机器学习纳米学位

毕业项目—Quora 句子相似度匹配

Stephen

2019 年 4 月 18 日

I. 问题的定义

项目概述

除了 Quora，物理学家可以帮助厨师解决数学问题并获得烹饪技巧吗？ Quora 是一个获取和分享知识的地方。这是一个提出问题并与提供独特见解和质量答案的人联系的平台。这使人们能够相互学习，更好地了解世界。

每个月有超过 1 亿人访问 Quora，因此很多人提出类似措辞的问题也就不足为奇了。具有相同意图的多个问题可能会导致寻求者花更多时间找到问题的最佳答案，并使作者觉得他们需要回答同一问题的多个版本。 Quora 重视规范性问题，因为它们为活跃的求职者和作家提供了更好的体验，并且从长远来看为这两个群体提供了更多价值。

目前，Quora 使用随机森林模型来识别重复的问题。现在要面临的是，通过应用先进技术来分类问题对是否重复来解决这种自然语言处理问题的挑战。这样做可以更容易地找到问题的高质量答案，从而改善 Quora 作家，求职者和读者的体验。

此项目基于深度学习和自然语言处理相关的技术来判断句子是否相似。

Kaggle 项目：https://www.kaggle.com/c/quora-question-pairs

问题陈述

问题：使用深度学习方法判断两句话是否代表同一个含义。

输入：两句话

输出：相似还是不相似

在这个项目中，该问题是一个二分类问题。作者将构建一个卷积神经网络，使它读入两句话并输出这两句话是否相似的概率[0,1]

评价指标

评估标准为 $LogLoss$，使用 kaggle 官方的二分类 $LogLoss$ 公式：

$$\text{LogLoss} = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log(\hat{y_i}) + (1 - y_i)\log(1 - \hat{y_i})]$$

其中

- n 是测试集中的所有句子对数量
- $\hat{y_i}$是句子相似的预测概率
- y_i：如果两句话相似，则为 1;如果不相似，则为 0
- log()是自然对数 e

较小的 log loss 更好

II. 分析

数据的探索

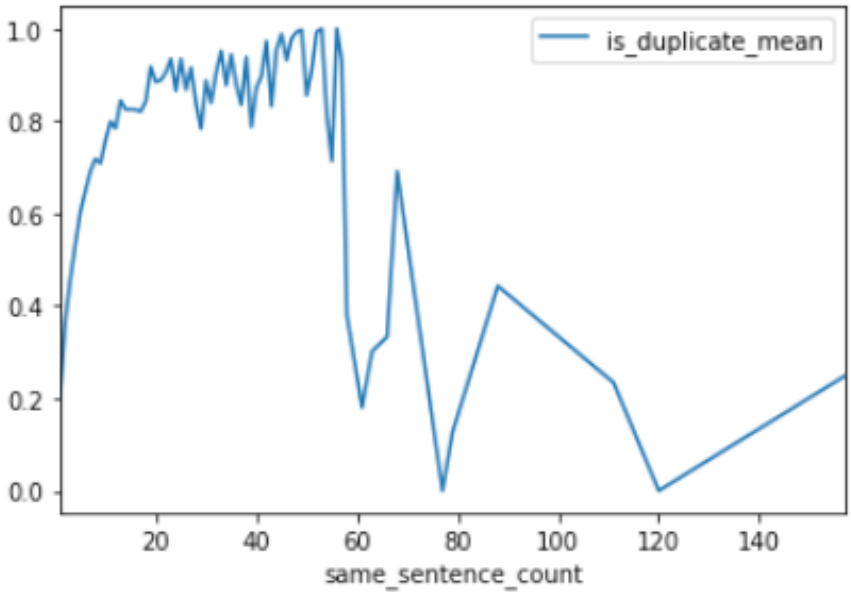
项目数据集可以从 kaggle 上下载。

此训练集共有 404267 对带标签的句子对，相似的句子为 149262 条，占比 36.92%。测试集共有 2345796 条不带标签的句子对。

经过一些系列的探索(探索方式在 data_explore.html 里)，得出以下结果：

训练集数据量	404267 对
测试集数据量	2345796 对
其中相似的句子数据量	149262 对，占比 36.92
其中不相似的句子数据量	255005 对，占比 63.08
训练集词表数量	95588 个
测试集词表数量	101312 个
测试集中有多少词汇不在训练集里	41448 个
训练集句子平均长度	11.156341
测试集句子平均长度	11.123814

句子出现频次 与 is_duplicate 的线性相关性



数据集介绍

1，训练集

字段名	描述
id	一对句子的唯一标识符
qid1	第 1 句句子的唯一标识符
qid2	第 2 句句子的唯一标识符
question1	第 1 句句子
question2	第 2 句句子
is_duplicate	是否重复 （0 为不重复，1 为重复）

2，测试集

字段名	描述
test_id	一对句子的唯一标识符
question1	第 1 句句子
question2	第 2 句句子

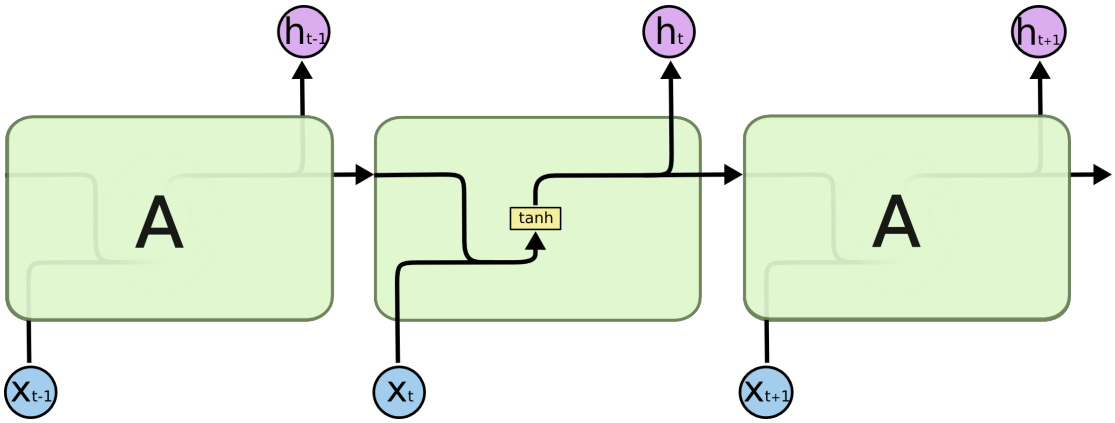
另外需要将数据集划分出训练集和验证集，比例定为 9：1。

算法和技术

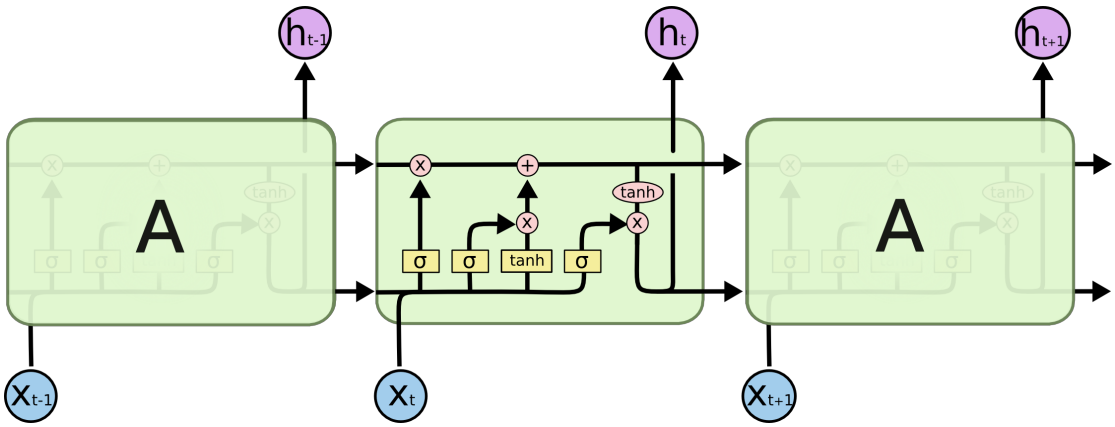
句子相似匹配是一个二元分类的问题。伴随着数据集的增加和计算能力的提升，特别是 GPU 计算能力的爆发性提升，使用卷积神经网络的深度学习便很好的解决了图片识别的问题。本次的项目中将采用 tensorflow 和 keras 来搭建学习模型，快速简单易上手。

RNN(Recurrent Neural Network 循环神经网络) 是一类用于处理序列数据的神经网络。首先我们要明确什么是序列数据，摘取百度百科词条：时间序列数据是指在不同时间点上收集到的数据，这类数据反映了某一事物、现象等随时间的变化状态或程度。这是时间序列数据的定义，当然这里也可以不是时间，比如文字序列，但总归序列数据有一个特点——后面的数据跟前面的数据有关系。

LSTM(long short-term memory)长短期记忆网络是 RNN 的一种变体，RNN 由于梯度消失的原因只能有短期记忆，LSTM 网络通过精妙的门控制将短期记忆与长期记忆结合起来，并且一定程度上解决了梯度消失的问题。



标准 RNN 中的重复模块包含单一的层



LSTM 中的重复模块包含四个交互的层

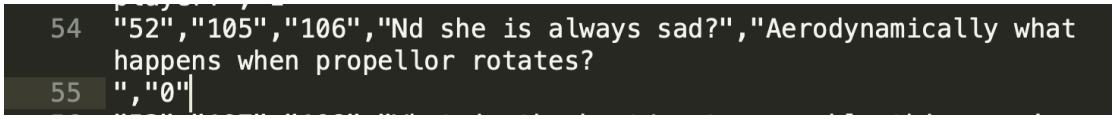
常用的优化器有随机梯度下降 SGD 及其优化算法 Adagrad、Adadelta、Adam 等。SGD 每次只随机选择一个样本来更新模型参数，因此每次的学习是非常快速的，并且可以进行在线更新。 但是 SGD 最大的缺点在于每次更新可能并不会按照正确的方向进行，因此可以带来优化波动(扰动)。不过从另一个方面来看，SGD 所带来的波动有个好处就是，对于类似盆地区域（即很多局部极小值点）那么这个波动的特点可能会使得优化的方向从当前的局部极小值点跳到另一个更好的局部极小值点，这样便可能对于非凸函数，最终收敛于一个较好的局部极值点，甚至全局极值点。 由于波动，因此会使得迭代次数（学习次数）增多，即收敛速度变慢。不过最终其会和全量梯度下降算法一样，具有相同的收敛性，即凸函数收敛于全局极值点，非凸损失函数收敛于局部极值点。

Adagrad 是对学习率进行了一个约束，自适应地为各个参数分配不同学习率的算法，适合处理稀疏梯度。缺点是其学习率是单调递减的，训练后期学习率非常小，需要手工设置一个全局的初始学习率等。Adadelta 是对 Adagrad 的扩展，能对学习率进行自适应约束，又进行了计算上的简化，训练时的加速效果不错，时间也很快，有效地克服 Adagrad 学习率收敛至零的缺点。通过比较，Adadelta 算法的表现效果通常不错，学习率自适应优化，不用手动调节。最后使用 Dropout 来防止过拟合，因为随机的丢弃一部分隐藏节点既加快了计算又减弱了神经元节点间的联合适应性，增强了泛化能力。

III. 方法

数据预处理

一， 手工处理



train.csv 和 test.csv 有少量类似这样的数据，明明是一行数据，被拆分成了 2 行，需要将这些合并成一行。

二，句子格式化

将一些短语的多种写法，统一成一种，比如 won't 改成 will not，这里参考已有的解决方案：https://www.kaggle.com/currie32/the-importance-of-cleaning-text

```
def text_to_wordlist(text, remove_stop_words=True, stem_words=False):
    # Clean the text, with the option to remove stop_words and to stem words.

    # Clean the text
    text = re.sub(r"[^A-Za-z0-9]", " ", text)
    text = re.sub(r"what's", "", text)
    text = re.sub(r"What's", "", text)
    text = re.sub(r"\'s", " ", text)
    text = re.sub(r"\'ve", " have ", text)
    text = re.sub(r"can't", "cannot ", text)
    text = re.sub(r"n't", " not ", text)
    text = re.sub(r"I'm", "I am", text)
    text = re.sub(r" m ", " am ", text)
    text = re.sub(r"\'re", " are ", text)
    text = re.sub(r"\'d", " would ", text)
    text = re.sub(r"\'ll", " will ", text)
    text = re.sub(r"60k", " 60000 ", text)
    text = re.sub(r" e g ", " eg ", text)
    text = re.sub(r" b g ", " bg ", text)
    text = re.sub(r"\0s", "0", text)
    text = re.sub(r" 9 11 ", "911", text)
    text = re.sub(r"e-mail", "email", text)
```

使用 GloVe 模型**训练词**向量

这里用到了 Glove 模型来生成 Embedding 层，相关论文：https://nlp.stanford.edu/pubs/glove.pdf

- 模型目标：进行词的向量化表示，使得向量之间尽可能多地蕴含语义和语法的信息。
- 输入：语料库
- 输出：词向量
- 方法概述：首先基于语料库构建词的共现矩阵，然后基于共现矩阵和 GloVe 模型学习词向量。

GLOVE 的模型具有下面几个优点：

a. 训练效率高

b. 计算规模和语料库大小成正比

c. 当语料库小或者向量维度小的时候，依然能取得不错的效果

其他预训练词向量介绍

word2vec

word2vec 来源于 2013 年的论文《Efficient Estimation of Word Representation in Vector Space》，它的核心思想是通过词的上下文得到词的向量化表示，有两种方法：CBOW（通过附近词预测中心词）、Skip-gram（通过中心词预测附近的词）：

ELMo

ELMo 来自于论文《Deep contextualized word representations》，它的官网有开源的工具：https://allennlp.org/elmo

word2vec 和 glove 存在一个问题，词在不同的语境下其实有不同的含义，而这两个模型词在不同语境下的向量表示是相同的，Elmo 就是针对这一点进行了优化，作者认为 ELMo 有两个优势：

1、能够学习到单词用法的复杂特性
2、学习到这些复杂用法在不同上下文的变化
针对点 1，作者是通过多层的 stack LSTM 去学习词的复杂用法，论文中的实验验证了作者的想法，不同层的 output 可以获得不同层次的词法特征。

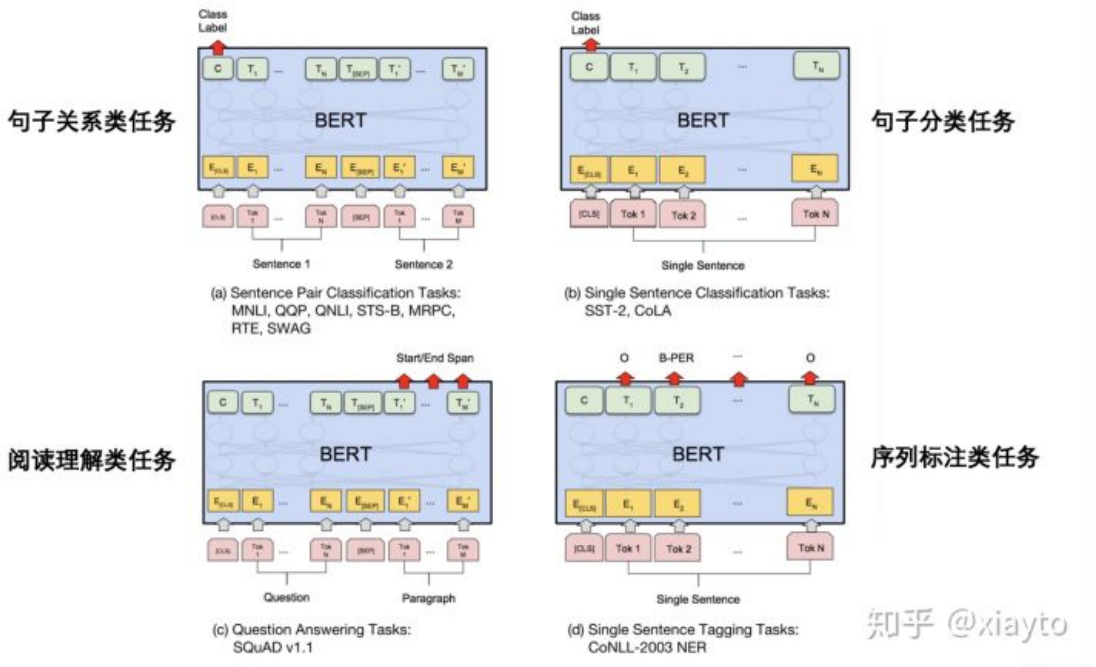
针对点 2，作者通过 pre-train+fine tuning 的方式实现，先在大语料库上进行 pre-train，再在下游任务的语料库上进行 fine tuning。

Bert

它来自于 googole 发表的论文《BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding》

NLP 一共有 4 大类的任务：

序列标注：分词 / 词性标注 / 命名实体识别...
分类任务：文本分类 / 情感分析...
句子关系判断：自然语言推理 / 深度文本匹配 / 问答系统...
生成式任务：机器翻译 / 文本摘要生成...
BERT 为这 4 大类任务的前 3 个都设计了简单至极的下游接口，省去了各种花哨的 Attention、stack 等复杂的网络结构，且实验效果全面取得了大幅度的提升。



上图是 BERT 论文中为下游任务设计的接口，可以先将 BERT 看作是一个黑盒，它跟 ELMo 的工作方式是类似的，都是现在大规模的语料库中 pre-train，然后将下游任务输入，进行比较轻量级的 fine-tuning。下游任务只要做一些轻微的改造就可以放入 BERT 的模型中 fine-tuning，可以看到对于句子关系类的任务，只要加上句子起始和结束的符号，句子之间加入分割符号，然后经过 BERT 模型它最后的一个位置的输出连接上一个 softmax 的分类器就可以了。对于序列标注的模型，也是只要加入起始和结束的符号，对于最后 BERT 每个位置的输出都加入一个线性的分类器就可以了。

TextCNN

在 2014 年提出，他使用了卷积 + 最大池化这两个在图像领域非常成功的好基友组合。我们先看一下他的结构。如下图所示，示意图中第一层输入为 7*5 的词向量矩阵，其中词向量维度为 5，句子长度为 7，然后第二层使用了 3 组宽度分别为 2、3、4 的卷积核，图中每种宽度的卷积核使用了两个。

其中每个卷积核在整个句子长度上滑动，得到 n 个激活值，图中卷积核滑动的过程中没有使用 padding，因此宽度为 4 的卷积核在长度为 7 的句子上滑动得到 4 个特征值。然后出场的就是卷积的好基友全局池化了，每一个卷积核输出的特征值列向量通过在整个句子长度上取最大值得到了 6 个特征值组成的 feature map 来供后级分类器作为分类的依据。

执行过程

1. 导出特征向量

先将句子转成特征向量，这里用到了 keras 提供的预处理包 keras.preproceing 下的 text 与序列处理模块 sequence 模块

（1）先将单词转换成数字向量

sequences_1 = text_to_word_sequence(text,fileter)

（2）将单词向量，填充到一个固定长度的序列中，这里长度为 30

data_1 = pad_sequences(sequences_1, maxlen=30)

（3）初始化语料库

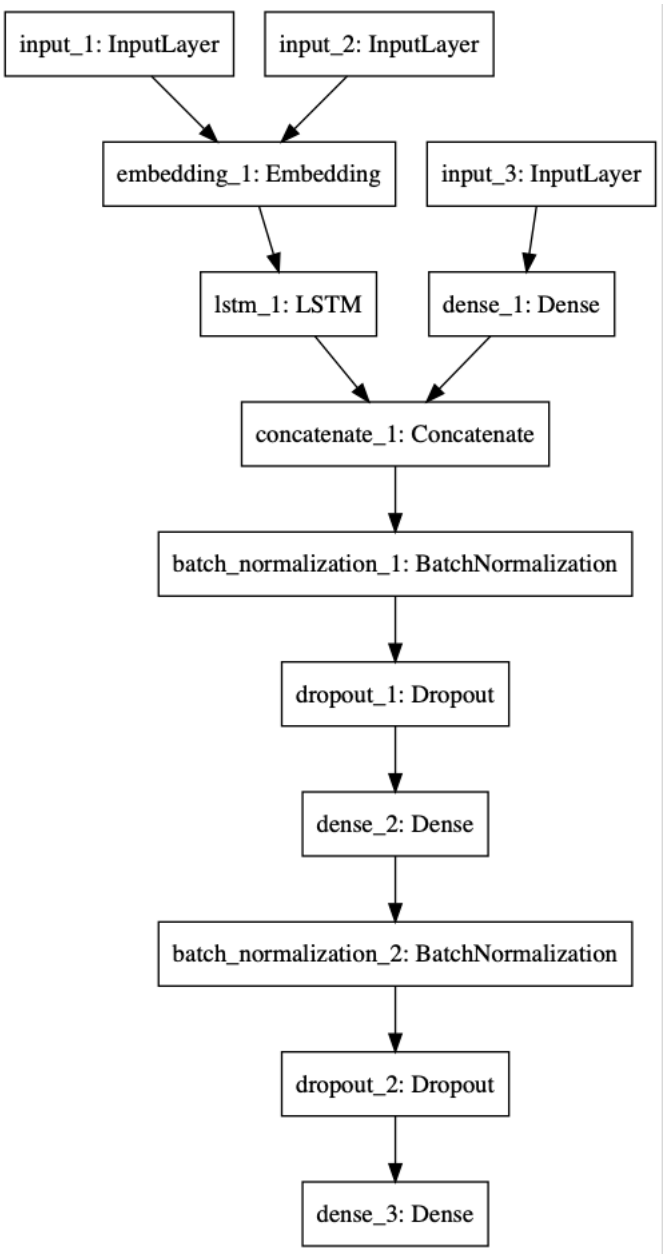
（4）生成 Word Embedding

2，数据调整与切分

训练和验证，比例 9：1

2. 构建模型

调用 Keras 的 API 构建模型


input_1: InputLayer
input_2: InputLayer
embedding_1: Embedding
input_3: InputLayer
lstm_1: LSTM
dense_1: Dense
concatenate_1: Concatenate
batch_normalization_1: BatchNormalization
dropout_1: Dropout
dense_2: Dense
batch_normalization_2: BatchNormalization
dropout_2: Dropout
dense_3: Dense

3. 开始训练模型

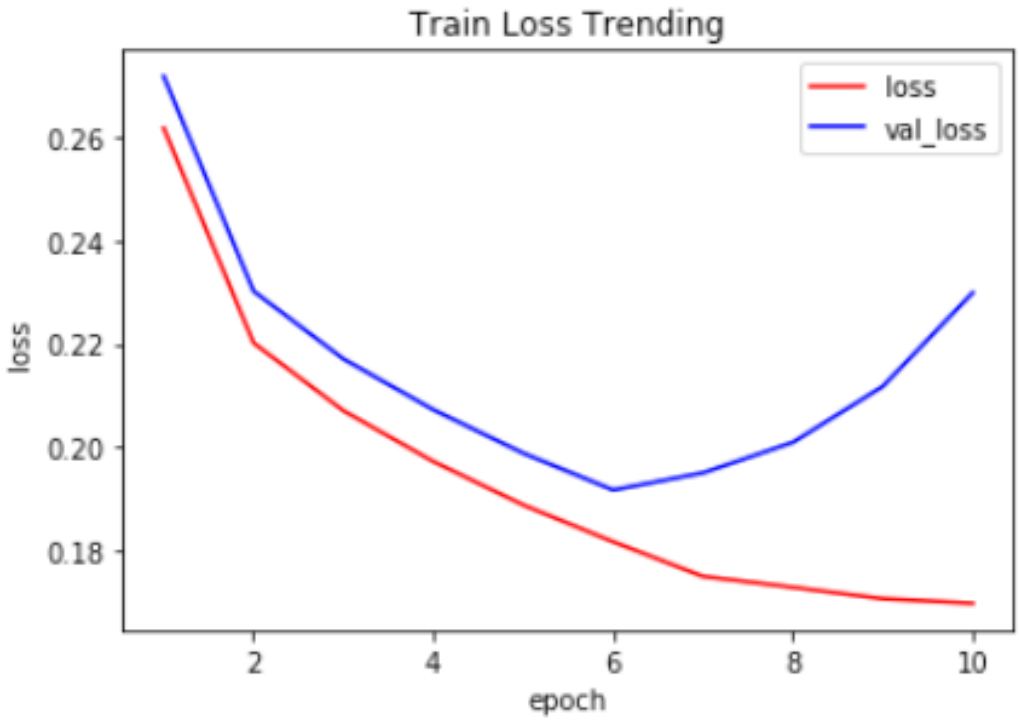
```
]: #开始训练
early_stopping = EarlyStopping(monitor='val_loss', patience=3)
best_model_path = d_result_dir + '/best_model.h5'
model_checkpoint = ModelCheckpoint(best_model_path, save_best_only=True, save_weights_only=True)
hist = model.fit([data_1_train, data_2_train, leaks_train], labels_train,
                 validation_data=([data_1_val, data_2_val, leaks_val], labels_val, validation_weight),
                 epochs=200, batch_size=4096, shuffle=True,
                 callbacks=[early_stopping, model_checkpoint],
                 class_weight=model_class_weight, verbose=2)

model.load_weights(best_model_path)
best_val_score = min(hist.history['val_loss'])
print("best_val_score:%f" % best_val_score)
```

```
Train on 727680 samples, validate on 80854 samples
Epoch 1/200
 - 156s - loss: 0.2917 - acc: 0.8189 - val_loss: 0.2400 - val_acc: 0.8359
Epoch 2/200
 - 153s - loss: 0.2402 - acc: 0.8359 - val_loss: 0.2272 - val_acc: 0.8417
Epoch 3/200
 - 153s - loss: 0.2271 - acc: 0.8404 - val_loss: 0.2187 - val_acc: 0.8491
Epoch 4/200
 - 153s - loss: 0.2173 - acc: 0.8452 - val_loss: 0.2135 - val_acc: 0.8566
Epoch 5/200
 - 153s - loss: 0.2089 - acc: 0.8501 - val_loss: 0.2037 - val_acc: 0.8585
Epoch 6/200
 - 153s - loss: 0.2018 - acc: 0.8546 - val_loss: 0.1997 - val_acc: 0.8605
```

用验证集防止过拟合，并设置早停策略（最多忍受 3 次 loss 没有进步）和最佳模型保存方法

收敛图如下：



4. 预测测试集

对测试集进行预测，导出 csv，然后上传到 kaggle 相关页面查看得分

得分：

submission_ft.csv 10 days ago by xho22 submission ft	0.30912	0.30254

完善

上面的模型，第一次跑完之后，上传 kaggle loss 得分仅有 0.3 左右，离要求的 0.18267 还有很大距离，接下去要做的就是要加上各种优化手段，以下这些是我所用到的。

1，增加弱特征（其他弱特征）

现在只用到了句子主要的特征，这里增加一些算交集，算频率的一些额外属性，作为弱特征

2，开始调整模型参数

我在代码里参数化了很多东西，比如 dense 层 unit 数量，LSTM 层的 unit 数量，Dropout 数量，优化器种类，学习率等，进行网格训练，最终得出一个较好的参数为：

```
lstm_units = 193
dense_units = 136
lstm_dropout = 0.19
dense_dropout = 0.18
leaks_dense_units = 68
optimizer = 'nadam'
```

3，重新分配标签的权重

这里参考了

https://github.com/howardyclo/Kaggle-Quora-Question-Pairs#class-label-reweighting

训练集标签为 1 的概率是 37%，测试集标签为 1 的概率是 17%，也就是说，可能会在验证集和测试集上获得不同的评估分数。为了解决这个问题，我在训练模型时分配了不同的标签权重

```
'''
标签重新分配权重
这里的值参考
https://github.com/howardyclo/Kaggle-Quora-Question-Pairs#class-label-reweighting
0.174/0.369 = 0.471544715
(1-0.174/(1-0.369) = 1.30903328
'''
validation_weight = np.ones(len(labels_val))
validation_weight *= 0.471544715
validation_weight[labels_val == 0] = 1.30903328
model_class_weight = {0: 1.30903328, 1: 0.471544715}
```

4，对预测数据进行微调

输入层，需要将 data_1 和 data_2，双向 stack 一下，为了解决数据对称性问题，并且在预测时，取平均数

该模型的最终测试数据得分为 0.18219。排名 657（top 19.86%），比之前的结果又有了巨大提升。

Submission and Description	Private Score	Public Score
submission.csv a day ago by xho22 add submission details	0.18219	0.18131

IV. 其他尝试

除了本文的以 LSTM 为中心的神经网络之外，还尝试过如下方法

1， 利用 Bert 生成 Word Embedding，在套用神经网络

2， 利用 Bert 的 Fine tuning 直接预测

训练的太慢了，光 Word Embedding 跑了 40 个小时，而且每句话向量达到了 768 个，需要一个很好的电脑和显卡才能完成。

需要作出的改进

首先，数据清洗比较欠缺，找到一些项目做特征工程，有非常细腻的过滤数据方法。有如下两个改进点：

1，图结构的特征提取对于该任务来说也是有非常大的提升的，具体你可以参考：

https://www.kaggle.com/tour1st/magic-feature-v2-0-045-gain

2， 主题模型这块，你可以加入一下这些特征：

使用预训练的 tf-idf vectorizer 分别转化两个问题，然后使用 TruncatedSVD/NMF/LDA 得到维度各为 20-50 的特征集

其次，没有用到集成方法，比如 stack 堆砌模型，主要还是有点偷懒，想着应该单模型可以搞定。

参考文献

[1] Global Vectors for Word Representation，
https://nlp.stanford.edu/pubs/glove.pdf
[2]Long Short-Term Memory,
https://www.bioinf.jku.at/publications/older/2604.pdf
[3] BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, https://arxiv.org/abs/1810.04805
[4] Convolutional Neural Networks for Sentence Classification
https://arxiv.org/abs/1408.5882

参考资料

- https://github.com/howardyclo/Kaggle-Quora-Question-Pairs
- https://github.com/HouJP/kaggle-quora-question-pairs
- https://github.com/qqgeogor/kaggle-quora-solution-8th
- https://github.com/drc10723/bert_quora_question_pairs
- https://github.com/google-research/bert