A Convolutional Neural Network for Modelling Sentences

1 Citation

Kalchbrenner, Nal, Edward Grefenstette, and Phil Blunsom. "A convolutional neural network for modelling sentences." arXiv preprint arXiv:1404.2188 (2014).

`https://arxiv.org/pdf/1404.2188v1.pdf`

2 Abstract

We make a Dynamic Convolutional Neural Network (DCNN) that uses Dynamic k-Max Pooling that does very well on binary/multiclass sentiment classification and six-way question classification. It sets state of the art on Twitter sentiment classification.

3 Introduction

Modeling a sentence as a vector is important for many NLP tasks. One approach is to learn word vectors and sum, average, or concatenate them together. Another option is to learn a function that combines word vectors together. A third option is to use logical forms. Neural network based approaches, like recursive neural networks and time-delay networks are popular.

Our approach uses a convolutional neural network. We introduce Dynamic k-Max pooling, which finds the subsequence with the k largest values in an input vector (notice that this preserves the order of these values in the vector). It also allows k to be dynamically chosen as a function of other network properties. Basically, we iterate convolution, (dynamic) k-max pooling, nonlinearity, and another operation called folding (i.e. sum pairs of rows in the input matrix) and end with a fully-connected layer and softmax.

4 Background

The Neural Bag of Words (NBoW) model learns an embedding matrix to map words or bigrams to vectors. It then sums or averages the vectors and sends the result through a fully connected layer and softmax. Another model, called the Recursive Neural Network (RecNN) basically builds a binary tree on word vectors in the sentence. It then learns a function that combines the left and right nodes to figure out a word-vector for the parent. Finally, the root node's vector is the representation for the sentence. A Recurrent Neural Network is just an RecNN that uses a linear structure instead of a binary tree. Yet another model is the Time-delay Neural Network.

Given a conv filter $\mathbf{m} \in \mathbb{R}^m$ and sentence $\mathbf{s} \in \mathbb{R}^s$, convolution produces the sequence $\mathbf{c}$ where:

$$c_j = \mathbf{m}^T \mathbf{s}_{j-m+1:j}$$

If we require $s \geq m$, we get $\mathbf{c} \in \mathbb{R}^{s-m+1}$ (narrow convolution). If we zero-pad the sentence, we can get a $\mathbf{c} \in \sim + \gtrdot - \nVdash$ (wide convolution). We prefer wide conv because it ensures every weight reaches every part of the sentence.

In practice, each word in a sentence is represented by a word vector, not by a scalar, so we have $\mathbf{s} \in \mathbb{R}^{d \times s}$ and $\mathbf{m} \in \mathbb{R}^{d \times m}$. This gives us a Time-Delay Neural Network (TDNN). If we use narrow conv and then take a maximum across each dimension (to handle variable length sentences), we get a Max-TDNN. We can then feed this into a fully-connected layer and softmax. This is a neat model, but it doesn't allow for hierarchical features and loses some ordering information (i.e. taking a maximum doesn't tell you where the feature occurred in the sentence).

5 Convolutional Neural Networks with Dynamic k-Max Pooling

First, we turn our words into word vectors. Then, we do wide conv. This gives us a $d \times (s + m - 1)$ matrix (feature map) for each conv filter that we use.

k-Max Pooling selects the length k subsequence (called $\mathbf{p}^k_{max}$) of the feature map where selected values are the k largest in the feature map. Notice this lets us select multiple values and preserve their order. To turn this into Dynamic k-Max Pooling, we just need to pick k in a formulaic way. For example, the k for layer l (out of L conv layers in the network) is:

$$k_l = \max\left(k_{top}, \lceil \frac{L-l}{L} s \rceil\right)$$

After this pooling, we add a bias $\mathbf{b} \in \mathbb{R}^d$ and apply a nonlinearity g.

So far, we've discussed how to compute one feature map $\mathbf{F}^1_1$. You can compute many such feature maps: $\mathbf{F}^1_1, ..., \mathbf{F}^1_n$. You can also feed these feature maps to the next layer to compute more feature maps: $\mathbf{F}^2_1, ...$ In other words, you can compute the j^{th} feature map at layer i with:

$$\mathbf{F}^i_j = \sum_{k=1}^{n} \mathbf{m}^i_{j,k} * \mathbf{F}^{i-1}_k$$

where $*$ is wide conv.

Our approach so far makes the features in row completely independent of the features in other rows. We'd like to inject some dependence, so we use an operation called folding, where we sum pairs of consecutive rows (cutting the number of rows from d to $\frac{d}{2}$).

6 Properties of the Sentence Model

Notice that our model can learn n-grams where $n \leq m$. It also can preserve order thanks to k-Max Pooling.

It also learns a DAG over the sentence. Basically, draw an edge from a node in a lower layer to a node in higher layer if the lower node is used in the conv operation for the higher node. Now, drop any node that isn't selected by k-Max Pooling. Finally, throw in a root node. The end result is a DAG. This is like a parse tree, but the network learned it instead of having an external system provide it.

7 Experiments

We make sure our network ends with a fully-connected layer and softmax. We minimize cross-entropy and use L2 regularization. The word embeddings, conv filters, and fully-connected layer weights are our parameters. We train with Adagrad (a variant of stochastic gradient descent). We implement convolutional with a Fast Fourier Transform (we can do this because the convolution is one-dimensional). We train on one GPU.

We train on binary and multiclass sentiment analysis on the Stanford Sentiment Treebank dataset (movie reviews). We beat NBoW and Max-TDNN. Our DCNN architecture is: wide conv ($m = 7, n = 6$), folding, dynamic k-max pooling, wide conv ($m = 5, n = 14$), folding, k-max pooling ($k = 4$). We use the tanh nonlinearity.

Next, we train on the TREC dataset, where the goal is to classify a question into one of six question types. Since the dataset is small, we use a word embedding size of $d = 32$. We get performance similar to the state of the art models, but we don't use complex hand-engineered features like they do.

Finally, we do Twitter sentiment analysis (where sentiment is given by the emoji in the tweet). This dataset is large (1.6M tweets), so we set $d = 60$. We set state of the art.

For each filter map in the first layer, we see which 7-grams triggers each map the most. We find that 7-grams with positive/negative words are highly triggering. Additionally, we have detectors for more complex 7-grams with "as...as", "with...that", etc structures.

8 Conclusion

Our Dynamic Convolutional Neural Network (DCNN) uses Dynamic k-Max Pooling to effectively model sentences, as demonstrated by great performance in multiple sentence classification tasks.