# BiLSTM实战1

## 课程介绍

本节以句子级情感分类任务为例介绍使用BiLSTM进行建模，自然语言处理中情感分类任务是对给定文本进行情感倾向分类的任务，粗略来看可以认为其是分类任务中的一类。对于情感分类任务，目前通常的做法是先对词或者短语进行表示，再通过某种组合方式把句子中词的表示组合成句子的表示。最后，利用句子的表示对句子进行情感分类。BiLSTM是Bi-directional Long Short-Term Memory的缩写，是由前向LSTM与后向LSTM组合而成。两者在自然语言处理任务中都常被用来建模上下文信息。利用传统的LSTM对句子进行建模存在一个问题，无法编码从后到前的信息。在更细粒度的分类时，如对于强程度的褒义、弱程度的褒义、中性、弱程度的贬义、强程度的贬义的五分类任务需要注意情感词、程度词、否定词之间的交互。举一个例子，“这个餐厅脏得不行，没有隔壁好”，这里的“不行”是对“脏”的程度的一种修饰，而通过BiLSTM可以更好的捕捉双向的语义依赖。

## 实验概述

## 实验步骤

### 步骤3：模型定义

在本节中，主要介绍相关的模型的定义，主要包含以下几个常用的基础模型：CNN,RNN(BiLSTM),attention-RNN以及RCNN,接下来会对各个模型的配置进行详解。

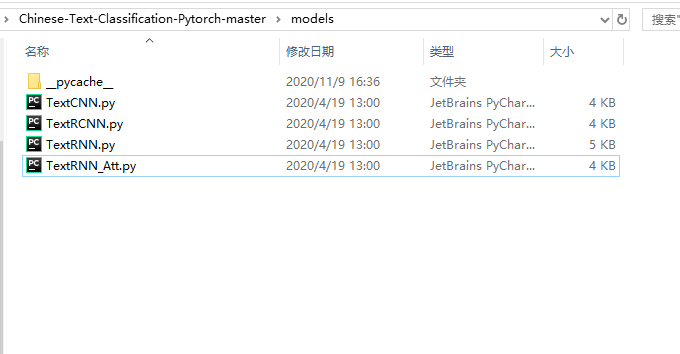


图 1 模型类型

接下来主要介绍各个模型的模块定义，主要分为两部分内容，包括模型本身的定义以及模型对应的配置(超参数)的定义。首先在CNN模型中，即TestRNN.py中，class Config(object)为配置参数定义。

|  |
| --- |
| # coding: UTF-8  import torch  import torch.nn as nn  import numpy as np  class Config(object):  """配置参数"""  def \_\_init\_\_(self, dataset, embedding):  #调用模型名称  self.model\_name = 'TextRNN'  # 训练集、验证集、测试集路径  self.train\_path = dataset + '/data/train.txt' # 训练集  self.dev\_path = dataset + '/data/dev.txt' # 验证集  self.test\_path = dataset + '/data/test.txt' # 测试集  # 数据集的所有类别  self.class\_list = [x.strip() for x in open(  dataset + '/data/class.txt', encoding='utf-8').readlines()] # 类别名单  # 构建好的词/字典路径  self.vocab\_path = dataset + '/data/vocab.pkl' # 词表  # 训练好的模型参数保存路径  self.save\_path = dataset + '/saved\_dict/' + self.model\_name + '.ckpt' # 模型训练结果  # 模型日志保存路径  self.log\_path = dataset + '/log/' + self.model\_name  # 如果词/字嵌入矩阵不随机初始化 则加载初始化好的词/字嵌入矩阵 类别为float32 并转换为tensor 否则为None  self.embedding\_pretrained = torch.tensor(  np.load(dataset + '/data/' + embedding)["embeddings"].astype('float32'))\  if embedding != 'random' else None # 预训练词向量  self.device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu') # 设备  self.dropout = 0.5 # 随机失活  self.require\_improvement = 1000 # 若超过1000batch效果还没提升，则提前结束训练  self.num\_classes = len(self.class\_list) # 类别数  self.n\_vocab = 0 # 词表大小，在运行时赋值  self.num\_epochs = 10 # epoch数  self.batch\_size = 128 # mini-batch大小  self.pad\_size = 32 # 每句话处理成的长度(短填长切)  self.learning\_rate = 1e-3 # 学习率  self.embed = self.embedding\_pretrained.size(1)\  if self.embedding\_pretrained is not None else 300 # 字向量维度, 若使用了预训练词向量，则维度统一  self.hidden\_size = 128 # lstm隐藏层  self.num\_layers = 2 # lstm层数 |

在测试函数test()中，首先加载当前在验证集测试中损失最小的参数，然后计算模型参数在测试集上的精度，每个batch的平均损失 分类报告、混淆矩阵。

|  |
| --- |
| def test(config, model, test\_iter):  # test  # 加载使验证集损失最小的参数  model.load\_state\_dict(torch.load(config.save\_path))  model.eval()  start\_time = time.time()  # 计算测试集准确率，每个batch的平均损失 分类报告、混淆矩阵  test\_acc, test\_loss, test\_report, test\_confusion = evaluate(config, model, test\_iter, test=True)  msg = 'Test Loss: {0:>5.2}, Test Acc: {1:>6.2%}'  print(msg.format(test\_loss, test\_acc))  print("Precision, Recall and F1-Score...")  print(test\_report)  print("Confusion Matrix...")  print(test\_confusion)  time\_dif = get\_time\_dif(start\_time)  print("Time usage:", time\_dif) |

然后定义模型类，在这里我们采用了一个双层的bilstm进行训练，网络结构为：embeding-lstm(2)-fc，代码详解如下

|  |
| --- |
| class Model(nn.Module):  def \_\_init\_\_(self, config):  super(Model, self).\_\_init\_\_()  if config.embedding\_pretrained is not None:# 加载初始化好的预训练词/字嵌入矩阵 微调funetuning  self.embedding = nn.Embedding.from\_pretrained(config.embedding\_pretrained, freeze=False)  else:# 否则随机初始化词/字嵌入矩阵 指定填充对应的索引  self.embedding = nn.Embedding(config.n\_vocab, config.embed, padding\_idx=config.n\_vocab - 1)  # 2层双向lstm batch\_size为第一维度  self.lstm = nn.LSTM(config.embed, config.hidden\_size, config.num\_layers,  bidirectional=True, batch\_first=True, dropout=config.dropout)  # 输出层  self.fc = nn.Linear(config.hidden\_size \* 2, config.num\_classes)  def forward(self, x):  x, \_ = x#输入大小为（batch,SEQ\_LEN）  out = self.embedding(x) # [batch\_size, seq\_len, embeding]=[128, 32, 300]  out, \_ = self.lstm(out)  out = self.fc(out[:, -1, :]) # 句子最后时刻的 hidden state(batch,hidden\_size\*2)->(batch,classes)  return out |

在TestRCNN.py中，定义与RNN相似，具体参数与模型定义如下

|  |
| --- |
| # coding: UTF-8  import torch  import torch.nn as nn  import torch.nn.functional as F  import numpy as np  class Config(object):  """配置参数"""  def \_\_init\_\_(self, dataset, embedding):  self.model\_name = 'TextRCNN'  self.train\_path = dataset + '/data/train.txt' # 训练集  self.dev\_path = dataset + '/data/dev.txt' # 验证集  self.test\_path = dataset + '/data/test.txt' # 测试集  self.class\_list = [x.strip() for x in open(  dataset + '/data/class.txt', encoding='utf-8').readlines()] # 类别名单  self.vocab\_path = dataset + '/data/vocab.pkl' # 词表  self.save\_path = dataset + '/saved\_dict/' + self.model\_name + '.ckpt' # 模型训练结果  self.log\_path = dataset + '/log/' + self.model\_name  self.embedding\_pretrained = torch.tensor(  np.load(dataset + '/data/' + embedding)["embeddings"].astype('float32'))\  if embedding != 'random' else None # 预训练词向量  self.device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu') # 设备  self.dropout = 1.0 # 随机失活  self.require\_improvement = 1000 # 若超过1000batch效果还没提升，则提前结束训练  self.num\_classes = len(self.class\_list) # 类别数  self.n\_vocab = 0 # 词表大小，在运行时赋值  self.num\_epochs = 10 # epoch数  self.batch\_size = 128 # mini-batch大小  self.pad\_size = 32 # 每句话处理成的长度(短填长切)  self.learning\_rate = 1e-3 # 学习率  self.embed = self.embedding\_pretrained.size(1)\  if self.embedding\_pretrained is not None else 300 # 字向量维度, 若使用了预训练词向量，则维度统一  self.hidden\_size = 256 # lstm隐藏层  self.num\_layers = 1 # lstm层数  '''Recurrent Convolutional Neural Networks for Text Classification'''  class Model(nn.Module):  def \_\_init\_\_(self, config):  super(Model, self).\_\_init\_\_()  if config.embedding\_pretrained is not None:# 加载初始化好的预训练词/字嵌入矩阵 微调funetuning  self.embedding = nn.Embedding.from\_pretrained(config.embedding\_pretrained, freeze=False)  else:# 否则随机初始化词/字嵌入矩阵 指定填充对应的索引  self.embedding = nn.Embedding(config.n\_vocab, config.embed, padding\_idx=config.n\_vocab - 1)  # 单层双向lstm batch\_size为第一维度  self.lstm = nn.LSTM(config.embed, config.hidden\_size, config.num\_layers,  bidirectional=True, batch\_first=True, dropout=config.dropout)  # 沿长度方向作全局最大池化  self.maxpool = nn.MaxPool1d(config.pad\_size)  #输出层  self.fc = nn.Linear(config.hidden\_size \* 2 + config.embed, config.num\_classes)  def forward(self, x):  x, \_ = x  embed = self.embedding(x) # [batch\_size, seq\_len, embeding]=[64, 32, 64]  out, \_ = self.lstm(embed)  out = torch.cat((embed, out), 2) #把词嵌入和lstm输出进行拼接 (batch,seq\_len.embed+hidden\_size\*2)  out = F.relu(out)  out = out.permute(0, 2, 1) #进行转置，变更输入形状(batch,embed+hidden\_size\*2,seq\_len)  out = self.maxpool(out).squeeze()#（batch,embed+hidden\_size\*2）  out = self.fc(out)#(batch,classes)  return out |