《自然语言处理》实验报告

一、实验目的

理解、掌握深度学习模型 CNN 和 LSTM 的结构和 tensorflow 开发框架，掌握华为 ModelArts 平台的使用方法。

二、实验项目内容

1. 练习使用 ModelArts 开发平台，包括开发流程、对象存储服务、自定义模型和预置模型加载、运行等；
2. 在 tensorflow 框架基础上，开发一基于 CNN 和 LSTM 的文本分类模型；
3. 在 ModelArts 平台或本地开发环境中，对自定义 CNN 和 LSTM 文本分类模型进行测试和优化。

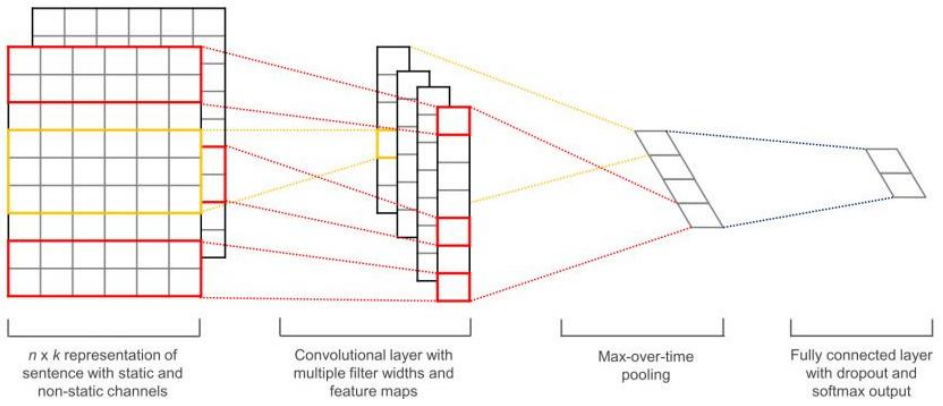
三、实验过程或算法（源程序）

实验原理

传统语言模型（如 n-gram）以词为单元进行划分，无法兼顾语义的相似性。并且在面对数据稀疏时，会出现极端的预测结果。面对上述传统语言模型的先天缺陷，以传统神经网络为基础，引入了神经语言模型，常见的有前馈神经网络语言模型。本实验需要开发的 CNN 和 LSTM 的神经网络模型都是 FNN 的变体。

1. CNN 模型描述

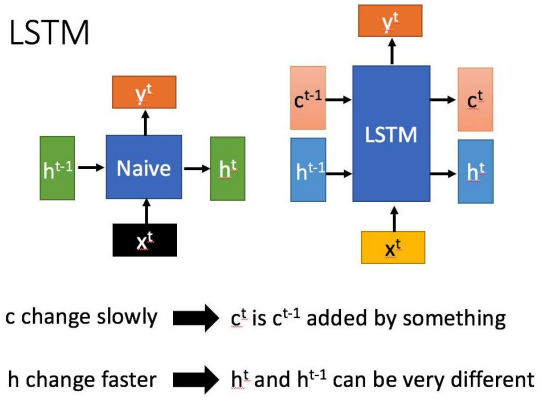
传统全连接神经网络用于文本分类任务时，如果想要较好的分类效果则需要使用大量的参数。这不仅使得训练困难，并且过多的参数也可能会导致过拟合。在卷积神经网络中，使用卷积层提取特征，池化层数据降维，全连接层将局部特征还原为全局特征，最终使用 Softmax 网络实现分类。



在使用 CNN 进行文本分类时，模型输入的维度不同于图像。图像每个像素点由 RGB 3 个通道组成，加上一张二维平面，可以说是三维的。但文本分类的输入只是一个二维矩阵，包含 M 个词向量，每一行是一个 N 维词向量。并且在卷积层，对卷积核大小也有要求：一般来说，**卷积核的长度和词向量的维度应该是一致的**。比如一个词向量是 N 维的，那么卷积核就应该是 X*N 维的，其中 X 表示卷积过程想提取 X 个词间的特征。

2. LSTM 模型描述

循环神经网络 RNN 一般用于处理序列数据，在文本分类中能够“记住”词语上下文的内容的影响，对文本分类有着重要的应用。但随着网络深度的增加，RNN 网络出现了梯度消失或爆炸等问题，网络变得无法训练。于是，选择性保留或遗忘某些信息的长短时记忆网络 LSTM 应运而生。



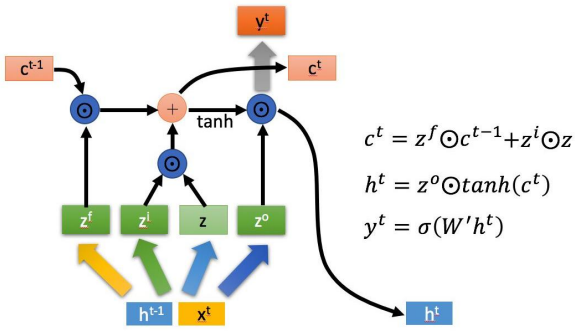
相比于 RNN 只有一个状态 Ht，LSTM 有两个状态，分别是记忆状态 Ct 和历史积累 Ht。Ct 变化的很慢，每次只在前一次上有少许增量，而 Ht 在不同节点间差别很大。

（1）遗忘门：z^f 作为忘记门控，控制上一个状态 Ct-1 中的哪些需要忘记；

（2）输入门：对当前的输入 z 选择性记忆，由 z^i 执行；

（3）输出门：决定哪些将会被当成当前状态的输出，通过 z^o 执行

LSTM 通过门控状态控制传输状态，记住需要长时间记忆的，忘记不重要的信息。最终的输出 Yt 由 Ht 变化得到。



算法源程序

本实验采用的是 20_newsgroup 数据集，该数据集收录了 18000 多条新闻组文档，均匀分为 20 个不同主题。一些新闻组的主题特别相似，还有一些完全不相关：

comp.graphics comp.os.ms-windows.misc comp.sys.ibm.pc.hardware comp.sys.mac.hardware comp.windows.x	rec.autos rec.motorcycles rec.sport.baseball rec.sport.hockey	sci.crypt sci.electronics sci.med sci.space
misc.forsale	talk.politics.misc talk.politics.guns talk.politics.mideast	talk.religion.misc alt.atheism soc.religion.christian

1. 数据处理

（部分代码参考 Keras 官网实例详解 4.37 pretrained_word_embeddings.py）

（1）读取词向量

```
BASE_DIR = ''
# glove模型路径
GLOVE_DIR = os.path.join(BASE_DIR, 'glove.6B.100d.txt')
# 文本语料路径
TEXT_DATA_DIR = os.path.join(BASE_DIR, '20_newsgroup')

MAX_SEQUENCE_LENGTH = 1000
MAX_NUM_WORDS = 20000
EMBEDDING_DIM = 100
VALIDATION_SPLIT = 0.2

# 词向量
embeddings_index = {}
with open(os.path.join(BASE_DIR, 'glove.6B.100d.txt'), 'r', encoding='utf-8') as f:
    for line in f:
        word, coefs = line.split(maxsplit=1)
        coefs = np.fromstring(coefs, 'f', sep=' ')
        embeddings_index[word] = coefs
```

（2）获取训练文本与标签

```
texts = []  # 文本样例
labels_index = {}  # 将标签名称映射到id
labels = []  # 标签号
for name in sorted(os.listdir(TEXT_DATA_DIR)):
    path = os.path.join(TEXT_DATA_DIR, name)
    if os.path.isdir(path):
        label_id = len(labels_index)
        labels_index[name] = label_id
        for fname in sorted(os.listdir(path)):
            if fname.isdigit():
                fpath = os.path.join(path, fname)
                args = {} if sys.version_info < (
                    3,) else {'encoding': 'latin-1'}
                with open(fpath, **args) as f:
                    t = f.read()
                    i = t.find('\n\n')  # 去掉头部
                    if 0 < i:
                        t = t[i:]
                    texts.append(t)
                labels.append(label_id)
```

（3）使用 token 分词把词转化为标号

```
tokenizer = Tokenizer(num_words=MAX_NUM_WORDS)
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)
word_index = tokenizer.word_index

data = pad_sequences(sequences, maxlen=MAX_SEQUENCE_LENGTH)
labels = to_categorical(np.asarray(labels))
print('Shape of data tensor:', data.shape)
print('Shape of label tensor:', labels.shape)
```

（4）将词向量字典传入 tokenizer 并根据将训练数据转化为 sequences

```
indices = np.arange(data.shape[0])
np.random.shuffle(indices)#打乱数据集
data = data[indices]
labels = labels[indices]
num_validation_samples = int(VALIDATION_SPLIT * data.shape[0])
```

（5）划分训练集和测试集

```
x_train = data[:-num_validation_samples]
y_train = labels[:-num_validation_samples]
x_val = data[-num_validation_samples:]
y_val = labels[-num_validation_samples:]
```

（6）准备嵌入矩阵

```
num_words = min(MAX_NUM_WORDS, len(word_index) + 1)
embedding_matrix = np.zeros((num_words, EMBEDDING_DIM))
for word, i in word_index.items():
    if i >= MAX_NUM_WORDS:
        continue
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        # 嵌入索引中未找到的单词将全部为零.
        # 从预训练模型的词向量到语料库的词向量映射
        embedding_matrix[i] = embedding_vector
```

（7）将预先训练的单词放入嵌入层并保持嵌入固定

```
embedding_layer = Embedding(num_words,
                            EMBEDDING_DIM,
                            embeddings_initializer=Constant(embedding_matrix),
                            input_length=MAX_SEQUENCE_LENGTH,
                            trainable=False)
```

2. CNN 模型

（1）定义卷积层输出为 128 维，卷积窗口大小为 5

```
sequence_input = Input(shape=(MAX_SEQUENCE_LENGTH,), dtype='int32')
embedded_sequences = embedding_layer(sequence_input)
x = Conv1D(128, 5, activation='relu')(embedded_sequences)    #输出128维，卷积窗口大小为5
```

（2）加入池化层，采用最大池化法

```
x = MaxPooling1D(5)(x)
```

（3）加入全连接层，并使用 Relu 激活函数

```
x = Dense(128, activation='relu')(x)
```

实际应用中，往往采用多层卷积与池化，因此可以将上述操作叠加嵌套。本实验采用了 2 层卷积+3 层池化，最后使用全连接层输出：

```
# 构建网络
sequence_input = Input(shape=(MAX_SEQUENCE_LENGTH,), dtype='int32')
embedded_sequences = embedding_layer(sequence_input)
x = Conv1D(128, 5, activation='relu')(embedded_sequences)    #输出128维，卷积窗口大小为5

x = MaxPooling1D(5)(x)
x = Conv1D(128, 5, activation='relu')(x)
x = MaxPooling1D(5)(x)
x = Conv1D(128, 5, activation='relu')(x)
x = GlobalMaxPooling1D()(x)
x = Dense(128, activation='relu')(x)
preds = Dense(len(labels_index), activation='softmax')(x)
```

（4）训练模型，采用 RMSProp 优化器

```
# 训练模型
model = Model(sequence_input, preds)
model.compile(loss='categorical_crossentropy', optimizer='rmsprop', metrics=['acc'])
history=model.fit(x_train, y_train,
                  batch_size=128,
                  epochs=20,
                  validation_data=(x_val, y_val))
```

3. LSTM 模型

其他同 CNN，只需在训练模型时加入 LSTM 网络即可：

```
# 训练模型
model = tf.keras.models.Sequential()
model.add(embedding_layer)
model.add(LSTM(16, dropout=0.5, recurrent_dropout=0.2))
model.add(tf.keras.layers.Activation('softmax'))
model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['acc'])
history=model.fit(x_train, y_train,
                  batch_size=128,
                  epochs=20,
                  validation_data=(x_val, y_val))
```

实验过程

1. 上传数据

（1）进入OBS对象存储服务，创建桶，选择“华南-广州”区域，存储策略为单AZ存储，桶策略为私有，其他保持默认即可：

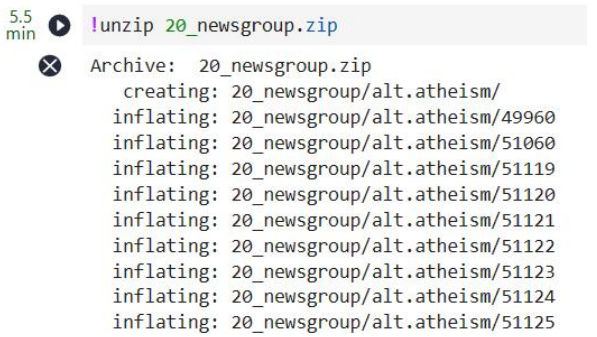
（2）在桶内创建三个文件夹分别存放代码、数据集和输出的模型：

（3）上传代码和数据集，由于100个文件上限，需要将新闻数据集压缩后上传：

2. CNN/LSTM 模型分类

（1）创建Notebok，镜像选择tensorflow1.13-cuda10.0-cudnn7-ubuntu18.04（另一个版本的镜像有千奇百怪的问题），配置如下：

（2）将OBS文件上传至Notebook并解压新闻组压缩文件：



（3）运行事先写好的代码即可

四、实验结果及分析和（或）源程序调试过程

结果分析：

1. CNN 模型

CNN 模型迭代过程训练集准确率与测试集准确率如下，训练集准确率高达 95%，测试集准确率大约为 75%：

```
Train on 15998 samples, validate on 3999 samples
Epoch 1/20
15998/15998 [==============================] - 4s 243us/step - loss: 2.4134 - acc: 0.2248 - val_loss: 1.6438 - val_acc: 0.4364
Epoch 2/20
15998/15998 [==============================] - 1s 80us/step - loss: 1.4153 - acc: 0.4914 - val_loss: 1.2057 - val_acc: 0.5724
Epoch 3/20
15998/15998 [==============================] - 1s 80us/step - loss: 1.0948 - acc: 0.6174 - val_loss: 1.0870 - val_acc: 0.6302
Epoch 4/20
15998/15998 [==============================] - 1s 80us/step - loss: 0.8833 - acc: 0.6938 - val_loss: 0.9448 - val_acc: 0.6842
Epoch 5/20
15998/15998 [==============================] - 1s 80us/step - loss: 0.7439 - acc: 0.7480 - val_loss: 0.9200 - val_acc: 0.7032
Epoch 6/20
15998/15998 [==============================] - 1s 80us/step - loss: 0.6458 - acc: 0.7805 - val_loss: 0.9216 - val_acc: 0.6969
Epoch 7/20
15998/15998 [==============================] - 1s 80us/step - loss: 0.5593 - acc: 0.8145 - val_loss: 0.8694 - val_acc: 0.7217
Epoch 8/20
15998/15998 [==============================] - 1s 80us/step - loss: 0.4772 - acc: 0.8428 - val_loss: 0.8649 - val_acc: 0.7259
Epoch 9/20
15998/15998 [==============================] - 1s 80us/step - loss: 0.4253 - acc: 0.8605 - val_loss: 0.8254 - val_acc: 0.7384
Epoch 10/20
15998/15998 [==============================] - 1s 80us/step - loss: 0.3698 - acc: 0.8775 - val_loss: 0.8221 - val_acc: 0.7374
Epoch 11/20
15998/15998 [==============================] - 1s 80us/step - loss: 0.3176 - acc: 0.8991 - val_loss: 0.8440 - val_acc: 0.7369
Epoch 12/20
15998/15998 [==============================] - 1s 80us/step - loss: 0.2878 - acc: 0.9082 - val_loss: 0.8599 - val_acc: 0.7414
Epoch 13/20
15998/15998 [==============================] - 1s 80us/step - loss: 0.2632 - acc: 0.9141 - val_loss: 0.8808 - val_acc: 0.7387
Epoch 14/20
15998/15998 [==============================] - 1s 80us/step - loss: 0.2255 - acc: 0.9271 - val_loss: 0.8919 - val_acc: 0.7417
Epoch 15/20
15998/15998 [==============================] - 1s 80us/step - loss: 0.2011 - acc: 0.9383 - val_loss: 0.9177 - val_acc: 0.7472
Epoch 16/20
15998/15998 [==============================] - 1s 80us/step - loss: 0.1847 - acc: 0.9427 - val_loss: 0.8997 - val_acc: 0.7462
Epoch 17/20
15998/15998 [==============================] - 1s 80us/step - loss: 0.1674 - acc: 0.9491 - val_loss: 0.9220 - val_acc: 0.7437
Epoch 18/20
15998/15998 [==============================] - 1s 80us/step - loss: 0.1576 - acc: 0.9517 - val_loss: 0.9166 - val_acc: 0.7492
Epoch 19/20
15998/15998 [==============================] - 1s 80us/step - loss: 0.1448 - acc: 0.9556 - val_loss: 0.9605 - val_acc: 0.7399
Epoch 20/20
15998/15998 [==============================] - 1s 80us/step - loss: 0.1377 - acc: 0.9573 - val_loss: 0.9937 - val_acc: 0.7404
```

训练集准确率与测试集准确率差距过大的原因是过拟合，加入正则化项和 Dropout 后略有缓解。

2. LSTM 模型

LSTM 模型迭代过程训练集准确率与测试集准确率如下，训练集准确率高达 93%，测试集准确率大约为 74%：

```
Train on 15998 samples, validate on 3999 samples
Epoch 1/20
15998/15998 [==============================] - 14s 905us/step - loss: 2.5490 - acc: 0.1854 - val_loss: 2.1758 - val_acc: 0.2628
Epoch 2/20
15998/15998 [==============================] - 14s 887us/step - loss: 2.0205 - acc: 0.3287 - val_loss: 2.5957 - val_acc: 0.2351
Epoch 3/20
15998/15998 [==============================] - 14s 892us/step - loss: 1.6304 - acc: 0.4487 - val_loss: 1.4841 - val_acc: 0.4954
Epoch 4/20
15998/15998 [==============================] - 14s 893us/step - loss: 1.3497 - acc: 0.5421 - val_loss: 1.2601 - val_acc: 0.5679
Epoch 5/20
15998/15998 [==============================] - 14s 895us/step - loss: 1.1335 - acc: 0.6151 - val_loss: 1.1418 - val_acc: 0.6122
Epoch 6/20
15998/15998 [==============================] - 14s 895us/step - loss: 0.9855 - acc: 0.6681 - val_loss: 1.0660 - val_acc: 0.6399
Epoch 7/20
15998/15998 [==============================] - 14s 898us/step - loss: 0.8693 - acc: 0.7060 - val_loss: 0.9468 - val_acc: 0.6807
Epoch 8/20
15998/15998 [==============================] - 14s 897us/step - loss: 0.7723 - acc: 0.7398 - val_loss: 0.9227 - val_acc: 0.6919
Epoch 9/20
15998/15998 [==============================] - 14s 880us/step - loss: 0.6914 - acc: 0.7652 - val_loss: 0.8846 - val_acc: 0.7117
Epoch 10/20
15998/15998 [==============================] - 14s 896us/step - loss: 0.6128 - acc: 0.7934 - val_loss: 0.8846 - val_acc: 0.7154
Epoch 11/20
15998/15998 [==============================] - 14s 895us/step - loss: 0.5426 - acc: 0.8171 - val_loss: 0.8321 - val_acc: 0.7297
Epoch 12/20
15998/15998 [==============================] - 14s 896us/step - loss: 0.4795 - acc: 0.8366 - val_loss: 0.9088 - val_acc: 0.7149
Epoch 13/20
15998/15998 [==============================] - 14s 894us/step - loss: 0.4230 - acc: 0.8534 - val_loss: 0.8540 - val_acc: 0.7389
Epoch 14/20
15998/15998 [==============================] - 14s 896us/step - loss: 0.3756 - acc: 0.8687 - val_loss: 0.9017 - val_acc: 0.7317
Epoch 15/20
15998/15998 [==============================] - 14s 899us/step - loss: 0.3289 - acc: 0.8854 - val_loss: 0.8736 - val_acc: 0.7414
Epoch 16/20
15998/15998 [==============================] - 14s 898us/step - loss: 0.2837 - acc: 0.9001 - val_loss: 0.9511 - val_acc: 0.7347
Epoch 17/20
15998/15998 [==============================] - 14s 896us/step - loss: 0.2502 - acc: 0.9137 - val_loss: 0.9982 - val_acc: 0.7317
Epoch 18/20
15998/15998 [==============================] - 14s 896us/step - loss: 0.2247 - acc: 0.9202 - val_loss: 1.0171 - val_acc: 0.7387
Epoch 19/20
15998/15998 [==============================] - 14s 896us/step - loss: 0.1955 - acc: 0.9282 - val_loss: 1.0208 - val_acc: 0.7444
Epoch 20/20
15998/15998 [==============================] - 14s 896us/step - loss: 0.1782 - acc: 0.9372 - val_loss: 1.1304 - val_acc: 0.7347
```

（本实验采用的是 LSTM 的变体 CuDNNLSTM，训练速度更快）

调试过程：

1. 问题：已经创建了桶，但 ModelArts 创建作业时无可用的桶：

解决方法：ModelArts 控制台的区域需要与桶的区域一致。

2. 问题：上传 20_newsgroup 数据集时超出文件上限，无法上传；

解决方法：压缩为 zip 文件后配置在线解压策略，参考官方教程。

3. 问题：训练时运行错误，找不到相应的目录或代码：

解决方法：将训练输入模块的本地路径进行复制，粘贴到代码相应位置即可。

4. 问题：CNN 模型训练集上准确率较高，但测试集显著降低：

```
15998/15998 [==============================] - 2s 134us/sample - loss: 0.1397 - acc: 0.9565 - val_loss: 0.8768 - val_acc: 0.7567
Epoch 18/20
15998/15998 [==============================] - 2s 132us/sample - loss: 0.1383 - acc: 0.9561 - val_loss: 0.9009 - val_acc: 0.7542
Epoch 19/20
15998/15998 [==============================] - 2s 132us/sample - loss: 0.1326 - acc: 0.9566 - val_loss: 0.8682 - val_acc: 0.7667
Epoch 20/20
15998/15998 [==============================] - 2s 132us/sample - loss: 0.1230 - acc: 0.9596 - val_loss: 0.8984 - val_acc: 0.7537
```

解决方法：模型过拟合，加入正则项并采用 Dropout 策略。

```
tf.keras.regularizers.l2(0.1)(embedded_sequences)
```

```
x = Dropout(0.1)(x)
```

5. 问题：引入模块出错

```
~/anaconda3/envs/TensorFlow-2.1/lib/python3.7/site-packages/keras/backend_config.py in <module>
     31
     32 @keras_export("keras.backend.epsilon")
---> 33 @tf.__internal__.dispatch.add_dispatch_support
     34 def epsilon():
     35     """Returns the value of the fuzz factor used in numeric expressions.

AttributeError: module 'tensorflow_core.compat.v2' has no attribute '__internal__'
```

解决方法：tensorflow 与 keras 版本冲突，诸如此类冲突本实验中还有很多，指定版本即可

```
4 min
!pip uninstall tensorflow
!pip install tensorflow==2.3.0
!pip uninstall keras
!pip install keras==2.3.1
```

```
Found existing installation: tensorflow 2.1.0
Uninstalling tensorflow-2.1.0:
  Would remove:
    /home/ma-user/anaconda3/envs/TensorFlow-2.1/bin/estimator_ckpt_converter
    /home/ma-user/anaconda3/envs/TensorFlow-2.1/bin/saved_model_cli
    /home/ma-user/anaconda3/envs/TensorFlow-2.1/bin/tensorboard
    /home/ma-user/anaconda3/envs/TensorFlow-2.1/bin/tf_upgrade_v2
    /home/ma-user/anaconda3/envs/TensorFlow-2.1/bin/tflite_convert
    /home/ma-user/anaconda3/envs/TensorFlow-2.1/bin/toco
    /home/ma-user/anaconda3/envs/TensorFlow-2.1/bin/toco_from_protos
    /home/ma-user/anaconda3/envs/TensorFlow-2.1/lib/python3.7/site-packages/tensorflow-2.1.0.dist-info/*
    /home/ma-user/anaconda3/envs/TensorFlow-2.1/lib/python3.7/site-packages/tensorflow/*
    /home/ma-user/anaconda3/envs/TensorFlow-2.1/lib/python3.7/site-packages/tensorflow_core/*
```

版本换了好多次，我也忘了最后是什么版本了...

6. 问题：为了加快训练速度，我调大了 batch_size，但测试集准确率低的离

谱：

```
# 训练
LSTM=model.fit(x_train, y_train, batch_size=1024,
          epochs=10, validation_data=(x_val, y_val))
```

```
Train on 15998 samples, validate on 3999 samples
Epoch 1/10
15998/15998 [==============================] - 22s 1ms/step - loss: 2.9148 - acc: 0.1071 - val_loss: 2.8529 - val_acc: 0.1135
Epoch 2/10
15998/15998 [==============================] - 19s 1ms/step - loss: 2.7621 - acc: 0.1608 - val_loss: 2.4591 - val_acc: 0.2236
Epoch 3/10
15998/15998 [==============================] - 19s 1ms/step - loss: 2.6761 - acc: 0.1862 - val_loss: 2.6399 - val_acc: 0.1805
Epoch 4/10
15998/15998 [==============================] - 19s 1ms/step - loss: 2.3824 - acc: 0.2323 - val_loss: 2.2659 - val_acc: 0.2453
Epoch 5/10
15998/15998 [==============================] - 19s 1ms/step - loss: 2.3239 - acc: 0.2546 - val_loss: 2.9072 - val_acc: 0.1528
Epoch 6/10
15998/15998 [==============================] - 19s 1ms/step - loss: 2.3092 - acc: 0.2630 - val_loss: 2.0856 - val_acc: 0.2981
Epoch 7/10
15998/15998 [==============================] - 19s 1ms/step - loss: 2.2032 - acc: 0.2783 - val_loss: 2.1609 - val_acc: 0.3028
Epoch 8/10
15998/15998 [==============================] - 19s 1ms/step - loss: 2.0833 - acc: 0.3116 - val_loss: 2.1255 - val_acc: 0.2756
Epoch 9/10
15998/15998 [==============================] - 19s 1ms/step - loss: 2.0331 - acc: 0.3209 - val_loss: 1.9897 - val_acc: 0.3256
Epoch 10/10
15998/15998 [==============================] - 19s 1ms/step - loss: 1.9608 - acc: 0.3519 - val_loss: 2.5744 - val_acc: 0.2261
```

解决方法：尽量保持 batch_size 不超过 128

7. 问题：经常出现加入某一层失败：

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/tmp/ipykernel_2100/643820015.py in <module>
      6 model = keras.models.Sequential()
      7 model.add(Embedding(max_features, 128))
----> 8 model.add(LSTM(128, dropout=0.2, return_sequences=False, recurrent_dropout=0.2))
      9 model.add(Dense(20, activation='softmax'))
     10 model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

TypeError: 'History' object is not callable
```

解决方法：重新运行划分数据集的 cell 即可

参考资料

[1] 卷积神经网络（CNN）应用于自然语言处理（NLP）

[2] 20 Newsgroups 数据集介绍

[3] 使用 20_newsgroup 集做训练集，载入 Glove 预训练权重训练模型

[4] fetch_20newsgroups 函数介绍

[5] 官网实例详解 4.37-keras 学习笔记四