情感分析作业报告

作者

魏家栋，计72班，学号2017011445

实现内容

利用神经网络进行情感分析，基于Keras框架，实现了全连接、CNN、RNN等神经网络模型，并对准确率进行测试和比较。

实现思路

实现了全连接网络（MLP）作为baseline，文本分别采用bags-of-words和TF-IDF进行表示，标签分别转化为分类问题（classification，标签最大值作为单标签预测）和回归问题（regression，标签归一化）进行表示。

实现了CNN、Text-CNN、LSTM、RNN模型，文本采用word-embedding进行表示，模型中的词嵌入层(embedding layer)分别初始化为预训练的词向量且不可训练(static)、初始化为预训练的词向量且可训练(non-static)、随机初始化且可训练(rand)，标签分别转化为分类问题和回归问题进行表示。

工作流程可分为以下步骤：

1. 对数据进行预处理，数据包括新浪新闻和预训练的词向量（见 https://github.com/Embedding/Chinese-Word-Vectors, 语料库: Sogou News, 表示形式: Word2vec / Skip-Gram with Negative Sampling (SGNS), 特点: Word + Character + Ngram, 感谢张晨同学上传到清华云盘https://cloud.tsinghua.edu.cn/f/7928cb6c3db34c67b1b0/）。对预训练的词向量，首先建立并存储一个单词到编号的字典，然后将编号到词向量以矩阵的形式存储。对新浪新闻，抽取其中的每条新闻的文本和标签信息，分别转化为矩阵。对文本矩阵，分别转化为bags-of-words、TF-IDF、word-embedding的表示形式并存储。对标签矩阵，分别转化为分类问题和回归问题并存储。
2. 训练神经网络。按照下面的模型结构建立神经网络，读入供训练的文本数据和标签数据，划分为训练集和验证集，开始训练。
3. 预测准确率等指标。读入供测试的文本数据和标签数据，利用模型进行预测，并计算准确率(accuracy)、F-score、相关系数(Correlation Coefficient)

模型结构

MLP（全连接）模型结构如下：

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 64)	2833728
dropout_1 (Dropout)	(None, 64)	0
dense_2 (Dense)	(None, 64)	4160

Layer (type)	Output Shape	Param #
dropout_2 (Dropout)	(None, 64)	0
dense_3 (Dense)	(None, 8)	520

Total params: 2,838,408

CNN模型结构如下：

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 350, 300)	13282800
conv1d_1 (Conv1D)	(None, 348, 128)	115328
max_pooling1d_1 (MaxPooling1D)	(None, 116, 128)	0
dropout_1 (Dropout)	(None, 116, 128)	0
conv1d_2 (Conv1D)	(None, 114, 64)	24640
max_pooling1d_2 (MaxPooling1D)	(None, 38, 64)	0
dropout_2 (Dropout)	(None, 38, 64)	0
conv1d_3 (Conv1D)	(None, 36, 32)	6176
max_pooling1d_3 (MaxPooling1D)	(None, 12, 32)	0
dropout_3 (Dropout)	(None, 12, 32)	0
flatten (Flatten)	(None, 384)	0
dense_1 (Dense)	(None, 128)	49280
dropout_4 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 8)	1032

Total params: 13,479,256

Text-CNN模型结构如下：
参考论文: Kim, Y. (2014). Convolutional Neural Networks for Sentence Classification. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP 2014), 1746–1751. 模型如图Text-CNN所示，不同的是，这里采用6个 region sizes: 2, 3, 4, 5, 6, 7，并且MaxPooling得到的是一个向量而不是图中的一个值。

Layer (type)	Output Shape	Param #	Connected to
input (InputLayer)	(None, 350)	0	
embedding (Embedding)	(None, 350, 300)	13282800	input[0][0]

Layer (type)	Output Shape	Param #	Connected to
conv1d_1 (Conv1D)	(None, 349, 4)	2404	embedding[0][0]
conv1d_2 (Conv1D)	(None, 349, 4)	2404	embedding[0][0]
conv1d_3 (Conv1D)	(None, 348, 4)	3604	embedding[0][0]
conv1d_4 (Conv1D)	(None, 348, 4)	3604	embedding[0][0]
conv1d_5 (Conv1D)	(None, 347, 4)	4804	embedding[0][0]
conv1d_6 (Conv1D)	(None, 347, 4)	4804	embedding[0][0]
conv1d_7 (Conv1D)	(None, 346, 4)	6004	embedding[0][0]
conv1d_8 (Conv1D)	(None, 346, 4)	6004	embedding[0][0]
conv1d_9 (Conv1D)	(None, 345, 4)	7204	embedding[0][0]
conv1d_10 (Conv1D)	(None, 345, 4)	7204	embedding[0][0]
conv1d_11 (Conv1D)	(None, 344, 4)	8404	embedding[0][0]
conv1d_12 (Conv1D)	(None, 344, 4)	8404	embedding[0][0]
max_pooling1d_1 (MaxPooling1D)	(None, 116, 4)	0	conv1d_1[0][0]
max_pooling1d_2 (MaxPooling1D)	(None, 116, 4)	0	conv1d_2[0][0]
max_pooling1d_3 (MaxPooling1D)	(None, 116, 4)	0	conv1d_3[0][0]
max_pooling1d_4 (MaxPooling1D)	(None, 116, 4)	0	conv1d_4[0][0]
max_pooling1d_5 (MaxPooling1D)	(None, 115, 4)	0	conv1d_5[0][0]

Layer (type)	Output Shape	Param #	Connected to
max_pooling1d_6 (MaxPooling1D)	(None, 115, 4)	0	conv1d_6[0][0]
max_pooling1d_7 (MaxPooling1D)	(None, 115, 4)	0	conv1d_7[0][0]
max_pooling1d_8 (MaxPooling1D)	(None, 115, 4)	0	conv1d_8[0][0]
max_pooling1d_9 (MaxPooling1D)	(None, 115, 4)	0	conv1d_9[0][0]
max_pooling1d_10 (MaxPooling1D)	(None, 115, 4)	0	conv1d_10[0][0]
max_pooling1d_11 (MaxPooling1D)	(None, 114, 4)	0	conv1d_11[0][0]
max_pooling1d_12 (MaxPooling1D)	(None, 114, 4)	0	conv1d_12[0][0]
dropout_1 (Dropout)	(None, 116, 4)	0	max_pooling1d_1[0][0]
dropout_2 (Dropout)	(None, 116, 4)	0	max_pooling1d_2[0][0]
dropout_3 (Dropout)	(None, 116, 4)	0	max_pooling1d_3[0][0]
dropout_4 (Dropout)	(None, 116, 4)	0	max_pooling1d_4[0][0]
dropout_5 (Dropout)	(None, 115, 4)	0	max_pooling1d_5[0][0]
dropout_6 (Dropout)	(None, 115, 4)	0	max_pooling1d_6[0][0]
dropout_7 (Dropout)	(None, 115, 4)	0	max_pooling1d_7[0][0]
dropout_8 (Dropout)	(None, 115, 4)	0	max_pooling1d_8[0][0]
dropout_9 (Dropout)	(None, 115, 4)	0	max_pooling1d_9[0][0]
dropout_10 (Dropout)	(None, 115, 4)	0	max_pooling1d_10[0][0]

Layer (type)	Output Shape	Param #	Connected to
dropout_11 (Dropout)	(None, 114, 4)	0	max_pooling1d_11[0][0]
dropout_12 (Dropout)	(None, 114, 4)	0	max_pooling1d_12[0][0]
flatten_1 (Flatten)	(None, 464)	0	dropout_1[0][0]
flatten_2 (Flatten)	(None, 464)	0	dropout_2[0][0]
flatten_3 (Flatten)	(None, 464)	0	dropout_3[0][0]
flatten_4 (Flatten)	(None, 464)	0	dropout_4[0][0]
flatten_5 (Flatten)	(None, 460)	0	dropout_5[0][0]
flatten_6 (Flatten)	(None, 460)	0	dropout_6[0][0]
flatten_7 (Flatten)	(None, 460)	0	dropout_7[0][0]
flatten_8 (Flatten)	(None, 460)	0	dropout_8[0][0]
flatten_9 (Flatten)	(None, 460)	0	dropout_9[0][0]
flatten_10 (Flatten)	(None, 460)	0	dropout_10[0][0]
flatten_11 (Flatten)	(None, 456)	0	dropout_11[0][0]
flatten_12 (Flatten)	(None, 456)	0	dropout_12[0][0]
concatenate (Concatenate)	(None, 5528)	0	flatten_1[0][0] flatten_2[0][0] flatten_3[0][0] flatten_4[0][0] flatten_5[0][0] flatten_6[0][0] flatten_7[0][0] flatten_8[0][0] flatten_9[0][0] flatten_10[0][0] flatten_11[0][0] flatten_12[0][0]
dense_1 (Dense)	(None, 128)	707712	concatenate[0][0]
dropout_13 (Dropout)	(None, 128)	0	dense_1[0][0]

Layer (type)	Output Shape	Param #	Connected to
dense_2 (Dense)	(None, 8)	1032	dropout_13[0][0]

Total params: 14,056,392

LSTM模型结构如下：

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 350, 300)	13282800
lstm (LSTM)	(None, 128)	219648
dropout (Dropout)	(None, 128)	0
dense (Dense)	(None, 8)	1032

Total params: 13,503,480

RNN模型结构如下：

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 350, 300)	13282800
simple_rnn (SimpleRNN)	(None, 128)	54912
dropout (Dropout)	(None, 128)	0
dense (Dense)	(None, 8)	1032

Total params: 13,338,744

文本表示形式的转化

文本有3种表示形式：bags-of-words、TF-IDF、word-embedding。这里利用keras内置的Tokenizer类，可以方便地将文本转化为这三种形式。Tokenizer类接收一个文本矩阵，能够生成一个词表，并统计各个词在各个文档中出现的次数等信息，从而给出词频矩阵(bags-of-words)、TF-IDF矩阵、词嵌入矩阵。

标签表示形式的转化

标签有2种表示形式：分类问题、回归问题。分类问题只需将标签中的最大值置为1，其余置为0；回归问题只需将每个标签除以标签总数进行归一化。

测试结果

各类神经网络的测试结果（包括准确率accuracy、F-score、相关系数Correlation Coefficient）如下表所示：

			accuracy	F-score	Correlation Coefficient
MLP	bags-of-words	classification	0.5745	0.2031	0.5819
		regression	0.5700	0.1661	0.5604
	TF-IDF	classification	0.5821	0.2207	0.5891
		regression	0.5467	0.1586	0.5367
CNN	static	classification	0.5821	0.2265	0.6104
		regression	0.5736	0.1705	0.5880
	non-static	classification	0.5606	0.2029	0.5855
		regression	0.5709	0.2004	0.6021
	random	classification	0.5413	0.1577	0.5168
		regression	0.5180	0.1948	0.5486
Text-CNN	static	classification	0.5404	0.1995	0.5564
		regression	0.5700	0.1879	0.5782
	non-static	classification	0.5512	0.2150	0.5617
		regression	0.5489	0.1695	0.5621
	random	classification	0.5373	0.1614	0.5325
		regression	0.5561	0.1806	0.5883
LSTM	static	classification	0.5494	0.2367	0.5627
		regression	0.5377	0.1588	0.5554
	non-static	classification	0.4928	0.2413	0.5349
		regression	0.5382	0.2158	0.5758
	random	classification	0.4879	0.1007	0.4791
		regression	0.4987	0.1304	0.4936
RNN	static	classification	0.5180	0.1618	0.5001
		regression	0.5058	0.1590	0.4923
	non-static	classification	0.4825	0.1515	0.4732
		regression	0.4538	0.1433	0.4322
	random	classification	0.4681	0.0913	0.4277
		regression	0.4776	0.0808	0.4578

从这些测试结果可以看出：

- 全连接网络(MLP)和CNN的效果最好，其次是Text-CNN和LSTM，最后是RNN。全连接网络和CNN的较高准确率大约在56-58%，Text-CNN和LSTM的较高准确率大约在53-55%，而RNN的较高准确率大约在50-51%。这可能是因为LSTM和RNN的模型实现较为简单，影响了准确率，而全连接网络的结构较为简单，参数容易得到充分训练；也可能是因为数据集的规模较小、分布较不均匀，导致模型训练出现偏差。
- 分类问题相较于回归问题，训练得到的模型的准确率一般更高，而相关系数一般更低。这是因为分类问题采用交叉熵作为损失函数，目标是标签最大值尽可能正确，因此由单标签计算得到的准确率更高；回归问题采用均方误差作为损失函数，目标是标签分布尽可能正确，因此相关系数更高。
- 作为词嵌入层，预训练的词向量比随机的词向量(rand)效果更好，而预训练的词向量是否可再训练对结果影响不大，甚至不可再训练(static)的效果更好一些。预训练的词向量比随机的词向量效果更好是符合预期的，因为从随机词向量开始训练，参数数量很大，容易导致词嵌入层权值变化不大或者陷入极小值，效果不好。而同样是预训练的词向量，不可再训练比可再训练(non-static)的效果更好一些，这是不符合预期的，我原以为可再训练能够使得词向量更“适应”该语料库。我认为这可能是由于语料库的规模太小，导致训练和测试不充分。
- 在全连接网络(MLP)中，用bags-of-words或TF-IDF表示文本对效果影响不大。这可能是因为bags-of-words表示中，较为“中性”的词对应的权值接近于0，接近于TF-IDF减少高频停用词的效果，因此二者结果接近。

参数调整

优化器(如Adam、RMSProp)参数均采用默认值，默认值主要是建议值或原论文中的值，效果一般不错。需要调整的参数主要是“早停法”(见问题思考部分)对应的阈值和耐心值。该参数的调整方法为，不使用“早停法”，训练模型足够多个epoch，观察验证集上损失函数的变化。一般来说，验证集上损失函数大致呈先下降后上升的趋势，下降过程中一般呈梯度下降，即下降一下后波动一段再下降一下。为了尽可能使训练不在波动段停止，同时加快模型的训练速度，需要将耐心值设为稍大于观察到的连续波动的epoch数，将阈值设为稍小于损失函数下降接近最低点时的下降值。这样就可以通过实践得到适合的“早停法”阈值和耐心值。

问题思考

1. 我采用“早停法”即验证集调整的方式停止实验训练。首先我将提供的训练数据按照4:1的比例划分为训练集和验证集，在训练集上训练神经网络模型，在验证集上计算损失函数和准确率（但不用于训练模型）。我设定了一个阈值和一个耐心值，若在耐心值个训练周期内，验证集上损失函数下降不超过阈值（甚至是上升）时，训练停止。最后我将模型权值恢复为验证集上损失函数最小时的模型权值（而不是直接采用停止训练时的模型权值）。
   设定的阈值和耐心值与采用的神经网络模型和标签的表示方式有关。比如全连接网络，将标签转化为单标签预测（分类问题），阈值为0.001，耐心值为5；将标签归一化（回归问题），阈值为0.0003，耐心值为5。阈值不同是因为分类问题的损失函数为交叉熵，回归问题的损失函数为均方误差，交叉熵的减小相对均方误差更快，因此其阈值较大。耐心值是因为损失函数的下降呈“梯度”，即下降一下后波动一段再下降一下，根据实验观察取耐心值为5，能够较好的避免模型训练在损失函数未达到最小值时停止，同时避免训练过度。而lstm神经网络，标签为分类问题，阈值为0.005，耐心值为3，相较于全连接网络而言阈值更大、耐心值更小，这是因为lstm根据实验观察损失函数下降很快，并且在下降至最低点后迅速上升，因此更大的阈值和更小的耐心值能够迅速结束训练。
   相较于固定迭代次数，通过验证集调整能够自动判断何时停止，有效减缓了过拟合问题，并减少了

训练时间。而且由于何时停止是自动选择的，减少了调参的压力。但是由于验证集的损失函数有波动，可能存在由于波动导致判停条件提前成立导致“过早停止”的问题。

2. 神经网络各层参数的初始化遵循默认值。比如词嵌入层初始化为random-uniform，卷积层和全连接层权值初始化为Glorot-uniform(Xavier-uniform)，偏置初始化为全零。我认为随机初始化权值即可，避免模型训练陷入损失函数的极小值。
3. 我采用两种防止过拟合，一个是“早停法”，一个是Dropout层。“早停法”在验证集的损失函数一段时间内没有提升时停止训练，Dropout层随机舍弃该层的一部分神经元，二者都能有效避免过拟合。
4. 根据本次实验结果，全连接神经网络实现简单，训练较快，效果较好；CNN和text-CNN收敛速度较快，但每个epoch的训练时间较长，效果很好；RNN和LSTM收敛速度很快，但本次试验效果较差，理论上RNN非常适合变长序列的分析，可能是因为训练集的规模较小，参数没有充分调整。

心得体会

本次实验中，我通过全连接、CNN、text-CNN、RNN、LSTM等神经网络的实现，对各类神经网络的结构有了更深入的了解，学习了神经网络的调参。神经网络的确有些“黑箱”，但是效果也很好，适合应用于实践。

未来拓展

- 实现更复杂的模型，并进行比较。本次实验中全连接网络、RNN、LSTM等神经网络的实现较为简单，可能影响了其准确率，未来希望对更复杂的模型进行实验，进一步提升准确率。
- 对更多参数进行调整。本次实验只调整了早停法的参数（阈值和耐心值），优化器（如Adam、RMSProp）的参数、神经网络的初始化方式等均采用默认值，神经元的个数也没有经过调整和比较，可能影响了结果的准确率。未来希望对这些参数进行调整并实验，进一步提升准确率。
- 更深入地了解神经网络。由于Keras对神经网络的封装较好，很多时候我都只将神经网络作为一个“黑盒”看待，这的确方便了我从整体上思考神经网络结构等问题，但也让我对神经网络的内部实现和各类参数等知识缺乏了解。我希望未来能够更深入地了解神经网络的实现，从而更好地调整参数和对神经网络进行创新。