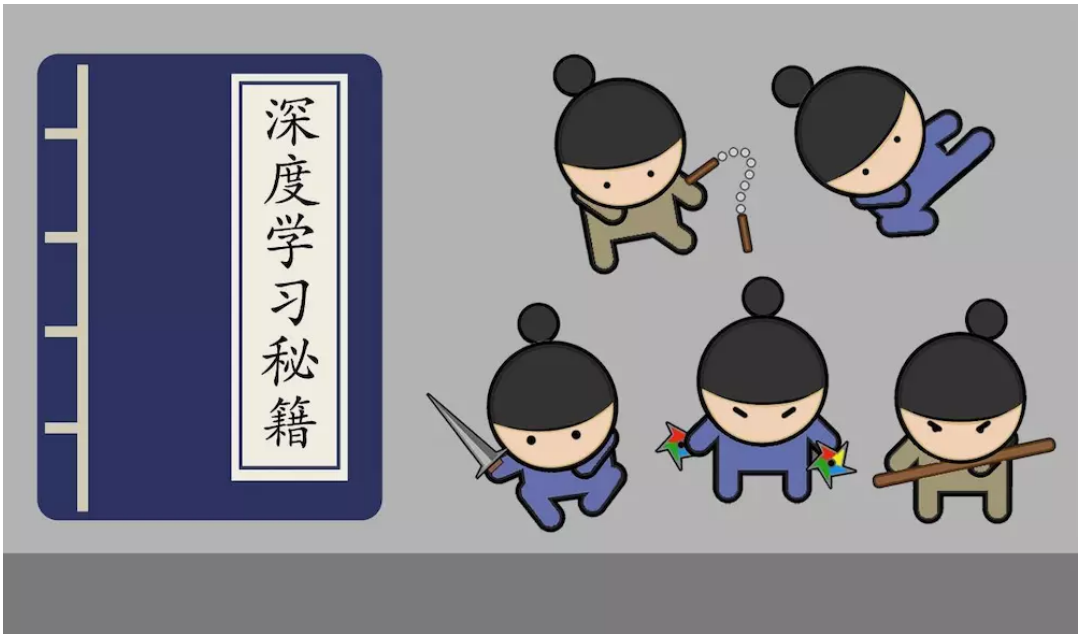
NLP第9课：一网打尽神经序列模型之RNN 及其变种 LSTM、GRU

米饭超人 (/u/ba83fba00eef) ＋关注
0.7 2018.12.02 21:20 字数 2737 阅读 304 评论 0 喜欢 5
(/u/ba83fba00eef)

首先，我们来思考下，当人工神经网络从浅层发展到深层；从全连接到卷积神经网络。在此过程中，人类在图片分类、语音识别等方面都取得了非常好的结果，那么我们为什么还需要循环神经网络呢？



enter image description here

因为，上面提到的这些网络结构的层与层之间是全连接或部分连接的，但在每层之间的节点是无连接的，这样的网络结构并不能很好的处理序列数据。

序列数据的处理，我们从语言模型 N-gram 模型说起，然后着重谈谈 RNN，并通过 RNN 的变种 LSTM 和 GRU 来实战文本分类。

语言模型 N-gram 模型

通过前面的课程，我们了解到一般自然语言处理的传统方法是将句子处理为一个词袋模型（Bag-of-Words，BoW），而不考虑每个词的顺序，比如用朴素贝叶斯算法进行垃圾邮件识别或者文本分类。在中文里有时候这种方式没有问题，因为有些句子即使把词的顺序打乱，还是可以看懂这句话在说什么，比如：

> T：研究表明，汉字的顺序并不一定能影响阅读，比如当你看完这句话后。
>
> F：研表究明，汉字的序顺并不定一能影阅响读，比如当你看完这句话后。

但有时候不行，词的顺序打乱，句子意思就变得让人不可思议了，例如：

> T：我喜欢吃烧烤。
>
> F：烧烤喜欢吃我。

那么，有没有模型是考虑句子中词与词之间的顺序的呢？有，语言模型中的 N-gram 就是一种。

N-gram 模型是一种语言模型（Language Model，LM），是一个基于概率的判别模型，它的输入是一句话（词的顺序序列），输出是这句话的概率，即这些词的联合概率（Joint Probability）。
使用 N-gram 语言模型思想，一般是需要知道当前词以及前面的词，因为一个句子中每个词的出现并不是独立的。比如，如果第一个词是“空气”，接下来的词是“很”，那么下一个词很大概率会是“新鲜”。类似于我们人的联想，N-gram 模型知道的信息越多，得到的结果也越准确。

在前面课程中讲解的文本分类中，我们曾用到基于 sklearn 的词袋模型，尝试加入抽取 `2-gram` 和 `3-gram` 的统计特征，把词库的量放大，获得更强的特征。

通过 ngram_range 参数来控制，代码如下：

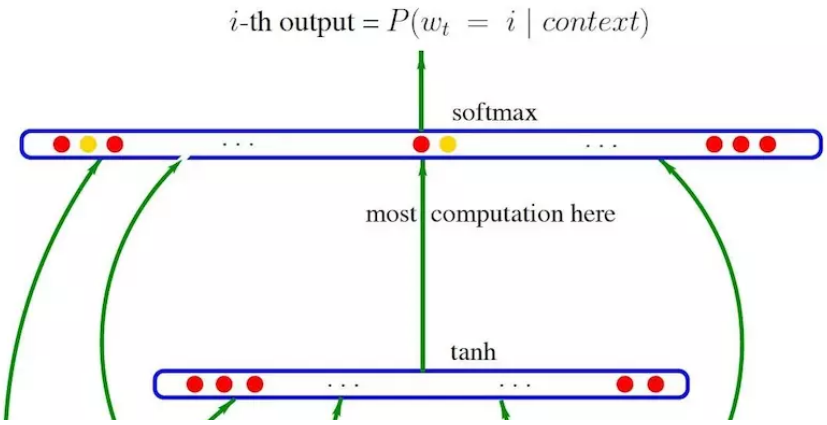
因此，N-gram 模型，在自然语言处理中主要应用在如词性标注、垃圾短信分类、分词器、机器翻译和语音识别、语音识别等领域。

然而 N-gram 模型并不是完美的，它存在如下优缺点：

- 优点：包含了前 N-1 个词所能提供的全部信息，这些词对于当前词的出现概率具有很强的约束力；
- 缺点：需要很大规模的训练文本来确定模型的参数，当 N 很大时，模型的参数空间过大。所以常见的 N 值一般为1，2，3等。还有因数据稀疏而导致的数据平滑问题，解决方法主要是拉普拉斯平滑和内插与回溯。

所以，根据 N-gram 的优缺点，它的进化版 NNLM（Neural Network based Language Model）诞生了。

NNLM 由 Bengio 在2003年提出，它是一个很简单的模型，由四层组成，输入层、嵌入层、隐层和输出层，模型结构如下图（来自百度图片）：



enter image description here

NNLM 接收的输入是长度为 N 的词序列，输出是下一个词的类别。首先，输入是词序列的 index 序列，例如词“我”在字典（大小为|V|）中的 index 是10，词“是”的 index 是23，“小明”的 index 是65，则句子“我是小明”的 index 序列就是 10、 23、65。嵌入层（Embedding）是一个大小为 `|V|×K` 的矩阵，从中取出第10、23、65行向量拼成 3×K 的矩阵就是 Embedding 层的输出了。隐层接受拼接后的 Embedding 层输出作为输入，以 tanh 为激活函数，最后送入带 softmax 的输出层，输出概率。

NNLM 最大的缺点就是参数多，训练慢，要求输入定长 N 这一点很不灵活，同时不能利用完整的历史信息。

因此，针对 NNLM 存在的问题，Mikolov 在2010年提出了 RNNLM，有兴趣可以阅读相关论文 (http://www.fit.vutbr.cz/~imikolov/rnnlm/thesis.pdf)，其结构实际上是用 RNN 代替 NNLM 里的隐层，这样做的好处，包括减少模型参数、提高训练速度、接受任意长度输入、利用完整的历史信息。同时，RNN 的引入意味着可以使用 RNN 的其他变体，像 LSTM、BLSTM、GRU 等等，从而在序列建模上进行更多更丰富的优化。

以上，从词袋模型说起，引出语言模型 N-gram 以及其优化模型 NNLM 和 RNNLM，后续内容从 RNN 说起，来看看其变种 LSTM 和 GRU 模型如何处理类似序列数据。

RNN 以及变种 LSTM 和 GRU 原理

RNN 为序列数据而生

RNN 称为循环神经网路，因为这种网络有“记忆性”，主要应用在自然语言处理（NLP）和语音领域。RNN 具体的表现形式为网络会对前面的信息进行记忆并应用于当前输出的计算中，即隐藏层之间的节点不再无连接而是有连接的，并且隐藏层的输入不仅包括输入层的输出还包括上一时刻隐藏层的输出。

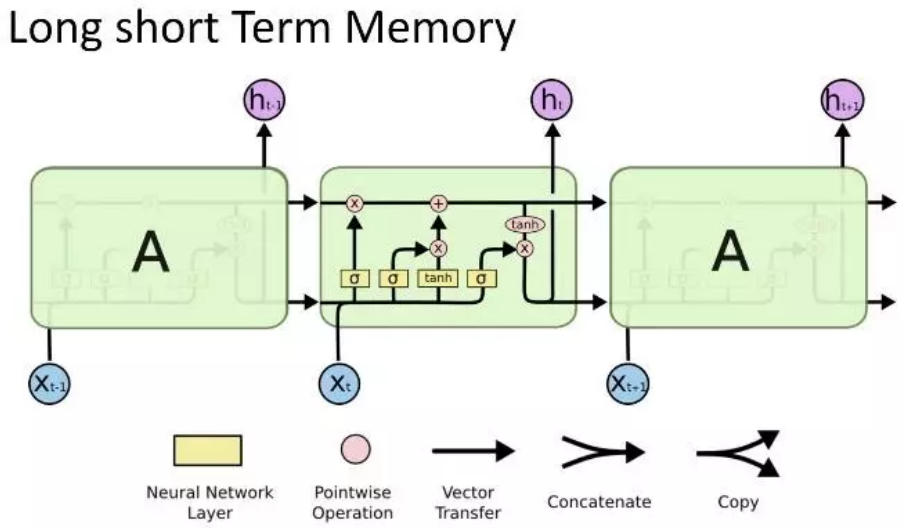
理论上，RNN 能够对任何长度的序列数据进行处理，但由于该网络结构存在“梯度消失”问题，所以在实际应用中，解决梯度消失的方法有：梯度裁剪（Clipping Gradient）和 LSTM（ `Long Short-Term Memory` ）。

下图是一个简单的 RNN 经典结构：

enter image description here

LSTM 结构

LSTM 在1997年由“Hochreiter & Schmidhuber”提出，目前已经成为 RNN 中的标准形式，用来解决上面提到的 RNN 模型存在“长期依赖”的问题。



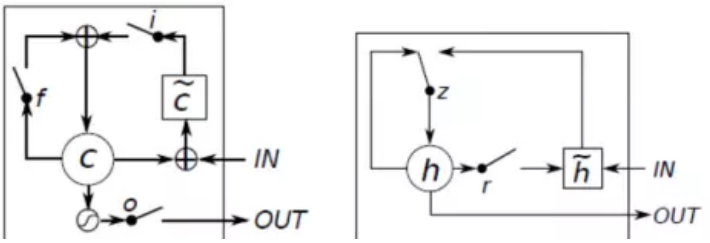
enter image description here

LSTM 通过三个“门”结构来控制不同时刻的状态和输出。所谓的“门”结构就是使用了 Sigmoid 激活函数的全连接神经网络和一个按位做乘法的操作，Sigmoid 激活函数会输出一个0~1之间的数值，这个数值代表当前有多少信息能通过“门”，0表示任何信息都无法通过，1表示全部信息都可以通过。其中，“遗忘门”和“输入门”是 LSTM 单元结构的核心。下面我们来详细分析下三种“门”结构。

• 遗忘门，用来让 LSTM“忘记”之前没有用的信息。它会根据当前时刻节点的输入 XtXt、上一时刻节点的状态 Ct−1 和上一时刻节点的输出 ht−1ht−1 来决定哪些信息将被遗忘。
• 输入门，LSTM 来决定当前输入数据中哪些信息将被留下来。在 LSTM 使用遗忘门“忘记”部分信息后需要从当前的输入留下最新的记忆。输入门会根据当前时刻节点的输入 XtXt、上一时刻节点的状态 Ct−1Ct−1 和上一时刻节点的输出 ht−1ht−1 来决定哪些信息将进入当前时刻节点的状态 CtCt，模型需要记忆这个最新的信息。
• 输出门，LSTM 在得到最新节点状态 CtCt 后，结合上一时刻节点的输出 ht−1ht−1 和当前时刻节点的输入 XtXt 来决定当前时刻节点的输出。

GRU 结构

GRU（Gated Recurrent Unit）是2014年提出来的新的 RNN 架构，它是简化版的 LSTM。下面是 LSTM 和 GRU 的结构比较图（来自于网络）：



enter image description here

在超参数均调优的前提下，据说效果和 LSTM 差不多，但是参数少了1/3，不容易过拟合。如果发现 LSTM 训练出来的模型过拟合比较严重，可以试试 GRU。

实战基于 Keras 的 LSTM 和 GRU 文本分类

上面讲了那么多，但是 RNN 的知识还有很多，比如双向 RNN 等，这些需要自己去学习，下面，我们来实战一下基于 LSTM 和 GRU 的文本分类。

本次开发使用 Keras 来快速构建和训练模型，使用的数据集还是第06课使用的司法数据。

整个过程包括：

1. 语料加载
2. 分词和去停用词
3. 数据预处理
4. 使用 LSTM 分类
5. 使用 GRU 分类

第一步，引入数据处理库，停用词和语料加载：

```
#引入包
import random
import jieba
import pandas as pd

#加载停用词
stopwords=pd.read_csv('stopwords.txt',index_col=False,quoting=3,sep="\t",names=[
stopwords=stopwords['stopword'].values

#加载语料
laogong_df = pd.read_csv('beilaogongda.csv', encoding='utf-8', sep=',')
laopo_df = pd.read_csv('beilaogongda.csv', encoding='utf-8', sep=',')
erzi_df = pd.read_csv('beierzida.csv', encoding='utf-8', sep=',')
nver_df = pd.read_csv('beinverda.csv', encoding='utf-8', sep=',')
#删除语料的nan行
laogong_df.dropna(inplace=True)
laopo_df.dropna(inplace=True)
erzi_df.dropna(inplace=True)
nver_df.dropna(inplace=True)
#转换
laogong = laogong_df.segment.values.tolist()
laopo = laopo_df.segment.values.tolist()
erzi = erzi_df.segment.values.tolist()
nver = nver_df.segment.values.tolist()
```

第二步，分词和去停用词：

```
#定义分词和打标签函数preprocess_text
#参数content_lines即为上面转换的list
#参数sentences是定义的空list，用来储存打标签之后的数据
#参数category 是类型标签
def preprocess_text(content_lines, sentences, category):
    for line in content_lines:
        try:
            segs=jieba.lcut(line)
            segs = [v for v in segs if not str(v).isdigit()]#去数字
            segs = list(filter(lambda x:x.strip(), segs)) #去左右空格
            segs = list(filter(lambda x:len(x)>1, segs))#长度为1的字符
            segs = list(filter(lambda x:x not in stopwords, segs)) #去掉停用词
            sentences.append((" ".join(segs), category))# 打标签
        except Exception:
            print(line)
            continue

#调用函数、生成训练数据
sentences = []
preprocess_text(laogong, sentences,0)
preprocess_text(laopo, sentences, 1)
preprocess_text(erzi, sentences, 2)
preprocess_text(nver, sentences, 3)
```

第三步，先打散数据，使数据分布均匀，然后获取特征和标签列表：

```
#打散数据，生成更可靠的训练集
random.shuffle(sentences)

#控制台输出前10条数据，观察一下
for sentence in sentences[:10]:
    print(sentence[0], sentence[1])
#所有特征和对应标签
all_texts = [ sentence[0] for sentence in sentences]
all_labels = [ sentence[1] for sentence in sentences]
```

第四步，使用 LSTM 对数据进行分类：

```
#引入需要的模块
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.utils import to_categorical
from keras.layers import Dense, Input, Flatten, Dropout
from keras.layers import LSTM, Embedding,GRU
from keras.models import Sequential

#预定义变量
MAX_SEQUENCE_LENGTH = 100    #最大序列长度
EMBEDDING_DIM = 200    #embdding 维度
VALIDATION_SPLIT = 0.16    #验证集比例
TEST_SPLIT = 0.2    #测试集比例
#keras的sequence模块文本序列填充
tokenizer = Tokenizer()
tokenizer.fit_on_texts(all_texts)
sequences = tokenizer.texts_to_sequences(all_texts)
word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))
data = pad_sequences(sequences, maxlen=MAX_SEQUENCE_LENGTH)
labels = to_categorical(np.asarray(all_labels))
print('Shape of data tensor:', data.shape)
print('Shape of label tensor:', labels.shape)

#数据切分
p1 = int(len(data)*(1-VALIDATION_SPLIT-TEST_SPLIT))
p2 = int(len(data)*(1-TEST_SPLIT))
x_train = data[:p1]
y_train = labels[:p1]
x_val = data[p1:p2]
y_val = labels[p1:p2]
x_test = data[p2:]
y_test = labels[p2:]

#LSTM训练模型
model = Sequential()
model.add(Embedding(len(word_index) + 1, EMBEDDING_DIM, input_length=MAX_SEQUENCE
model.add(LSTM(200, dropout=0.2, recurrent_dropout=0.2))
model.add(Dropout(0.2))
model.add(Dense(64, activation='relu'))
model.add(Dense(labels.shape[1], activation='softmax'))
model.summary()
#模型编译
model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['acc'])
print(model.metrics_names)
model.fit(x_train, y_train, validation_data=(x_val, y_val), epochs=10, batch_size
model.save('lstm.h5')
#模型评估
print(model.evaluate(x_test, y_test))
```

训练过程结果为：

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_13 (Embedding)     (None, 100, 200)          78400
_________________________________________________________________
lstm_11 (LSTM)               (None, 200)               320800
_________________________________________________________________
dropout_10 (Dropout)         (None, 200)               0
_________________________________________________________________
dense_11 (Dense)             (None, 64)                12864
_________________________________________________________________
dense_12 (Dense)             (None, 4)                 260
=================================================================
```

enter image description here

第五步，使用 GRU 进行文本分类，上面就是完整的使用 LSTM 进行 文本分类，如果使用 GRU 只需要改变模型训练部分：

```
model = Sequential()
model.add(Embedding(len(word_index) + 1, EMBEDDING_DIM, input_length=MAX_SEQUENCE
model.add(GRU(200, dropout=0.2, recurrent_dropout=0.2))
model.add(Dropout(0.2))
model.add(Dense(64, activation='relu'))
model.add(Dense(labels.shape[1], activation='softmax'))
model.summary()

model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['acc'])
print(model.metrics_names)
model.fit(x_train, y_train, validation_data=(x_val, y_val), epochs=10, batch_size
model.save('lstm.h5')

print(model.evaluate(x_test, y_test))
```

训练过程结果：

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_12 (Embedding)     (None, 100, 200)          78400
_________________________________________________________________
gru_4 (GRU)                  (None, 200)               240600
_________________________________________________________________
dropout_9 (Dropout)          (None, 200)               0
_________________________________________________________________
dense_9 (Dense)              (None, 64)                12864
_________________________________________________________________
dense_10 (Dense)             (None, 4)                 260
=================================================================
Total params: 332,124
Trainable params: 332,124
Non-trainable params: 0
_________________________________________________________________
['loss', 'acc']
Train on 1102 samples, validate on 275 samples
Epoch 1/10
1102/1102 [==============================] - 11s 10ms/step - loss: 1.3378 - acc: 0.3848 - val_loss: 1.1833 - val_acc: 0.4836
Epoch 2/10
1102/1102 [==============================] - 10s 9ms/step - loss: 0.9942 - acc: 0.5771 - val_loss: 0.8684 - val_acc: 0.6364
Epoch 3/10
1102/1102 [==============================] - 10s 9ms/step - loss: 0.6594 - acc: 0.6797 - val_loss: 0.5194 - val_acc: 0.7200
Epoch 4/10
1102/1102 [==============================] - 10s 9ms/step - loss: 0.4273 - acc: 0.7468 - val_loss: 0.3908 - val_acc: 0.7418
Epoch 5/10
1102/1102 [==============================] - 10s 9ms/step - loss: 0.3745 - acc: 0.7450 - val_loss: 0.3793 - val_acc: 0.7382
Epoch 6/10
1102/1102 [==============================] - 10s 9ms/step - loss: 0.4281 - acc: 0.7296 - val_loss: 0.3945 - val_acc: 0.6909
Epoch 7/10
1102/1102 [==============================] - 10s 9ms/step - loss: 0.3795 - acc: 0.7505 - val_loss: 0.3803 - val_acc: 0.7018
Epoch 8/10
1102/1102 [==============================] - 11s 10ms/step - loss: 0.3841 - acc: 0.7468 - val_loss: 0.3751 - val_acc: 0.7164
Epoch 9/10
1102/1102 [==============================] - 11s 10ms/step - loss: 0.3602 - acc: 0.7414 - val_loss: 0.3744 - val_acc: 0.7418
Epoch 10/10
1102/1102 [==============================] - 10s 9ms/step - loss: 0.3592 - acc: 0.7523 - val_loss: 0.3756 - val_acc: 0.7382
345/345 [==============================] - 1s 4ms/step
[0.34260822119920148, 0.77101449206255479]
```

enter image description here

总结

本文从词袋模型谈起，旨在引出语言模型 N-gram 以及其优化模型 NNLM 和 RNNLM，并通过 RNN 以及其变种 LSTM 和 GRU 模型，理解其如何处理类似序列数据的原理，并实战基于 LSTM 和 GRU 的中文文本分类。

参考文献：

1. Hinton 神经网络公开课编程题2——神经概率语言模型（ NNLM ）(https://blog.csdn.net/u012328159/article/details/72847297)

2. Recurrent Neural Networks Tutorial, Part 3 – Backpropagation Through Time and Vanishing Gradients (http://www.wildml.com/2015/10/recurrent-neural-networks-tutorial-part-3-backpropagation-through-time-and-vanishing-gradients/)

米饭超人 (/u/ba83fba00eef) ♂

写了 199636 字，被 473 人关注，获得了 532 个喜欢

(/u/ba83fba00eef)

+关注

每一件与众不同的绝世好东西，其实都是以无比寂寞的勤奋为前提的，要么是血，要么是汗，要么是大把大…