



Published in Towards Data Science



Joshua Kim

Follow

Dec 2, 2017 · 4 min read · Listen



Save

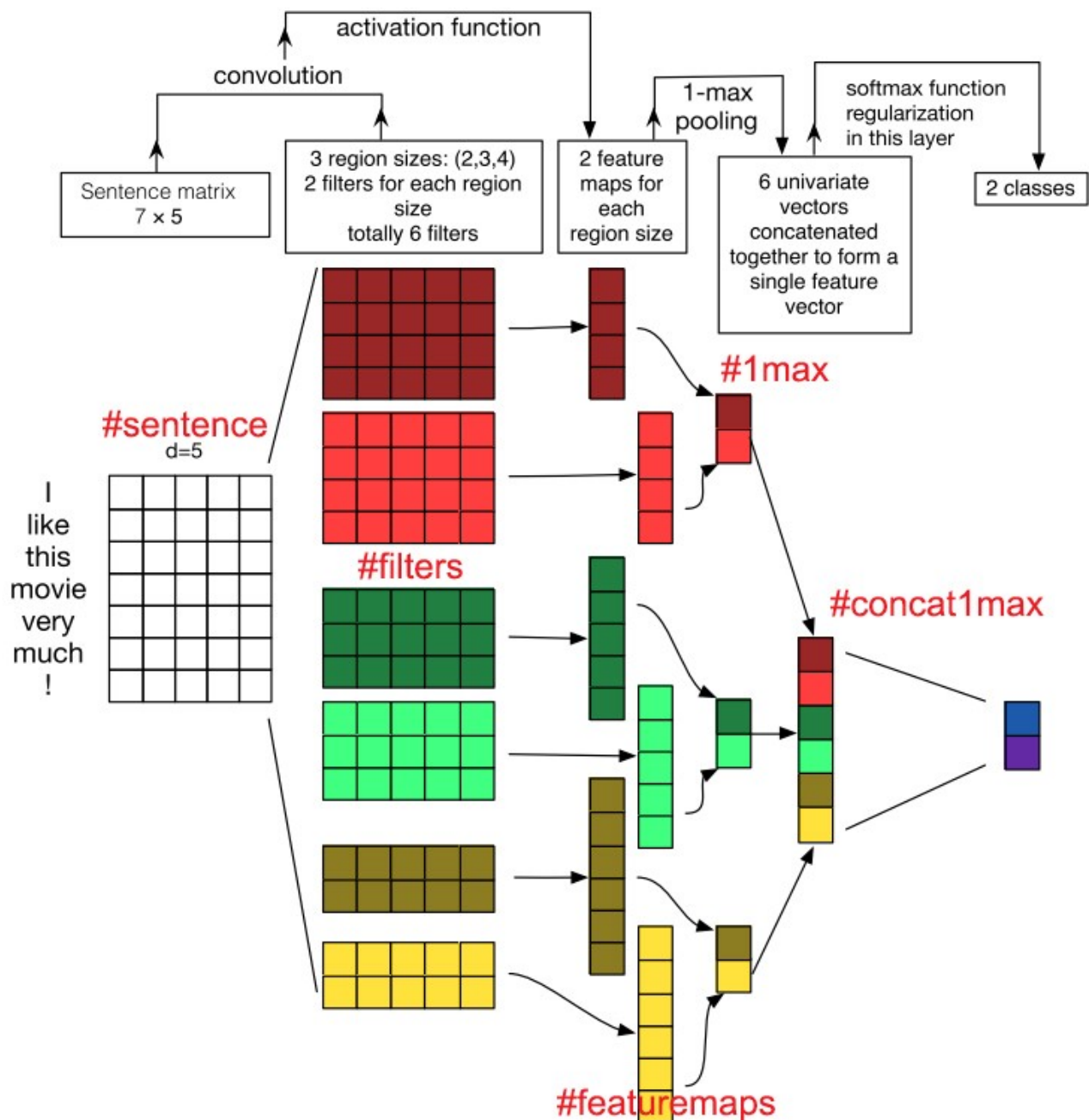


Understanding how Convolutional Neural Network (CNN) perform text classification with word embeddings

CNN has been successful in various text classification tasks. In [1], the author showed that a simple CNN with little hyperparameter tuning and static vectors achieves excellent results on multiple benchmarks — improving upon the state of the art on 4 out of 7 tasks.

However, when learning to apply CNN on word embeddings, keeping track of the dimensions of the matrices can be confusing. The aim of this short post is to simply to keep track of these dimensions and understand how CNN works for text classification. We would use a one-layer CNN on a 7-word sentence, with word embeddings of dimension 5 — a toy example to aid the understanding of CNN. All examples are from [2].

Setup



Above figure is from [2], with #hash-tags added to aid discussion. Quoting the original caption here, to be discussed later. "Figure 1: Illustration of a CNN architecture for sentence classification. We depict three filter region sizes: 2,3,4, each of which has 2 filters. Filters perform convolutions on the sentence matrix and generate (variable-length) feature maps; 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 later 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."

#sentence

The example is "I like this movie very much!", there are 6 words here and the exclamation mark is treated like a word — some researchers do this differently and disregard the exclamation mark — in total there are 7 words in the sentence. The authors chose 5 to be the dimension of the word vectors. We let s denote the length

of sentence and d denote the dimension of the word vector, hence we now have a sentence matrix of the shape $s \times d$, or 7×5 .

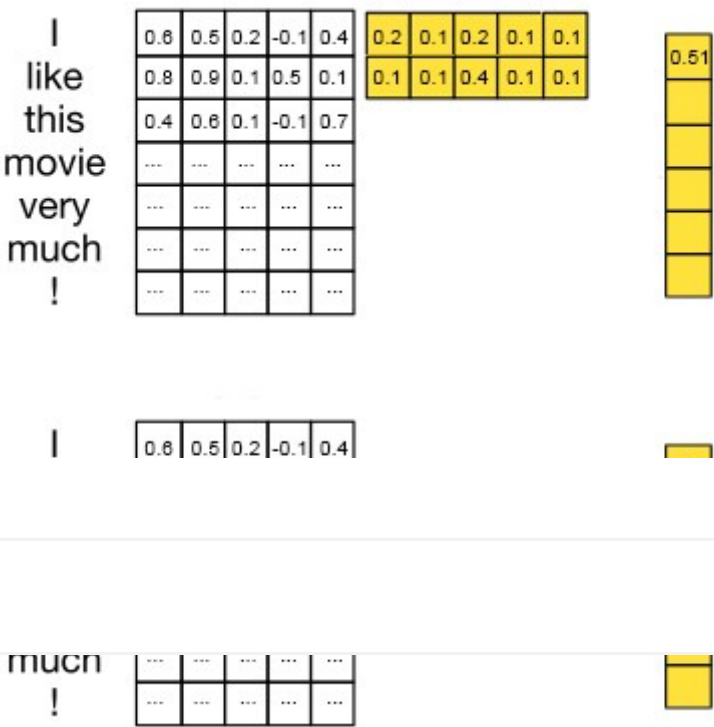
#filters

One of the desirable properties of CNN is that it preserves 2D spatial orientation in computer vision. Texts, like pictures, have an orientation. Instead of 2-dimensional, texts have a one-dimensional structure where words sequence matter. We also recall that all words in the example are each replaced by a 5-dimensional word vector, hence we fix one dimension of the filter to match the word vectors (5) and vary the region size, h . Region size refers to the number of rows — representing word — of the sentence matrix that would be filtered.

In the figure, #filters are the illustrations of the filters, not what has been filtered out from the sentence matrix by the filter, the next paragraph would make this distinction clearer. Here, the authors chose to use 6 filters — 2 complementary filters to consider (2,3,4) words.

#featuremaps

For this section, we step-through on how CNN perform convolutions / filtering. I have filled in some numbers in the sentence matrix and the filter matrix for clarity.



The action of the 2-word filter on the sentence matrix.

First, the two-word filter, represented by the 2×5 yellow matrix w , overlays across the word vectors of “I” and “like”. Next, it performs an element-wise product for all its 2×5 elements, and then sum them up and obtain one number ($0.6 \times 0.2 + 0.5 \times 0.1 + \dots + 0.1 \times 0.1 = 0.51$). 0.51 is recorded as the first element of the output sequence, o , for this filter. Then, the filter moves down 1 word and overlays across the word vectors of ‘like’ and ‘this’ and perform the same operation to get 0.53. Therefore, o will have the shape of $(s-h+1 \times 1)$, in this case $(7-2+1 \times 1)$



To obtain the feature map, c , we add a bias term (a scalar, i.e., shape 1×1) and apply an activation function (e.g. ReLU). This gives us c , with the same shape as o ($s-h+1 \times 1$).

#1max

Notice that the dimensionality of c is dependent both s and h , in other words, it will vary across sentences of different lengths and filters of different region sizes. To tackle this problem, the authors employ the 1-max pooling function and extract the largest number from each c vector.

#concat1max

After 1-max pooling, we are certain to have a fixed-length vector of 6 elements (= number of filters = number of filters per region size (2) x number of region size considered (3)). This fixed length vector can then be fed into a softmax (fully-connected) layer to perform the classification. The error from the classification is then back-propagated back into the following parameters as part of learning:

- The w matrices that produced o
- The bias term that is added to o to produce c
- Word vectors (optional, use v  469 |  8 nce to decide)

Conclusion

This short post clarifies the workings of the CNN on word embeddings by focussing on the dimensionality of matrices in each intermediate step.

References

1. Kim Y. Convolutional Neural Networks for Sentence Classification. 2014;

2. Zhang Y, Wallace B. A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification. arXiv preprint arXiv:151003820. 2015; PMID: 463165

Machine Learning

Deep Learning

NLP

Naturallanguageprocessing

Convolutional Network

Sign up for The Variable

By Towards Data Science

Every Thursday, the Variable delivers the very best of Towards Data Science: from hands-on tutorials and cutting-edge research to original features you don't want to miss. [Take a look.](#)

By signing up, you will create a Medium account if you don't already have one. Review our [Privacy Policy](#) for more information about our privacy practices.



Get this newsletter

[About](#) [Help](#) [Terms](#) [Privacy](#)

Get the Medium app



Download on the
App Store



GET IT ON
Google Play