# 自然语言处理系列（7）：TextCNN调参技巧

这篇文章翻译自A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification

近年来，卷积神经网络在句子分类任务上取得了显著的成绩(Kim, 2014;Kalchbrenner et al .,2014)，然而，这些模型要求从业者指定精确的模型结构与模型参数，例如，选择滤波器大小、正则化参数等等。目前尚不清楚对于句子分类的任务，不同的参数设定会对模型性能造成什么样的影响。因此，在这里我们对单层卷积神经网络进行情感分析，探索不同的参数对模型性能的影响；我们的目标是找出对语句分类来说的重要因素和不重要因素。由于一层的CNN结构简单，实验的表现也很好(Kim, 2014)，我们就着重使用这个模型来验证(而不同更复杂的模型)，从我们广泛的实证结果中得到一些了实用的建议，这些结果对于那些有兴趣用CNN对句子分类的人来说很有用。我们的实验结果所证实的一个重要结论是，研究人员应该记录性能差异，因为这可能是由于随机初始化或推理产生的。

# 一、Introduction

在这个工作中，我们关注的是情感分类的重要任务。最近，研究表明，神经网络(CNNs)对这项任务的表现很好(Kim, 2014; Kalchbrenner et al., 2014; Wang et al., 2015; Goldberg, 2015; Iyyer et al., 2015)。这类模型利用词的分布式表示，首先将包含每个句子转换成一个向量，从而形成一个矩阵作为输入给CNN(图1)。实证结果令人印象深刻。这些模型不需要太复杂就能实现强大的结果：例如，Kim(2014)提出了一种直接的单层CNN架构，它可以在多个任务中实现一致的(或类似的)结果。因此，现在有了令人信服的支持，更倾向于使用CNN而不是稀疏线性模型来进行句子分类任务。然而，CNN的一个缺点是，它们要求从业者指定要使用的精确模型架构，并设置超参数。同样的，做出这样的决定似乎是一种黑箱操作，特别是因为在模型中有许多“自由参数”可以探索。这与广泛用于文本分类的线性模型形成了鲜明的对比，例如正则化的逻辑回归和线性支持向量机(SVMs) (Joachims, 1998)。这样的模型特征通常是通过对文本的稀疏表示而产生的，并且需要相对较少的调优:通常只需要设置正则化项的系数(即:模型偏差)。使用训练数据进行线性搜索来确定参数是设置超参数的方法。

最近关于CNN的句子分类的研究，已经提供了用于实现报告结果的设置。然而这些参数设定是通过并非是特定的调参过程。但实际上，搜索CNN的参数空间是极其昂贵的，至少有两个原因:（1）训练这些模型的速度相对较慢，即使使用gpu。例如，在SST-1数据集(Socher et al., 2013)中使用与(Kim, 2014)类似的配置，进行10倍交叉验证，需要1个小时。可能的模型架构和超参数空间是巨大的。例如，我们所讨论的简单的CNN架构，至少需要指定以下内容:输入的词向量表示;滤波器大小;特征图的数量;激活功能;池化策略;dropout比例(如果有的话);和l2范数的系数(如果有的话)。

实际上，对所有这些参数进行调优是不可行的，尤其是考虑到参数估计所需的运行时间。因此，我们的目的是要根据经验来确定那些需要花费精力进行调整的参数，以及那些在性能上无关紧要的，或者在特定的数据集上有“最佳”效果的参数。我们从前人对神经模型的经验分析中得到启发，该模型由Coates et al.(2011)和Breuel (Breuel, 2015)进行，研究了非监督特征学习效果的影响因素，以及随机梯度下降(SGD)超参数对训练的影响。在这里，我们考虑了模型结构的配置和单层CNNs的超参数值对句子分类任务的影响。我们报告了大量实验的结果，探索了不同的模型结构，运行了7个句子分类数据集。

# 二、背景和预备

深度学习方法已在机器学习中得到很好的应用(LeCun et al.， 2015;Bengio,2009)。对于图像和语音处理任务来说，它们尤其成功(也很受欢迎)。然而，最近这些方法已经开始超越传统的自然语言处理(NLP)任务的线性模型(Goldberg, 2015)，这个领域的大部分兴趣都集中在如何得到分布式的词语表达(Bengio et al.， 2003;Mikolov et al.， 2013)并共同将这种“内部”表征嵌入到分类模型中(Col lobert and Weston, 2008;Collobert et al .,2011)或句子建模(Kalchbrenner et al.，2014;Socher et al .,2013)。

在(Kalchbrenner et al.， 2014)中，作者构建了一个包含多个卷积层的CNN架构。他们的模型使用了动态k-max池。他们的模型假定潜在的、密集的、低维度的词向量(在推理之前初始化为随机值)。

Kim(2014)定义了一个更简单的架构，在相同的数据集上实现了类似的结果(Kalchbrenner et al.，2014)。这个模型也将每个单词都表示为一个稠密的、低维的向量(Mikolov et al.， 2013)，他们使用预先训练的词向量，并考虑两种方法:静态和非静态。在前一种方法中，词向量被视为静态输入，而在后一种方法中，则动态调整为特定任务的词向量。

在其他地方，Johnson和Zhang(2014)引入了相似的模型，但改用了高维的one-hot向量表示。他们考虑了这一方法的两种变体，seq-CNN和bow-CNN。前者完全保留了顺序结构(以在非常高维的空间输入空间中操作的代价)，而后者保留了一些序列，但在小区域内丢失了顺序。他们的重点是更长的文本的分类，而不是句子(当然，这个模型也可以用于句子的分类)。Kim的体系结构相对简单——这与Johnson和Zhang(2014)所提出的基本相同，模块化的词向量——再加上在多个数据集上观察到的强大的经验性能，使得这是一个很有吸引力的句子分类方法。然而，在实践中，我们需要做一些模型架构决策和设置各种超参数。目前，很少有经验数据可以指导此类决定;解决这一差距是我们的目标。

## 2.1 CNN

我们首先描述我们在本文中使用的相对简单的CNN架构。我们从一个标记化的句子开始，然后我们将它转换成一个句子矩阵，其中的行根据每个词得到的单词向量。例如，这些可能是谷歌

word2vec (Mikolov et al.， 2013)或GloVe(Pennington et al.， 2014)模型的输出。我们用d表示向量的维数。如果给定句子的长度(即词汇数)是s,然后句子的维数矩阵s×d.接下来，我们可以有效地将句子矩阵作为一个“图像”，通过线性滤波器对它进行卷积操作。在NLP应用中，数据具有固有的顺序结构。直观上，因为行表示离散的符号(即单词)，所以使用宽度等于向量的维数的滤波器是合理的。（比如d)。然后我们可以考虑只改变滤波器的“高度”，它指的是共同考虑的相邻行数(词向量)。从这一点开始，我们将把滤波器的高度称为滤波器的区域大小。

假设有一个滤波器的参数化权向量$w \in R^{h \times d}$和区域大小$h$;$w$包含要估计的$h \cdot d$个参数。我们用$A \in R^{s \times d}$表示句子矩阵,并使用$A[i, j]$代表从第$i$行到第$j$行的子矩阵。卷积算子的输出序列是通过对A的子矩阵进行重复的卷积操作而得到的:

$$o_i = w \cdot A[i : i + h - 1] \ (1)$$

其中，$i = 1....s - h + 1$，.表示子矩阵和滤波器的点积（先对对应元素做乘法，然后求和），输出序列的长度为$s - h + 1$。再加上一个偏置项$b$以及激活函数$f$得到对应的特征图
$c \in R^{s-h+1}$:

$$c_i = f(o_i + b)$$

注意，我们可以使用多个滤波器来实现相同的区域大小，目标是每个滤波器从相同的区域学习互补的特性。也可以指定多个不同区域大小的过滤器(例如:“高度”)。

每个滤波器生成的特征图的维数，正好是句子长度和滤波区域大小的函数。然后，将一个池化函数应用到每个feature map中，以减少需要估计的参数的尺寸和数量。通常，池化操作为1-max池函数(Boureau et al.，2010b)，它从每个feature map生成一个一维特性。或者，可以将池化操作修改为在特征映射中在相同大小的区域内对每个区域对应的显著特征进行编码。每个滤波器映射生成的输出可以被连接到一个“顶部”特征向量，在1-max池的情况下它的大小将独立于单个的句子长度。然后通过一个softmax函数来生成这个表示，以生成最终的分类。在这个softmax层，可以选择应用“dropout策略”(Hinton et al.， 2012)作为正则化方法。这需要在向量中随机设置一些值为0。我们也可以选择施加l2范数约束，当它超过这个值时，将向量的l2范数线性扩展到一个指定的阈值。在训练过程中，最小化的目标是分类的交叉熵损失，估计的参数包括滤波器的权向量(s)、激活函数中的偏置项，以及softmax函数的权向量。请注意，我们可以选择固定词向量(我们将其称为“static”)或作为模型的附加参数，并在模型训练过程中调整(我们将把这种方法称为“non-static”)。我们探索了这两种变体。图1提供了一个简单的示意图，以说明刚刚描述的模型架构。

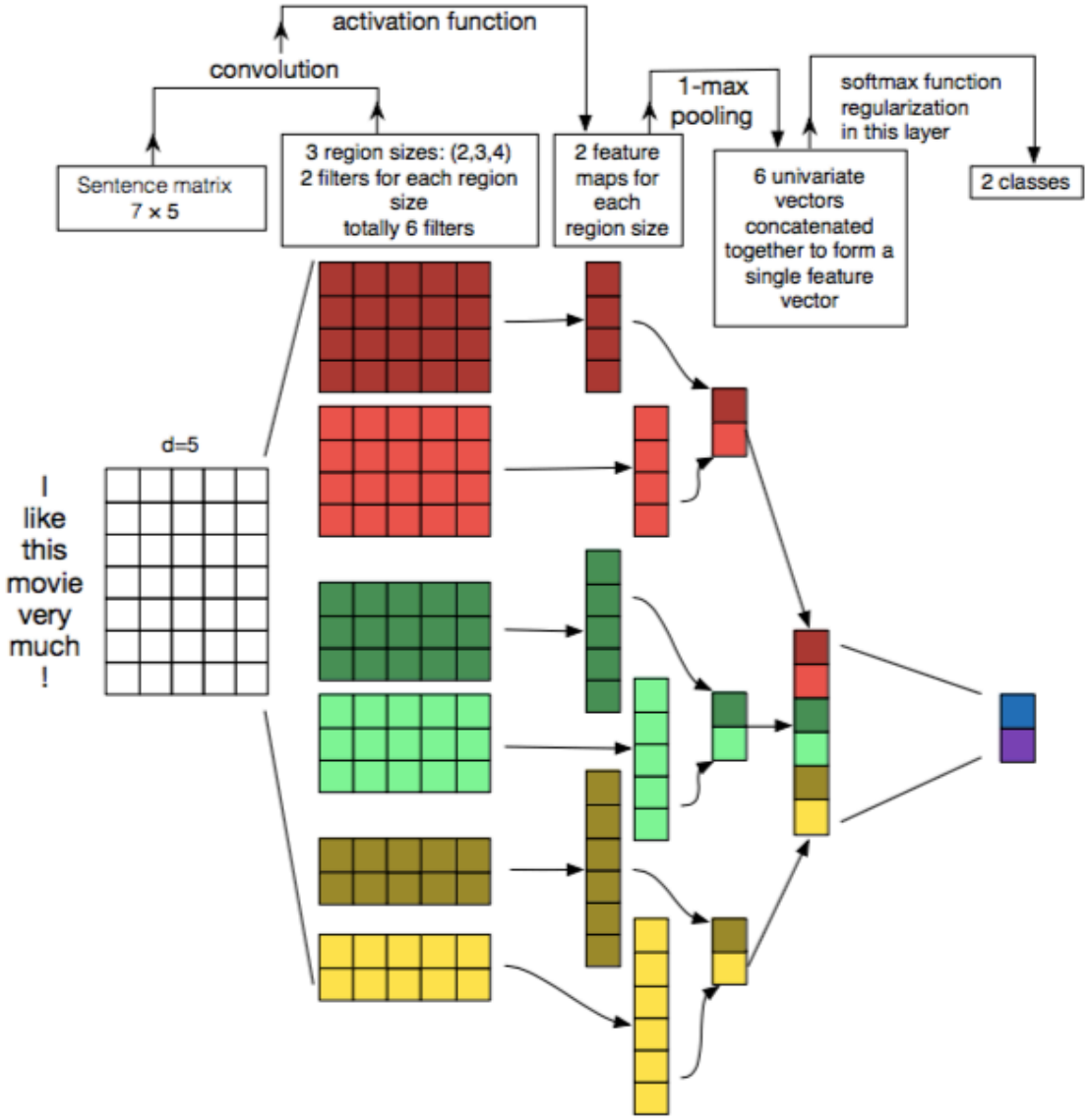


Figure 1: Illustration of a Convolutional Neural Network (CNN) architecture for sentence classification. Here we depict three filter region sizes: 2, 3 and 4, each of which has 2 filters. Every filter performs convolution on the sentence matrix and generates (variable-length) feature maps. Then 1-max pooling is performed over each map, i.e., the largest number from each feature map is recorded. Thus a univariate feature vector is generated from all six maps, and these 6 features are concatenated to form a feature vector for the penultimate layer. The final softmax layer then receives this feature vector as input and uses it to classify the sentence; here we assume binary classification and hence depict two possible output states.

# 三、数据集

我们使用同样的7个数据集(Kim, 2014)，简要总结如下:

- MR:句子极性数据集(Pang and Lee, 2005)。
- SST-1: Stanford Sentiment Treebank (Socher et al.， 2013)。请注意，为了使输入表示在任务中一致，我们只对句子进行训练和测试。与之形成对比的是(Kim, 2014)，在这篇文章中，作者对短语和句子进行了训练。

- SST-2:从SST-1派生而来，但只对两个类进行解析。我们只对句子进行训练和测试，不包括短语。
- Subj:主观性数据集(Pang and Lee, 2005)。
- TREC:问题分类数据集(Li and Roth, 2002)。
- CR:客户审核数据集(Hu and Liu, 2004)。
- MPQA:观点极性数据集(Wiebe et al.， 2005)

在表1中，我们报告了所有七个数据集的平均长度和标记化语句的最大长度。有关这些数据集的更多细节，请参考(Kim, 2014)。

| Dataset | Average length | Maximum length |
|---|---|---|
| MR | 20 | 56 |
| SST-1 | 18 | 53 |
| SST-2 | 19 | 53 |
| Subj | 23 | 120 |
| TREC | 10 | 37 |
| CR | 19 | 105 |
| MPQA | 3 | 36 |

Table 1: Average length and maximum length of the 7 datasets

# 四、baseline模型的性能

为了给CNN的结果提供一个参考点，我们首先报告了使用稀疏正则化SVM进行句子分类的性能。我们使用unigram和bigram特性，只对所有数据集保持最频繁的3万个特征。我们还想通过将信息直接嵌入到这些模型中，来探索实现的相对收益。为此，我们用平均的单词向量(从谷歌word2vec3或GloVe4)来计算这个表达式，并计算出包含句子的单词，类似于(Lai et al.， 2015)中的方法。然后，我们使用RBF-kernel SVM作为在这个稠密特性空间中操作的分类器。我们还尝试将unigram, bi-gram和word2vec作为句子的特征，使用线性支持向量机作为分类器。我们通过嵌套的交叉折叠验证来优化正则化超参数，从而提高了精度。对所有的数据集都进行了十折交叉验证，结果如表2所示。为了保持一致性，我们对之前工作中描述的数据使用相同的预处理步骤(Kim, 2014)。从这些结果中可以立即发现的一件事是，将word2vec输出引入到特征向量中可以实现性能提升。

| Dataset | bow-SVM | wv-SVM | bow+wv-SVM | gv-SVM | bow+gv-SVM |
|---|---|---|---|---|---|
| MR | 78.24 | 78.53 | 79.67 | 78.09 | 80.07 |
| SST-1 | 37.92 | 44.34 | 43.15 | 44.31 | 44.29 |
| SST-2 | 80.54 | 81.97 | 83.30 | 81.92 | 83.52 |
| Subj | 89.13 | 90.94 | 91.74 | 92.61 | 92.22 |
| TREC | 87.95 | 83.61 | 87.33 | 80.28 | 88.71 |
| CR | 80.21 | 80.79 | 81.31 | 81.75 | 82.27 |
| MPQA | 85.38 | 89.27 | 89.70 | 88.97 | 89.27 |

Table 2: Performance of SVM with different feature sets. **bow-SVM** uses only uni- and bi-gram features. **wv-SVM** uses a naive word2vec-based representation, i.e., the average (300-dimensional) word vector, calculated over the words constituting the sentence. **bow-wv-SVM** combines these feature sets by concatenating bow vectors with the average word2vec representations. For completeness, we also report analogous results using GloVe (denoted by **gv**).

# 五、CNN情感分析

我们现在报告的结果来自于我们的主要分析工作，目的是使用CNNs对句子情感分析，作为一个具体的架构和超参数设置的功能。为此，我们以baseline配置(如下所述)作为起点，该配置在之前的工作(Kim, 2014)中表现得很好。然后，我们依次探讨了修改该baseline配置组件的效果，并保持其他设置不变。

我们用“静态”和“非静态”两种词向量来进行实验。在前一种情况下，在训练过程中，单词向量不会被更新，而在后一种情况下，向量会不断调整。非静态配置优于静态配置。因此，本文只报告非静态结果，尽管我们提供了附录中静态配置的结果。

## 5.1 Baseline 参数设置

我们现在考虑CNN的baseline模型配置的性能。具体来说，我们从之前工作中使用的模型架构和超参数开始(Kim, 2014)。为了将由于各种体系结构决策和超参数设置导致的性能差异置于环境中，必须严格评估参数估计过程中的差异。不幸的是，尽管有一个高度随机的推理过程，但大多数之前的工作并没有说明这样的差异。该方差可归因于随机梯度下降(SGD)、随机dropout和随机权值参数初始化的估计。我们表明，通过10倍交叉验证计算的平均性能在重复运行时表现出较高的方差。

我们首先使用表3中描述的原始参数设置，并为每个数据集复制实验100次，其中每一个复制都是一个10倍的CV，并且复制的折叠是固定的。表3中的“ReLU”指的是整流线性单元(Maas et al., 2013)，这是CNN常用的激活函数。我们记录每个重复试验的10折交叉验证的平均精度，并报告超过100次重复试验的平均值、最小值和最大值。我们对静态和非静态方法都这样做。这提供了一种我们可以观察到的不改变模型的方差的感觉。结果如表4所示。图2提供了在所有数据集上对这两种方法的100次重复的平均精度的密度图。为了清晰显示，我们排除了SST-1，因为在这个

数据集上，精度明显降低(但是，结果可以在表中找到)，由于我们对某些数据集进行了不同的分割和处理，正如前面所描述的那样，结果也与原来的不同。因为在这个工作中，我们只关心CNN的每个部分对性能的敏感性和影响，我们不太关心绝对的准确性，也不会比较我们在之前的作品中得到的结果。

| Description | Values |
|---|---|
| input word vectors | Google word2vec |
| filter region size | (3,4,5) |
| feature maps for each region size | 100 |
| activation function | ReLU |
| pooling | 1-max pooling |
| dropout rate | 0.5 |
| $l2$ norm constraint on weight vector | 3 |

Table 3: Baseline configuration.

在确定了CNNs的基准性能之后，我们现在考虑不同架构决策和超参数设置的影响。为此，我们保留所有其他的设置常量(如表3所示)，并且只改变感兴趣的组件。对于我们所考虑的每一个配置，我们重复实验10次，每一次实验都是10折交叉验证。就像原始参数设置的100次重复试验一样，我们也报告了10次10折交叉验证试验的平均均值、最小均值和最大值。对于所有的实验，我们对数据使用与(Kim, 2014)相同的预处理步骤。类似地，我们使用ADADELTA更新规则 (Zeiler, 2012)，并将minibatch大小设置为50。

| Dataset | Non-static Word2vec-CNN | Static Word2vec-CNN |
|---|---|---|
| MR | 81.24 (80.69, 81.56) | 80.66 (80.16, 81.22) |
| SST-1 | 47.08 (46.42,48.01) | 45.54 (45.03,46.27) |
| SST-2 | 85.49 (85.03, 85.90) | 84.84 (84.34,85.20) |
| Subj | 93.20 (92.97, 93.45) | 92.84 (92.56,93.06) |
| TREC | 91.54 (91.15, 91.92) | 90.66 (90.02, 91.18) |
| CR | 83.92 (82.95, 84.56) | 83.57 (82.78, 84.28) |
| MPQA | 89.32 (88.84, 89.73) | 89.57 (89.18, 89.85) |

Table 4: Performance on several datasets using non-static and static word2vec-CNN. In each cell we report: mean (min, max); we will use this format for all tables involving replications.

## 5.2 word2vec

句子分类模型的一个很好的特性是，它以分布式的词语作为输入的形式开始，这是一种灵活的结构，它可以在不同的预先训练的词向量中交换。因此，我们首先探讨了CNNs对所使用的输入表示的句子分类的敏感性。特别地，我们用Glove表示替换谷歌word2vec。谷歌word2vec使用了一个局部上下文窗口模型，从谷歌新闻(Mikolov et al.， 2013)中训练了1000亿单词，而GloVe则提出了一个模型，它利用了一个非常大的语料库(Pennington et al.， 2014)，利用全局单词的联

合作用来统计数据。在本文中，我们使用了一个Glove版本，它是从一个包含8400亿个web数据标记的语料库中训练出来的，并且还有300个维度。我们保留所有其他设置与原始配置相同。我们的报告结果见表5。(请注意，我们还报告了SVM的结果，这些结果在表2中增加了平均Glove向量。)

| Dataset | Non-static GloVe-CNN | Static GloVe-CNN |
|---|---|---|
| MR | 81.03 (80.68,81.48) | 80.10 (79.55,80.51) |
| SST-1 | 45.65 (45.09,45.94) | 44.76 (44.09,45.09) |
| SST-2 | 85.22 (85.04,85.48) | 84.15 (83.94,84.33) |
| Subj | 93.64 (93.51,93.77) | 93.44 (93.28,93.60) |
| TREC | 90.38 (90.19,90.59) | 89.68 (89.26,90.05) |
| CR | 84.33 (84.00,84.67) | 83.50 (82.84,83.98) |
| MPQA | 89.57 (89.31,89.78) | 89.17 (88.98,89.46) |

Table 5: Performance of GloVe-CNN

作为获取对所有数据集最佳性能的潜在简单方法，我们还考虑了一种方法，该方法利用了这两种预先训练出来的表示方法。具体地说，我们将word2vec和Glove向量连接到每个单词，生成了600维的单词向量，我们将它们作为CNN的输入。预训练的向量可能并不总是适用于特定的单词(在word2vec或Glove中，或者两者都有);在这种情况下，我们随机初始化相应的子向量，如上所述。结果见表6。这里我们报告的结果只针对非静态变量，考虑到它的一般优势。

| Dataset | Non-static GloVe+Word2vec CNN |
|---|---|
| MR | 81.02 (80.75,81.32) |
| SST-1 | 45.98 (45.49,46.65) |
| SST-2 | 85.45 (85.03,85.82) |
| Subj | 93.66 (93.39,93.87) |
| TREC | 91.37 (91.13,91.62) |
| CR | 84.65 (84.21,84.96) |
| MPQA | 89.55 (89.22,89.88) |

Table 6: Performance of non-static GloVe + word2vec CNN

从这些结果中可以看出，使用Glove和word2vec时的相对性能取决于数据集，不幸的是，仅仅将这些表示连接起来并不一定有帮助。实际上，当面对一个新的数据集时，很可能需要使用训练数据来尝试不同的预先训练的单词向量。我们也尝试用长、稀疏的one-hot向量作为输入词表示(Johnson and Zhang, 2014)。在这个策略中，每个单词被编码成一个热矢量，它是一个稀疏的高维向量。在这种情况下，句子矩阵的宽度等于词汇量。在训练过程中，一个one-hot向量是固定

的，因为这个方法就像它在一个预构建的字典中搜索每个单词一样。性能如表7所示。

将结果与word2vec和Glove的结果进行比较，我们可以看到在相同的CNN基本配置下，one-hot的性能比word2vec或Glove差。

| Dataset | One-hot vector CNN |
|---|---|
| MR | 77.83 (76.56,78.45) |
| SST-1 | 41.96 (40.29,43.46) |
| SST-2 | 79.80 (78.53,80.52) |
| Subj | 91.14 (90.38,91.53) |
| TREC | 88.28 (87.34,89.30) |
| CR | 78.22 (76.67,80.00) |
| MPQA | 83.94 (82.94,84.31) |

Table 7: Performance of one-hot vector CNN

我们不排除有特定配置的可能性，one-hot的CNN可能会比其他的输入表示的句子分类地更好。但我们这里的证据是，one-hot表示的CNN可能不适合句子分类。这可能是由于稀疏性;这些句子可能过于简短，不足以提供足够的信息来进行这种高维编码(而对于长文档来说，这可能不是一个问题)。

## 5.3 滤波器区域大小

我们首先将区域大小设为1来看看滤波器区域大小的效果，我们将这个区域的feature map的数量设置为100(与原来的配置一样)。我们考虑区域大小为1、3、5、7、10、15、20、25和30，并记录每个区域大小的10倍交叉验证的平均值、最小值和最大精度，并将结果显示在表8中。

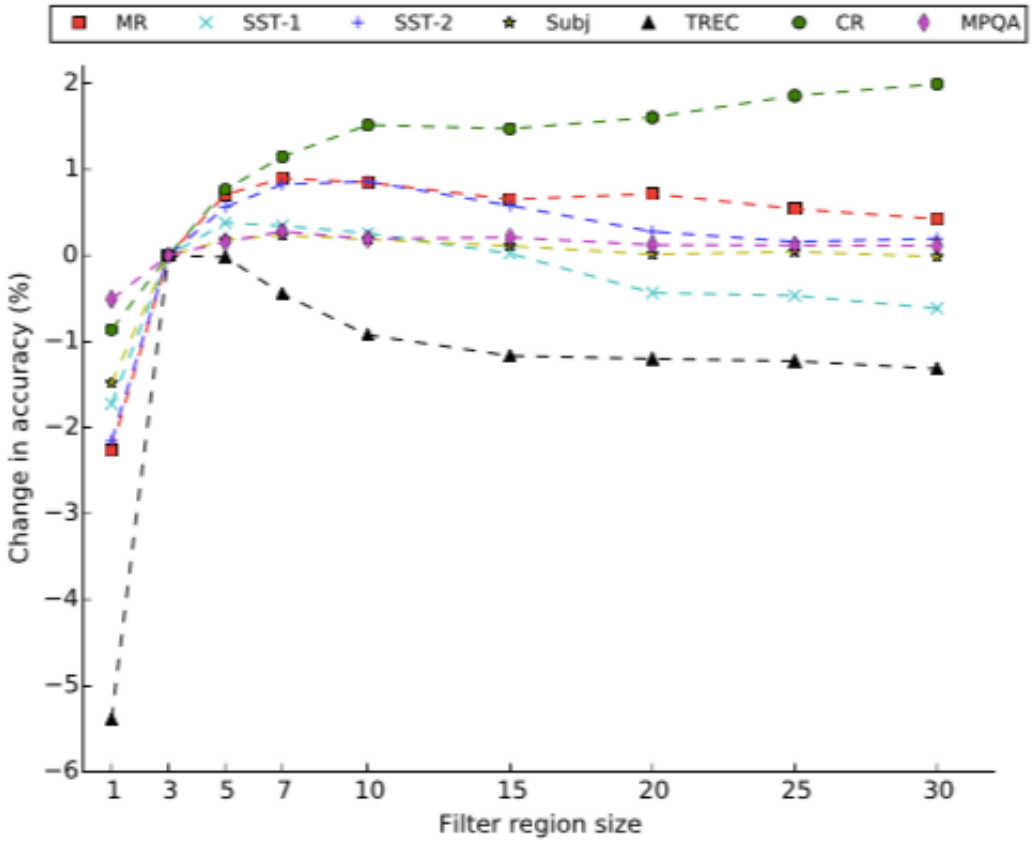


Figure 3: Effect of region size using only one region size in non-static word2vec-CNN

图3显示了每个区域大小和区域大小为3时的10次重复实验的平均精度之间的差异。因为我们只对精确的趋势感兴趣，因为我们改变了CNN的区域大小或其他组件(而不是每个任务的绝对性能)，我们只显示了从任意baseline的精度变化(这里，一个区域大小为3)。我们遵循本公约的所有数据，以方便解释。

从图中可以看出，每个数据集都有自己的最佳滤波区域大小范围。实际上，这表明在指定范围内执行粗网格搜索;这里的数据表明，句子分类的合理范围可能是2到25。然而，对于包含较长句子的数据集，例如CR(最大语句长度为105)，最优区域的大小可能更大。这也可能是由于在CR中，在更大的窗口下，更容易预测正面/负面的客户评论。

| Region size | MR | SST-1 | SST-2 | Subj | TREC | CR | MPQA |
|---|---|---|---|---|---|---|---|
| 1 | 77.85 (77.47,77.97) | 44.91 (44.42,45.37) | 82.59(82.20,82.80) | 91.23 (90.96,91.48) | 85.82 (85.41,86.12) | 80.15 (79.27,80.89) | 88.53 (88.29,88.86) |
| 3 | 80.48 (80.26,80.65) | 46.64 (46.21,47.07) | 84.74 (84.47,85.00) | 92.71 (92.52,92.93) | 91.21 (90.88,91.52) | 81.01 (80.64,81.53) | 89.04 (88.71,89.27) |
| 5 | 81.13 (80.96,81.32) | 47.02 (46.74,47.40) | 85.31 (85.04,85.71) | 92.89 (92.64,93.07) | 91.20 (90.96,91.43) | 81.78 (80.75,82.52) | 89.20 (88.99,89.37) |
| 7 | 81.65 (81.45,81.85) | 46.98 (46.70,47.37) | 85.57 (85.16,85.90) | 92.95 (92.72,93.19) | 90.77 (90.53,91.15) | 82.16 (81.70,82.87) | 89.32 (89.17,89.41) |
| 10 | 81.43 (81.28,81.75) | 46.90 (46.50,47.56) | 85.60 (85.33,85.90) | 92.90 (92.71,93.10) | 90.29 (89.89,90.52) | 82.53 (81.58,82.92) | 89.23 (89.03,89.52) |
| 15 | 81.26 (81.01,81.43) | 46.66 (46.13,47.23) | 85.33 (84.96,85.74) | 92.82 (92.61,92.98) | 90.05 (89.68,90.28) | 82.49 (81.61,83.06) | 89.25 (89.03,89.44) |
| 20 | 81.06 (80.87,81.30) | 46.20 (45.40,46.72) | 85.02 (84.94,85.24) | 92.72(92.47,92.87) | 90.01 (89.84,90.50) | 82.62 (82.16,83.03) | 89.16 (88.92,89.28) |
| 25 | 80.91 (80.73,81.10) | 46.17 (45.20,46.92) | 84.91 (84.49,85.39) | 92.75 (92.45,92.96) | 89.99 (89.66,90.40) | 82.87 (82.21,83.45) | 89.16 (88.99,89.45) |
| 30 | 80.91 (80.72,81.05) | 46.02 (45.21,46.54) | 84.94 (84.63,85.25) | 92.70 (92.50,92.90) | 89.90 (89.58,90.13) | 83.01 (82.44,83.38) | 89.15 (88.93,89.41) |

Table 8: Effect of single filter region size using non-static CNN.

我们还探讨了合并多个不同的过滤器区域大小的效果，同时保持每个区域大小的feature map的数量为100。在这里，我们发现将几个过滤器与区域大小接近最佳的单一区域大小可以提高性能，但是在最优范围之外添加区域大小可能会损害性能。例如，从图3可以看出，MR数据集的最佳单个区域大小是7。因此，我们将几个不同的过滤器区域大小结合到这个最优范围内，并将其与在此范围之外使用区域大小的方法进行比较。从表9可以看出，使用(5,6,7)和(7,8,9)和(6,7,8,9)——靠近最佳单一区域大小的集合——产生最好的结果。当与(3,4,5)baseline设置比较时，差异尤其明显。注意，即使只使用单个良好的过滤器区域大小(这里为7)，结果也比组合不同的大小(3、4、5)更好。最佳的组合是简单地使用许多特征映射(这里为400)，以及所有区域大小等于7，即最好的区域大小。

但是，我们注意到在某些情况下(例如，对于TREC数据集)，使用多个不同的，但接近最优的区域大小表现最好。我们在表6的TREC数据集上使用几个区域大小提供了另一个示例性经验结果。从单个区域大小的性能来看，我们发现TREC的最佳单过滤区域大小是3和5，因此我们研究这些值附近的区域大小，并将其与使用多个区域大小的值进行比较。

| Multiple region size | Accuracy (%) |
|---|---|
| (7) | 81.65 (81.45,81.85) |
| (3,4,5) | 81.24 (80.69, 81.56) |
| (4,5,6) | 81.28 (81.07,81.56) |
| (5,6,7) | 81.57 (81.31,81.80) |
| (7,8,9) | 81.69 (81.27,81.93) |
| (10,11,12) | 81.52 (81.27,81.87) |
| (11,12,13) | 81.53 (81.35,81.76) |
| (3,4,5,6) | 81.43 (81.10,81.61) |
| (6,7,8,9) | 81.62 (81.38,81.72) |
| (7,7,7) | 81.63 (81.33,82.08) |
| **(7,7,7,7)** | **81.73 (81.33,81.94)** |

Table 9: Effect of filter region size with several region sizes using non-static word2vec-CNN on MR dataset

| Multiple region size | Accuracy (%) |
|---|---|
| (3) | 91.21 (90.88,91.52) |
| (5) | 91.20 (90.96,91.43) |
| (2,3,4) | 91.48 (90.96,91.70) |
| (3,4,5) | 91.56 (91.24,91.81) |
| (4,5,6) | 91.48 (91.17,91.68) |
| (7,8,9) | 90.79 (90.57,91.26) |
| (14,15,16) | 90.23 (89.81,90.51) |
| **(2,3,4,5)** | **91.57 (91.25,91.94)** |
| (3,3,3) | 91.42 (91.11,91.65) |
| (3,3,3,3) | 91.32 (90.53,91.55) |

Table 10: Effect of filter region size with several region sizes using non-static word2vec-CNN on TREC dataset

这里我们看到(3,3,3)和(3,3,3)比(2,3,4)和(3,4,5)更差。然而，结果仍然表明，在最优的最佳区域尺寸附近的区域大小的组合比在最优的单一区域大小下使用多个区域的大小要好得多。此外，我们再次看到一个良好的区域大小(3)超过了几个次优区域大小(7、8、9)和(14、15、16)。

根据这些观察,我们认为这建议先进行粗线通过一个过滤器搜索区域大小找到最好的考虑数据集的大小,然后探索附近的几个区域大小的组合这最好的尺寸,包括结合不同的区域大小和副本的最优尺寸。

## 5.4 特征图数量

我们再次保持其他配置不变，因此有3个过滤器区域大小:3、4和5。我们只更改每个相对于100的baseline的特征映射的数量。我们考虑大小10,50,100,200,400,600,1000,2000。报告结果如图4所示。

| | 10 | 50 | 100 | 200 | 400 | 600 | 1000 | 2000 |
|---|---|---|---|---|---|---|---|---|
| MR | 80.47 (80.14,80.99) | 81.25 (80.90,81.56) | 81.17 (81.00,81.38) | 81.31 (81.00,81.60) | 81.41 (81.21,81.61) | 81.38 (81.09, 81.68) | 81.30 (81.15,81.39) | 81.40 (81.13,81.61) |
| SST-1 | 45.90 (45.14,46.41) | 47.06 (46.58,47.59) | 47.09 (46.50,47.66) | 47.09 (46.34,47.50) | 46.87 (46.41,47.43) | 46.84 (46.29,47.47) | 46.58 (46.26,47.14) | 46.75 (45.87,47.67) |
| SST-2 | 84.26 (83.93,84.73) | 85.23 (84.86,85.57) | 85.50 (85.31,85.66) | 85.53 (85.24,85.69) | 85.56 (85.27,85.79) | 85.70 (85.57,85.93) | 85.75 (85.53,86.01) | 85.74 (85.49,86.02) |
| Subj | 92.24 (91.74,92.43) | 93.07 (92.94,93.28) | 93.19 (93.08,93.45) | 93.29 (93.07,93.38) | 93.24 (92.96,93.39) | 93.34 (93.22,93.44) | 93.32 (93.17,93.49) | 93.34 (93.05,93.49) |
| TREC | 90.64 (90.19,90.86) | 91.40 (91.12,91.59) | 91.54 (91.17,91.90) | 91.54 (91.23,91.71) | 91.52 (91.30,91.70) | 91.50 (91.23,91.71) | 91.44 (91.26,91.56) | 91.54 (91.28,91.75) |
| CR | 79.95 (79.36,80.41) | 83.19 (82.32,83.50) | 83.86 (83.52,84.15) | 84.30 (83.80,84.64) | 84.44 (84.14,85.02) | 84.62 (84.31,84.94) | 84.58 (84.35,84.85) | 84.47 (83.84,85.03) |
| MPQA | 89.02 (88.89,89.19) | 89.21 (88.97,89.41) | 89.21 (88.90,89.51) | 89.50 (89.27,89.68) | 89.57 (89.13,89.81) | 89.66 (89.35,89.90) | 89.55 (89.22,89.73) | 89.66 (89.47,89.94) |

Table 11: Performance of number of feature maps for each filter using non-static word2vec-CNN

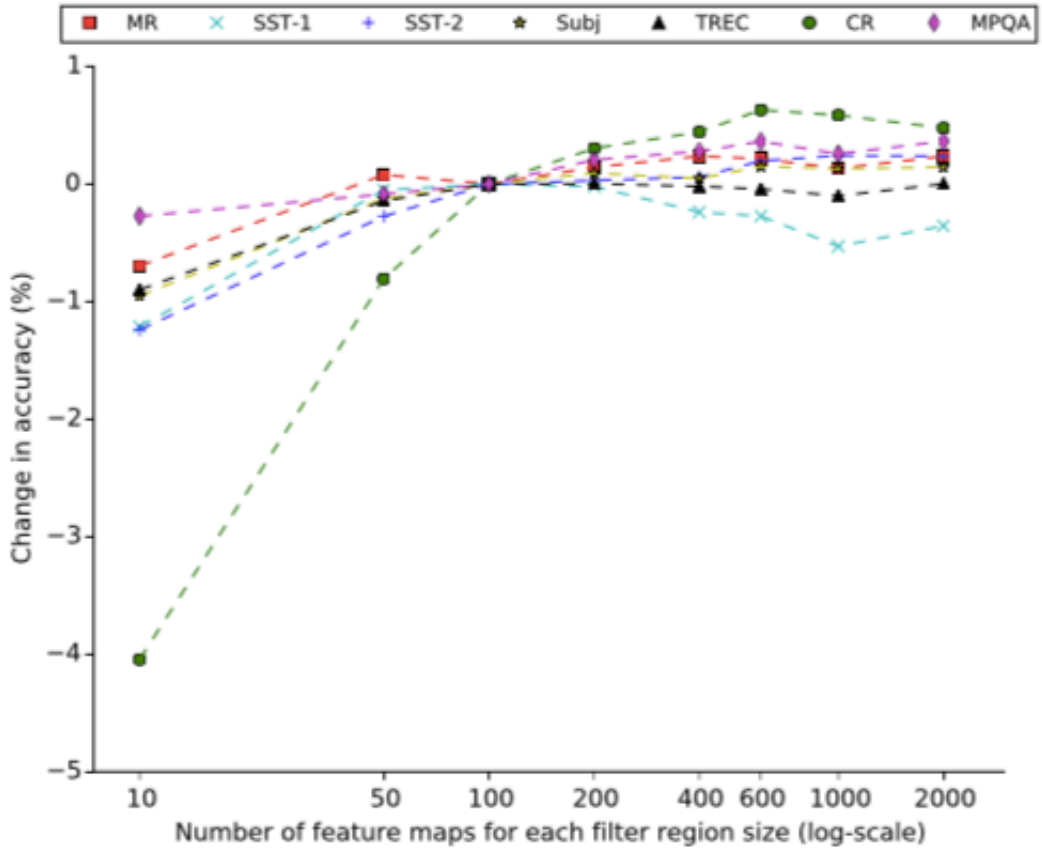


Figure 4: Effect of number of feature maps for each filter region size using non-static word2vec-CNN

每个过滤器区域大小的“最佳”feature map数量取决于数据集。然而，增加超过600个feature

map，充其量只能带来边际收益，而且往往会损害业绩(可能是由于过拟合)。另一个重要的事实是，当feature map的数量增加时，需要更长的时间来训练模型。实际上，这里的证据表明，搜索范围可能在100到600之间。注意，当一个人面临一个新的类似的句子分类问题时，这个范围只是提供一个可能的标准。当然，有可能在某些情况下，超过600个特征图是有益的，但这里的证据表明，花费精力去探索这一点可能是不值得的。

## 5.5 激活函数

我们考虑了卷积层的七个不同的激活函数，包括:ReLU(根据baseline配置)，双曲正切(tanh)，Sigmoid函数(Maas et al.， 2013)， SoftPlus函数(Dugas et al.， 2001)， Cube function (Chen and Manning, 2014)和tanh Cube function (Pei et al.， 2015)。我们使用“Iden”来表示原本函数，这意味着不使用任何激活函数。表15展示了使用不同的激活函数在非静态CNN中实现的结果。在9个数据集中，最好的激活函数是Iden、ReLU和tanh。在只有一个数据集(MPQA)中，SoftPlus函数的性能超过了其他函数。Sigmoid、Cube和tanh数据集始终比其他激活函数执行得更糟糕。因此，我们在这里不报告结果。tanh函数的性能可能是由于它的零中心特性(与Sigmoid相比)。ReLU与Sigmoid相比具有非饱和形式的优点，并且已经观察到可以加速SGD的收敛(Krizhevsky等，2012)。一个有趣的结果是，不应用任何激活函数(Iden)有时会有所帮助。这表明在一些数据集上，一个线性变换足以捕获单词嵌入和输出标签之间的相关性。然而，如果存在多个隐藏层，则Iden可能比非线性激活函数更不合适。实际上，对于单层CNNs中激活函数的选择，我们的研究结果表明对ReLU和tanh进行了实验，也可能是Iden。

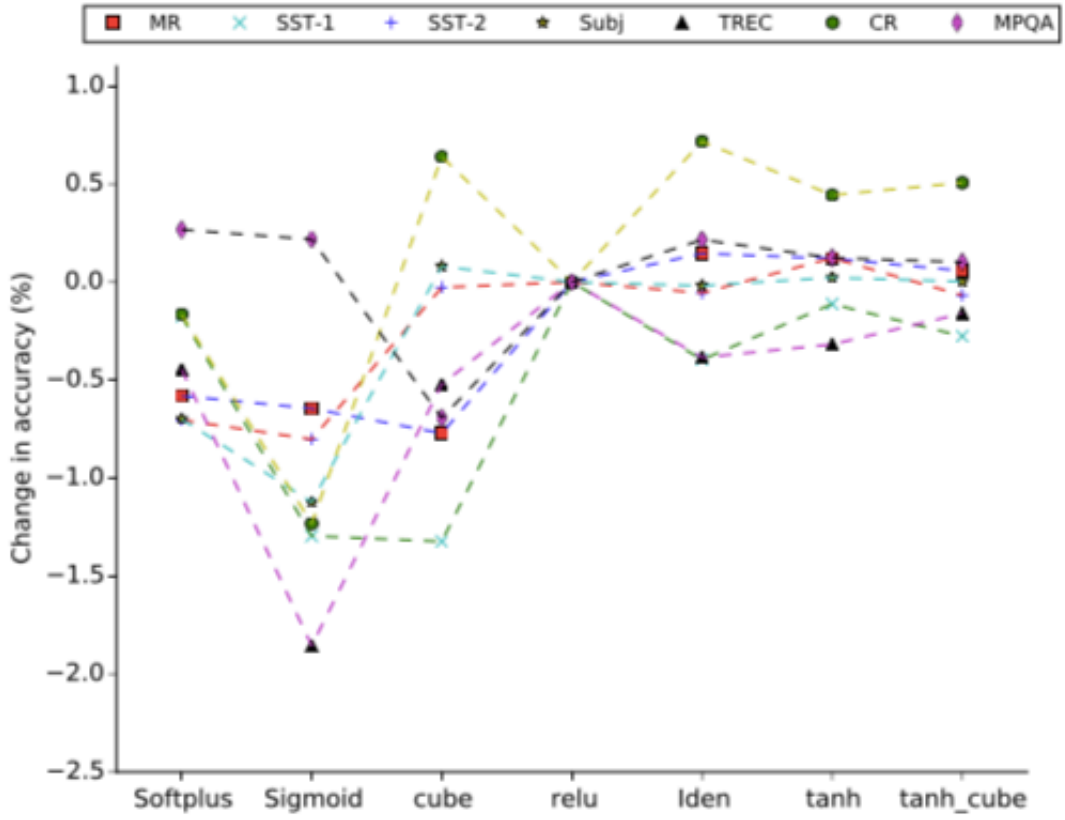


Figure 5: Effect of activation function in non-static word2vec-CNN

| | Sigmoid | tanh | Softplus | Iden | ReLU | Cube | tahn-cube |
|---|---|---|---|---|---|---|---|
| MR | 80.51 (80.22, 80.77) | 81.28 (81.07, 81.52) | 80.58 (80.17, 81.12) | 81.30 (81.09, 81.52) | 81.16 (80.81, 83.38) | 80.39 (79.94,80.83) | 81.22 (80.93,81.48) |
| SST-1 | 45.83 (45.44, 46.31) | 47.02 (46.31, 47.73) | 46.95 (46.43, 47.45) | 46.73 (46.24,47.18) | 47.13 (46.39, 47.56) | 45.80 (45.27,46.51) | 46.85 (46.13,47.46) |
| SST-2 | 84.51 (84.36, 84.63) | 85.43 (85.10, 85.85) | 84.61 (84.19, 84.94) | 85.26 (85.11, 85.45) | 85.31 (85.93, 85.66) | 85.28 (85.15,85.55) | 85.24 (84.98,85.51) |
| Subj | 92.00 (91.87, 92.22) | 93.15 (92.93, 93.34) | 92.43 (92.21, 92.61) | 93.11 (92.92, 93.22) | 93.13 (92.93, 93.23) | 93.01 (93.21,93.43) | 92.91 (93.13,93.29) |
| TREC | 89.64 (89.38, 89.94) | 91.18 (90.91, 91.47) | 91.05 (90.82, 91.29) | 91.11 (90.82, 91.34) | 91.54 (91.17, 91.84) | 90.98 (90.58,91.47) | 91.34 (90.97,91.73) |
| CR | 82.60 (81.77, 83.05) | 84.28 (83.90, 85.11) | 83.67 (83.16, 84.26) | 84.55 (84.21, 84.69) | 83.83 (83.18, 84.21) | 84.16 (84.47,84.88) | 83.89 (84.34,84.89) |
| MPQA | 89.56 (89.43, 89.78) | 89.48 (89.16, 89.84) | 89.62 (89.45, 89.77) | 89.57 (89.31, 89.88) | 89.35 (88.88, 89.58) | 88.66 (88.55,88.77) | 89.45 (89.27,89.62) |

Table 12: Performance of different activation functions using non-static word2vec-CNN

## 5.6 池化

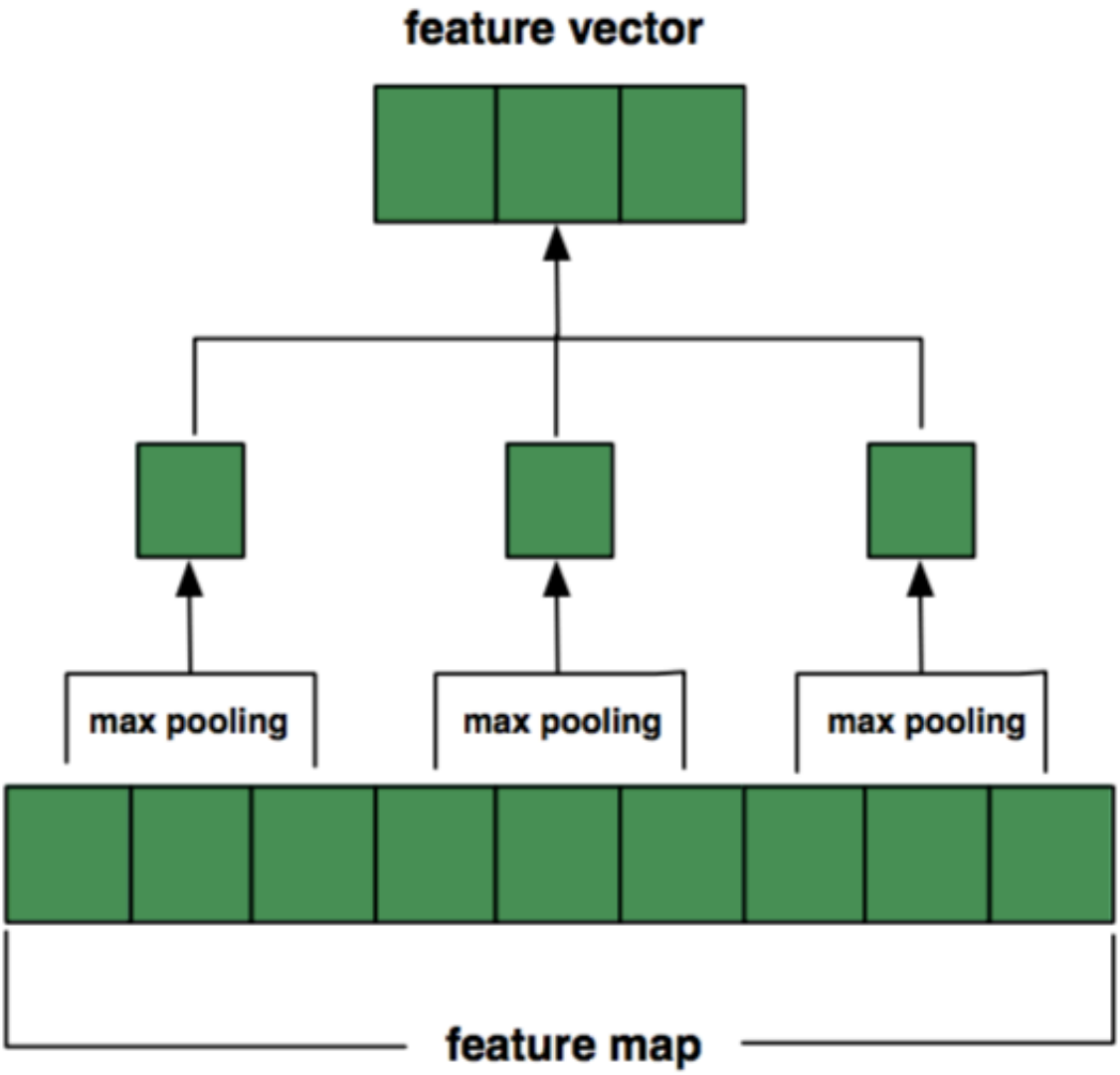


Figure 6: Local max pooling within 3 sized region

接下来我们研究了池化策略和池化区域大小的影响。我们将过滤器区域大小和特征映射的数量固定在baseline配置中，从而只改变池策略或池区域大小。在baseline参数设定中，我们对所有的feature map都使用了最大池化。得到长度为1的特征向量。但是，也可以在较小的相同大小的局部区域上执行池化操作，而不是在整个feature map (Boureau et al.， 2011)上执行。feature map上的每个小区域都经过池化操作生成单个数字，并且这些数字可以连接成一个feature map对应的特征向量。下面的步骤与1-max池相同:我们将所有的特征向量连接在一起，形成分类层的单个特征向量。我们试验了大小为3、10、20和30的局部区域，并发现1-max池比所有局部最大池配置的性能好。所有的数据集都呈现了这个结果。我们还考虑了类似于k-max池化的策略(Kalchbrenner et al.， 2014)，其中从整个feature map中提取了最大的k个值，并保留了这些值的相对顺序。我们对k进行了探索，发现1-max池的性能最好，始终优于k-max池。

| | max,3 | max,10 | max,20 | max,30 | max,all (1-max) |
|---|---|---|---|---|---|
| MR | 79.75 (79.47,80.03) | 80.20 (80.02,80.35) | 80.68 (80.14,81.21) | 80.99 (80.65,81.30) | 81.28 (81.16,81.54) |
| SST-1 | 44.98 (44.06,45.68) | 46.10(45.37,46.84) | 46.75 (46.35,47.36) | 47.02 (46.59,47.59) | 47.00 (46.54,47.26) |
| SST-2 | 83.69(83.46,84.07) | 84.63 (84.44,84.88) | 85.18 (84.64,85.59) | 85.38 (85.31,85.49) | 85.50 (85.31,85.83) |
| Subj | 92.60 (92.28,92.76) | 92.87 (92.69,93.17) | 93.06 (92.81,93.19) | 93.13 (92.79,93.32) | 93.20 (93.00,93.36) |
| TREC | 90.29 (89.93,90.61) | 91.42 (91.16,91.71) | 91.52 (91.23,91.72) | 91.47 (91.15,91.64) | 91.56 (91.67,91.88) |
| CR | 81.72 (81.21,82.20) | 82.71 (82.06,83.30) | 83.44(83.06,83.90) | 83.70 (83.31,84.25) | 83.93 (83.48,84.39) |
| MPQA | 89.15 (88.83,89.47) | 89.39 (89.14,89.56) | 89.30 (89.16,89.60) | 89.37 (88.99,89.61) | 89.39 (89.04,89.73) |

Table 13: Performance of local max pooling using non-static word2vec-CNN

| | 1 (1-max) | 5 | 10 | 15 | 20 |
|---|---|---|---|---|---|
| MR | 81.25 (81.00,81.47) | 80.83 (80.69,80.91) | 80.05 (79.69,80.41) | 80.11 (79.89,80.36) | 80.05 (79.72,80.25) |
| SST-1 | 47.24 (46.90,47.65) | 46.63 (46.31,47.12) | 46.04 (45.27,46.61) | 45.91 (45.16,46.49) | 45.31 (44.90,45.63) |
| SST-2 | 85.53 (85.26,85.80) | 84.61(84.47,84.90) | 84.09 (83.94,84.22) | 84.02 (83.57,84.28) | 84.04 (83.74,84.34) |
| Subj | 93.18 (93.09,93.31) | 92.49 (92.33,92.61) | 92.66 (92.50,92.79) | 92.52 (92.33,92.96) | 92.58 (92.50,92.83) |
| TREC | 91.53 (91.26,91.78) | 89.93 (89.75,90.09) | 89.73 (89.61,89.83) | 89.49(89.31,89.65) | 89.05(88.85,89.34) |
| CR | 83.81 (83.44,84.37) | 82.70 (82.14,83.11) | 82.46 (82.17,82.76) | 82.26 (81.86, 82.90) | 82.09 (81.74,82.34) |
| MPQA | 89.39 (89.14, 89.58) | 89.36 (89.17,89.57) | 89.14 (89.00,89.45) | 89.31 (89.18,89.48) | 88.93 (88.82,89.06) |

Table 14: Performance of global k-max pooling using non-static word2vec-CNN

接下来，我们考虑取区域的平均值，而不是区域的最大值(Boureau等人，2010a)。我们保留了其余的参数。我们尝试了区域大小为 的局部平均池化。我们发现，至少在CR和TREC数据集上，平均池化比最大池化的性能差(很多)。由于在平均池下观察到的性能和运行时间非常慢，所以我们没有对所有数据集进行完整的实验。我们对池化策略的分析表明，1-max池化对句子分类任务的效果总是优于其他策略。这可能是因为预测上下文的位置无关紧要，而句子中的某些n-grams可以比共同考虑的整个句子更具预测性。

| Pooling region | CR | TREC |
|---|---|---|
| 3 | 81.01 (80.73,81.28) | 88.89 (88.67,88.97) |
| 10 | 80.74 (80.36,81.09) | 88.10 (87.82,88.47) |
| 20 | 80.69 (79.72,81.32) | 86.45 (85.65,86.42) |
| 30 | 81.13 (80.16,81.76) | 84.95 (84.65,85.14) |
| all | 80.17 (79.97,80.84) | 83.30 (83.11,83.57) |

Table 15: Performance of local average pooling region size using non-static word2vec-CNN ('all' means average pooling over the whole feature map resulting in one number)

## 5.7 正则化

CNNs的两种常见正则化策略是dropout和l2范数。我们在这里探讨这些效应。从输入到倒数第一

层应用Dropout。我们试验了从0.0到0.9的dropout比率，并根据baseline配置将l2范数约束固定到3。非静态CNN的结果如图5所示，0.5指定为baseline。我们也展示了当我们去掉了dropout和l2范数约束时(即不执行正则化时)的准确性，表示为None。另外，我们还考虑了l2正则对权重向量的影响，这些权重向量参数化了softmax函数。回想一下，当一个权重向量的l2范数超过这个阈值时，它是线性伸缩的，因此较小的c意味着更强的正则化。像dropout，这个策略只适用于倒数第一层。我们在图8中显示了不同c对非静态CNN的相对影响，我们将dropout率固定在0.5；3是这里的baseline模型的正则化参数，（再一次地，任意地)。

| | None | 0.0 | 0.1 | 0.3 | 0.5 | 0.7 | 0.9 |
|---|---|---|---|---|---|---|---|
| MR | 81.15 (80.95,81.34) | 81.24 (80.82, 81.63 ) | 81.22 (80.97 ,81.61 ) | 81.30 (81.03 ,81.48 ) | 81.33 (81.02, 81.74 ) | 81.16 (80.83, 81.57 ) | 80.70 (80.36, 80.89) |
| SST-1 | 46.30 (45.81,47.09) | 45.84 (45.13 ,46.43 ) | 46.10 (45.68, 46.36 ) | 46.61 (46.13, 47.04 ) | 47.09 (46.32, 47.66 ) | 47.19 (46.88 ,47.46 ) | 45.85 (45.50, 46.42 ) |
| SST-2 | 85.42 (85.13,85.23) | 85.53 (85.12 ,85.88 ) | 85.69 (85.32, 86.06 ) | 85.58 (85.30, 85.76 ) | 85.62 (85.25, 85.92 ) | 85.41 (85.18, 85.65 ) | 84.49 (84.35, 84.82 ) |
| Subj | 93.23 (93.09,93.37) | 93.21 (93.09 ,93.31 ) | 93.27 (93.12 ,93.45 ) | 93.28 (93.06, 93.39 ) | 93.14 (93.01, 93.32 ) | 92.94 (92.77 ,93.08 ) | 92.03 (91.80 ,92.24 ) |
| TREC | 91.38 (91.18,91.59) | 91.39 (91.13 ,91.66 ) | 91.41 (91.26, 91.63 ) | 91.50 (91.22 ,91.76 ) | 91.54 (91.41, 91.68 ) | 91.45 (91.17, 91.77 ) | 88.83 (88.53 ,89.19 ) |
| CR | 84.36 (84.06,84.70) | 84.04 (82.91, 84.84 ) | 84.22 (83.47, 84.60 ) | 84.09 (83.72, 84.51 ) | 83.92 (83.12, 84.34 ) | 83.42 (82.87, 83.97 ) | 80.78 (80.35, 81.34 ) |
| MPQA | 89.30 (88.91,89.68) | 89.30 (89.01, 89.56 ) | 89.41 (89.19, 89.64 ) | 89.40 (89.18, 89.77 ) | 89.25 (88.96, 89.60 ) | 89.24 (88.98, 89.50 ) | 89.06 (88.93, 89.26 ) |

Table 16: Effect of dropout rate using non-static word2vec-CNN

| | MR | SST-1 | SST-2 | Subj | TREC | CR | MPQA |
|---|---|---|---|---|---|---|---|
| 1 | 81.02 (80.75 ,81.29) | 46.93 (46.57, 47.33) | 85.02 (84.76,85.22) | 92.49 (92.35 92.63) | 90.90 (90.62 91.20) | 83.06 (82.50 83.42) | 89.17 (88.97 89.36) |
| 2 | 81.33 (81.04 ,81.71) | 47.11 (46.77, 47.43) | 85.40 (84.98,85.67) | 92.93 (92.82 93.15) | 91.44 (91.20 91.60) | 84.00 (83.57 84.34) | 89.31 (89.17 89.54) |
| 3 | 81.29 (80.96, 81.59) | 47.29 (46.90 ,47.82) | 85.47 (85.17,85.77) | 93.21 (93.03 93.37) | 91.44 (91.18 91.68) | 83.89 (83.24 84.47) | 89.18 (88.84 89.40) |
| 4 | 81.38 (81.21, 81.68) | 46.91 (46.22 ,47.38) | 85.33 (85.25,85.72) | 93.08 (92.96 93.22) | 91.56 (91.26 91.90) | 84.00 (83.21 84.60) | 89.27 (89.11 89.41) |
| 5 | 81.22 (81.03, 81.49) | 46.93 (46.44 ,47.38) | 85.46 (84.98,85.73) | 93.14 (92.90 93.33) | 91.58 (91.39 91.87) | 83.99 (83.73 84.31) | 89.33 (89.02 89.55) |
| 10 | 81.19 (80.94 ,81.42) | 46.74 (46.19, 47.12) | 85.41 (85.04,85.83) | 93.11 (92.99 93.32) | 91.58 (91.29 91.81) | 83.94 (83.04 84.61) | 89.22 (89.01 89.40) |
| 15 | 81.12 (80.87, 81.29) | 46.91 (46.58 ,47.48) | 85.47 (85.23,85.74) | 93.15 (92.99 93.29) | 91.58 (91.37 91.84) | 83.92 (83.40 84.54) | 89.30 (88.93 89.66) |
| 20 | 81.13 (80.64, 81.33) | 46.96 (46.62 ,47.31) | 85.46 (85.17,85.64) | 93.10 (92.98 93.19) | 91.54 (91.28 91.73) | 84.09 (83.59 84.53) | 89.28 (88.92 89.43) |
| 25 | 81.22 (80.82, 81.66) | 47.02 (46.73, 47.67) | 85.42 (85.16,85.78) | 93.09 (92.95 93.25) | 91.45 (91.22 91.62) | 83.91 (83.24 84.40) | 89.33 (89.05 89.61) |
| 30 | 81.19 (80.79 ,81.43) | 46.98 (46.63 ,47.59) | 85.48 (85.27,85.79) | 93.06 (92.84 93.43) | 91.55 (91.26 91.84) | 83.94 (83.02 84.35) | 89.26 (89.10 89.54) |
| None | 80.19(79.95,80.39) | 46.30 (45.81,47.09) | 85.42 (85.13,85.23) | 93.23 (93.09,93.37) | 91.38 (91.18,91.59) | 84.36 (84.06,84.70) | 89.30 (88.91,89.68) |

Table 17: Effect of constraint on $l2$ norm using non-static word2vec-CNN

从图7和图8可以看出，根据数据集，非零的dropout比率可以帮助(尽管非常少)从0.1到0.5。但是，施加l2正则约束通常不会很大地提高性能(除了Opi)，甚至对至少一个数据集(CR)的性能产生负面影响。我们还研究了在增加feature map的数量时dropout比率效应。我们将每个过滤器大小的feature map的数量从100增加到500，并将max l2正则约束设置为3。dropout比率的影响如图7所示。我们看到，drouout比率的影响几乎和特征图的数量是100的时候一样，而且没有多大帮助。但是我们观察到，对于数据集SST-1来说，当它是0.7时，dropout比率实际上是有帮助的。从图4可以看出，当feature map的数量大于100时，可能由于过拟合而影响了性能，所以在这个情况下dropout将会减轻这种影响。

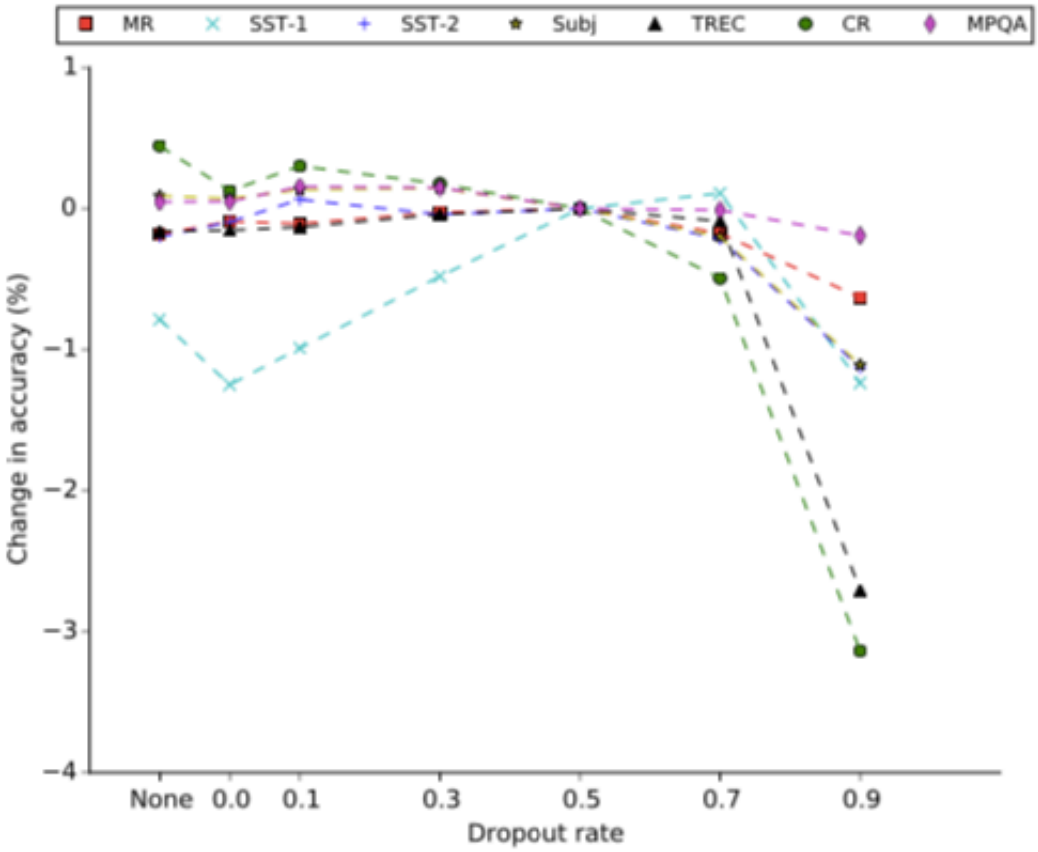


Figure 7: Effect of dropout rate in non-static word2vec-CNN

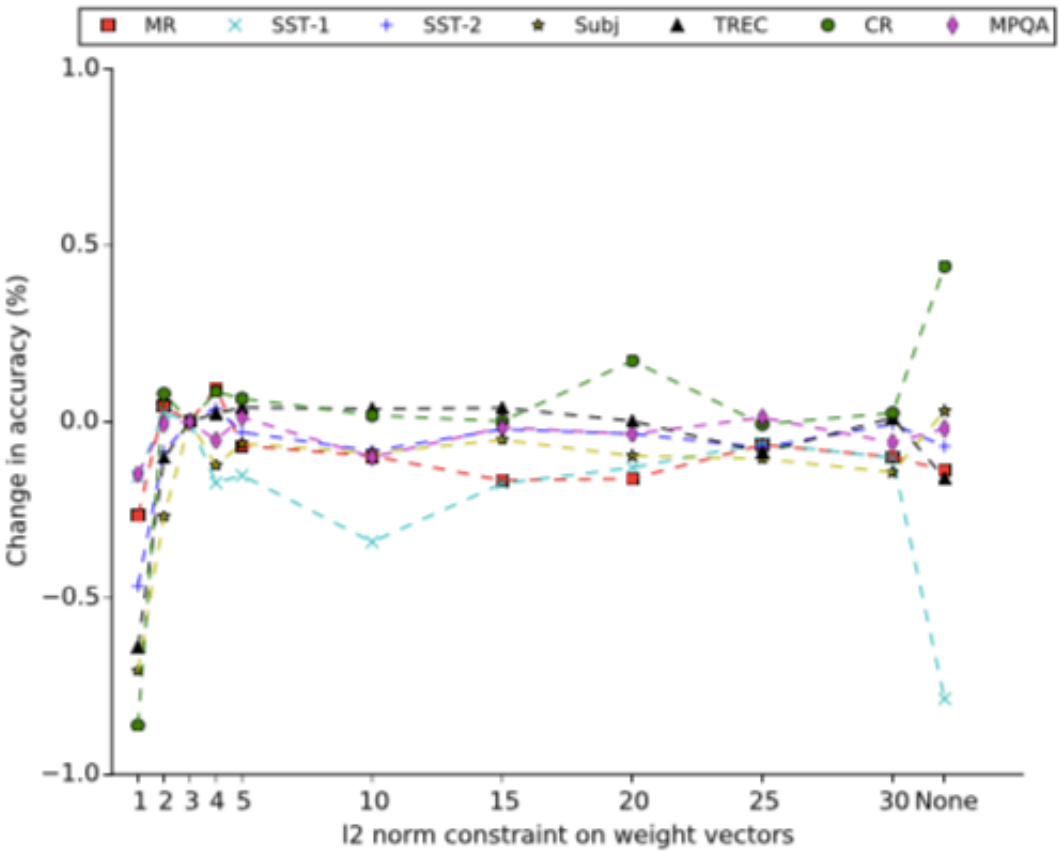


Figure 8: Effect of $l2$ norm constraint on weight vectors in non-static word2vec-CNN

我们也尝试了只在卷积层上应用“dropout”，但仍然将分类层的最大标准约束设置为3，使所有其他设置完全相同。这意味着我们在训练时随机将句子矩阵的元素设置为0，然后在测试时将p与句子矩阵相乘。从图8中可以看出，dropout比率对卷积层的影响如图8所示。我们再次看到，在卷积层上运用dropout帮助很小，而且很大的dropout率极大地伤害了性能。

总之，与现有的一些文献(Srivastava et al.， 2014)相反，我们发现dropout对CNN的表现没有什么好处。我们将这一现象归因于一层CNN的参数数量比多层深度学习模型要小。另一种可能的解释是，使用词嵌入有助于防止过拟合(与基于单词的编码相比)。然而，我们并不是主张完全放弃正则化。实际上，我们建议将dropout率设置为一个小的值(0 -0.5)，并使用一个相对较大的max正则约束，同时增加feature maps的数量，以查看更多的特性是否会有所帮助。当进一步增加feature map的数量似乎会降低性能时，增加dropout比率可能是值得的。

# 六、结论

我们对CNNs的句子分类进行了广泛的实验分析。我们总结了我们的主要发现，并从这些实际的指导中总结出了研究人员和实践者在现实场景中使用和部署cnn的方法。

## 6.1 主要实证结果的总结

- 以前的工作往往只报告模型实现的数据集的平均性能。但是，这种忽略方差完全是由于随机推理过程所使用的。这可以是相当大的:保持所有的常数(包括折叠)，因此方差是完全由随机推理过程决定的，我们发现，平均精度(通过10倍交叉验证计算)的范围可以达到1.5个点。在irony数据集上，AUC的范围甚至更大，达到3.4分(见表3)。在将来的工作中应该进行更多的复制，并且应该报告范围/方差，以防止可能的关于相对模型性能的错误结论。我们发现，即使将它们调到手边的任务，输入词向量表示(例如，在word2vec和Glove之间)的选择对性能有影响，但是不同的表示对不同的任务有更好的表现。至少对于句子分类来说，两者似乎都比直接使用one-hot向量要好。

- 然而,我们注意到:(1)如果有一个足够大量的训练数据，结果可能就不是这样,以及(2)与这里的简单版本相比，最近由约翰逊和张提出的semi-supervised CNN模型(Johnson and Zhang, 2015)可能提高性能(Johnson and Zhang, 2014)。

- 过滤区域的大小对性能有很大的影响，应该进行调整。

- feature map的数量也可以在性能上扮演重要的角色，增加feature map的数量会增加模型的训练时间。

- 1-max池一致优于其他池化策略。

- 正则化对模型的性能影响较小。

## 6.2对从业人员的具体建议

根据我们的经验结果，我们提供了关于CNN架构和超参数的指南，为那些希望在句子分类任务中部署cnnn的从业者提供参考。

- 考虑从表2中描述的基本配置开始，使用非静态word2vec或Golve，而不是one-hot矢量。但是，如果训练数据集的大小是非常大的，那么使用one-hot向量来探索可能是值得的。或者，如果一个人能够访问大量未标记的域内数据(Johnson和Zhang, 2015)，也可能是一个选项。

- 通过对单个过滤器区域大小的线性搜索来找到“最佳”的单一区域大小。一个合理的范围可能1至10。然而，对于像CR这样的长句的数据集，可能值得探索更大的过滤器区域大小。一旦确定了这一“最佳”区域大小，就可能值得探索将多个过滤器组合在一起，使用区域大小接近这个最佳大小的区域，因为根据经验，多个“好”区域大小总是优于仅使用单一最佳区域大小。

- 将每个过滤器区域的特征映射的数量从100个更改为600个，并且在这个过程中，使用一个

小的dropout比率(0 -0.5)和一个大的max正则约束。注意，增加feature map的数量会增加运行时间，因此需要权衡考虑。还要注意发现的最佳值是否在距离边界附近(Bengio, 2012)。如果最好的值接近600，那么尝试更大的值可能是值得的。

- 如果可能，考虑不同的激活函数:ReLU和tanh是最好的整体候选。对于我们的一层CNN来说，完全没有激活函数是值得的。
- 使用1-max池;似乎没有必要花费资源来评估替代战略。
- 关于正则化:当增加feature map的数量开始减少性能时，试着施加更强的正则化，例如，dropout比率大于0.5。
- 在评估模型的性能(或其特定配置)时，必须考虑方差。因此，应该重复交叉折叠验证过程，并考虑方差和范围。

当然，以上建议仅适用于包含有相似属性的句子的数据集。也许有一些与我们的发现背道而驰的例子。尽管如此，我们相信这些建议可能会为研究人员或实践者提供一个合理的起点，他们希望将简单的一层CNN应用到现实世界的句子分类任务中。我们强调，我们选择了这个简单的单层CNN，根据观察到的强大的经验性能，它将它定位为一个新的baseline模型，类似于词袋SVM和逻辑回归。因此，在实施更复杂的模型之前，应该考虑这种方法。