Convolutional Neural Networks for Sentence Classification

1 Citation

Kim, Yoon. "Convolutional neural networks for sentence classification." arXiv preprint arXiv:1408.5882 (2014).

`https://arxiv.org/pdf/1408.5882.pdf`

2 Abstract

We stack a convolutional layer, max pooling, dense layer, and softmax on word vectors to get state of the art performance on 4 of the 7 NLP tasks that we tried.

3 Introduction

Word vectors turn 1-of-V encoded words into a smaller dimensional vector where semantically similar vectors are near (measured in Euclidean or cosine distance) each other. We train a Convolutional Neural Network (CNN) on the word2vec vectors and get great performance on NLP tasks.

4 Model

Take a sentence with n words (pad the sentence if it's not long enough) and concatenate the word vectors together.

$$\mathbf{x}_{1:n} = \mathbf{x}_1 \oplus \mathbf{x}_2 \oplus ... \oplus \mathbf{x}_n$$

We can apply a convolution filter $\mathbf{w}$ over a window of h words to get a feature. That is:

$$c_i = f(\mathbf{w} \cdot \mathbf{x}_{i:i+h-1} + b)$$

where f is tanh. We can slide the filter over the input sentence to get a feature map:

$$\mathbf{c} = [c_1, c_2, ..., c_{n-h+1}]$$

We can then max-pool to get $\hat{c} = \max(\mathbf{c})$.

In practice, we use many filters, so we get many feature maps. This goes through max-pooling to get one feature vector for the sentence. This feature vector goes through a fully-connected (i.e. dense) layer and then gets classified with a softmax.

We also consider a two channel system where each word is represented by two word vectors - one that we propagate the gradient back through and another that we leave static.

We regularize with dropout on the dense layer. Without dropout, the dense layer is $y = \mathbf{w} \cdot \mathbf{z} + b$ for $\mathbf{z} = [\hat{c}_1, ..., \hat{c}_m]$ (where m is the number of filters in the conv layer). With dropout, the equation becomes:

$$y = \mathbf{w} \cdot (\mathbf{z} \otimes \mathbf{r}) + b$$

where $\otimes$ is element-wise product and $\mathbf{r}$ is a mask where element is sampled from a Bernoulli mask with probability p. At test time, we set weights to $\hat{\mathbf{w}} = p\mathbf{w}$.