Natural Language Processing
Convolutional Neural Networks

Felipe Bravo-Marquez

October 21, 2019

Convolutional Neural Networks

- Convolutional neural networks (CNNs) became very popular in the computer vision community due to its success for detecting objects (“cat”,“bicycles”) regardless of its position in the image.
- They identify indicative local predictors in a structure (e.g., images, sentences).
- These predictors are combined to produce a fixed size vector representation for the structure.
- When used in NLP, the network captures the n-grams that are most informative for the target predictive task.
- For sentiment classification, these local aspects correspond to n-grams conveying sentiment (e.g., not bad, very good).
- The fundamental idea of CNNs [LeCun et al., 1998] is to consider feature extraction and classification as one jointly trained task.

Basic Convolution + Pooling

- Sentences are usually modeled as sequences of word embeddings.
- These embeddings can be obtained either from pre-trained word embeddings or from an embedding layer.
- The CNN applies nonlinear (learned) functions or “filters” mapping windows of k words into scalar values.
- Several filters can be applied, resulting in an l-dimensional vector (one dimension per filter).
- The filters capture relevant properties of the words in the window.
- These filters correspond to the “convolution layer” of the network.

Basic Convolution + Pooling

- The “pooling” layer is used to combine the vectors resulting from the different windows into a single *l*-dimensional vector.
- This is done by taking the max or the average value observed in each of the dimensions over the different windows.
- The goal is to capture the most important “features” in the sentence, regardless of the position.
- The resulting *l*-dimensional vector is then fed further into a network that is used for prediction (e.g., softmax).
- The gradients are propagated back from the network’s loss tuning the parameters of the filter.
- The filters learn to highlight the aspects of the data (n-grams) that are important for the target task.

Basic Convolution + Pooling

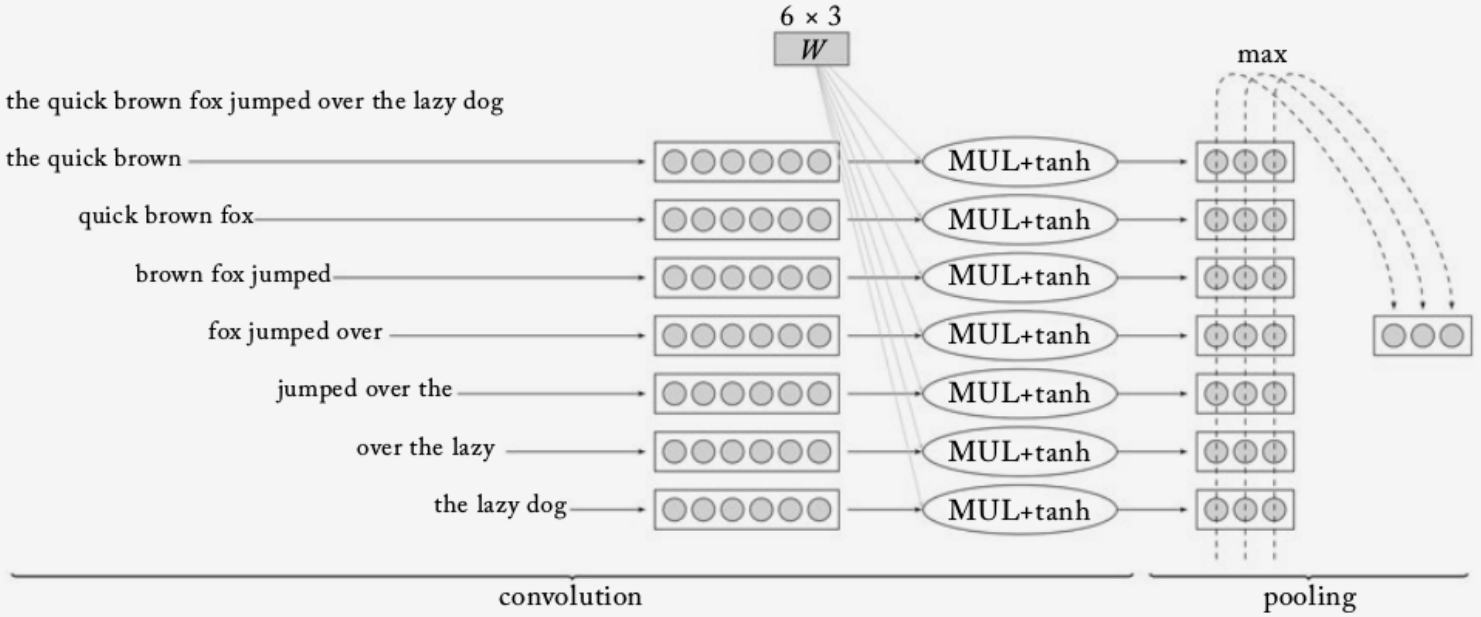


Figure 13.2: 1D convolution+pooling over the sentence "the quick brown fox jumped over the lazy dog." This is a narrow convolution (no padding is added to the sentence) with a window size of 3. Each word is translated to a 2-dim embedding vector (not shown). The embedding vectors are then concatenated, resulting in 6-dim window representations. Each of the seven windows is transfered through a 6×3 filter (linear transformation followed by element-wise tanh), resulting in seven 3-dimensional filtered representations. Then, a max-pooling operation is applied, taking the max over each dimension, resulting in a final 3-dimensional pooled vector.

[1]Source: [Goldberg, 2017]

1D Convolutions over Text

- We focus on the one-dimensional convolution operation[2].
- Consider a sequence of words $w_{1:n} = w_1, \ldots, w_n$ each with their corresponding d_{emb} dimensional word embedding $E_{[w_i]} = \vec{w}_i$.
- A 1D convolution of width k works by moving a sliding-window of size k over the sentence, and applying the same filter to each window in the sequence.
- A filter is a dot-product with a weight vector $\vec{u}$, which is often followed by a nonlinear activation function.

[2] 1D here refers to a convolution operating over 1-dimensional inputs such as sequences, as opposed to 2D convolutions which are applied to images.

1D Convolutions over Text

- Define the operator $\oplus(w_{i:i+k-1})$ to be the concatenation of the vectors $\vec{w}_i, \ldots, \vec{w}_{i+k-1}$.
- The concatenated vector of the i-th window is $\vec{x}_i = \oplus(w_{i:i+k-1}) = [\vec{w}_i; \vec{w}_{i+1}; \ldots; \vec{w}_{i+k-1}]$, $x_i \in \mathcal{R}^{k \cdot d_{emb}}$.
- We then apply the filter to each window vector resulting in scalar values $p_i = g(\vec{x}_i \cdot \vec{u})$. ($p_i \in \mathcal{R}$)
- It is customary to use l different filters, $\vec{u}_1, \ldots, \vec{u}_l$, which can be arranged into a matrix U, and a bias vector $\vec{b}$ is often added: $\vec{p}_i = g(\vec{x}_i \cdot U + \vec{b})$.
- Each vector $\vec{p}_i$ is a collection of l values that represent (or summarise) the i-th window ($\vec{p}_i \in \mathcal{R}^l$).
- Ideally, each dimension captures a different kind of indicative information.

Narrow vs. Wide Convolutions

- How many vectors $\vec{p}_i$ do we have?
- For a sentence of length n with a window of size k, there are $n - k + 1$ positions in which to start the sequence.
- We get $n - k + 1$ vectors $\vec{p}_{1:n-k+1}$.
- This approach is called **narrow convolution**.
- An alternative is to pad the sentence with $k - 1$ padding-words to each side, resulting in $n + k + 1$ vectors $\vec{p}_{1:n+k+1}$.
- This is called a **wide convolution**.

1D Convolutions over Text

- The main idea behind the convolution layer is to apply the same parameterized function over all k-grams in the sequence.
- This creates a sequence of m vectors, each representing a particular k-gram in the sequence.
- The representation is sensitive to the identity and order of the words within a k-gram.
- However, the same representation will be extracted for a k-gram regardless of its position within the sequence.

Vector Pooling

- Applying the convolution over the text results in m vectors $\vec{p}_{1:m}$, each $\vec{p}_i \in \mathcal{R}^l$.
- These vectors are then combined (pooled) into a single vector $c \in \mathcal{R}^l$ representing the entire sequence.
- Max pooling: this operator takes the maximum value across each dimension (most common pooling operation).

$$\vec{c}_{[j]} = \max_{1<i\leq m} \vec{p}_{i[j]} \quad \forall j \in [1, l]$$

  where $\vec{p}_{i[j]}$ denotes the j-th component of $\vec{p}_i$.
- Average Pooling (second most common): takes the average value of each index:

$$\vec{c} = \frac{1}{m} \sum_{i=1}^{m} \vec{p}_i$$

- Ideally, the vector $\vec{c}$ will capture the essence of the important information in the sequence.

Vector Pooling

- The nature of the important information that needs to be encoded in the vector $\vec{c}$ is task dependent.
- If we are performing sentiment classification, the essence are informative ngrams that indicate sentiment.
- If we are performing topic-classification, the essence are informative *n*-grams that indicate a particular topic.
- During training, the vector $\vec{c}$ is fed into downstream network layers (i.e., an MLP), culminating in an output layer which is used for prediction.
- The training procedure of the network calculates the loss with respect to the prediction task, and the error gradients are propagated all the way back through the pooling and convolution layers, as well as the embedding layers.
- The training process tunes the convolution matrix U, the bias vector $\vec{b}$, the downstream network, and potentially also the embeddings matrix E[3] such that the vector $\vec{c}$ resulting from the convolution and pooling process indeed encodes information relevant to the task at hand.

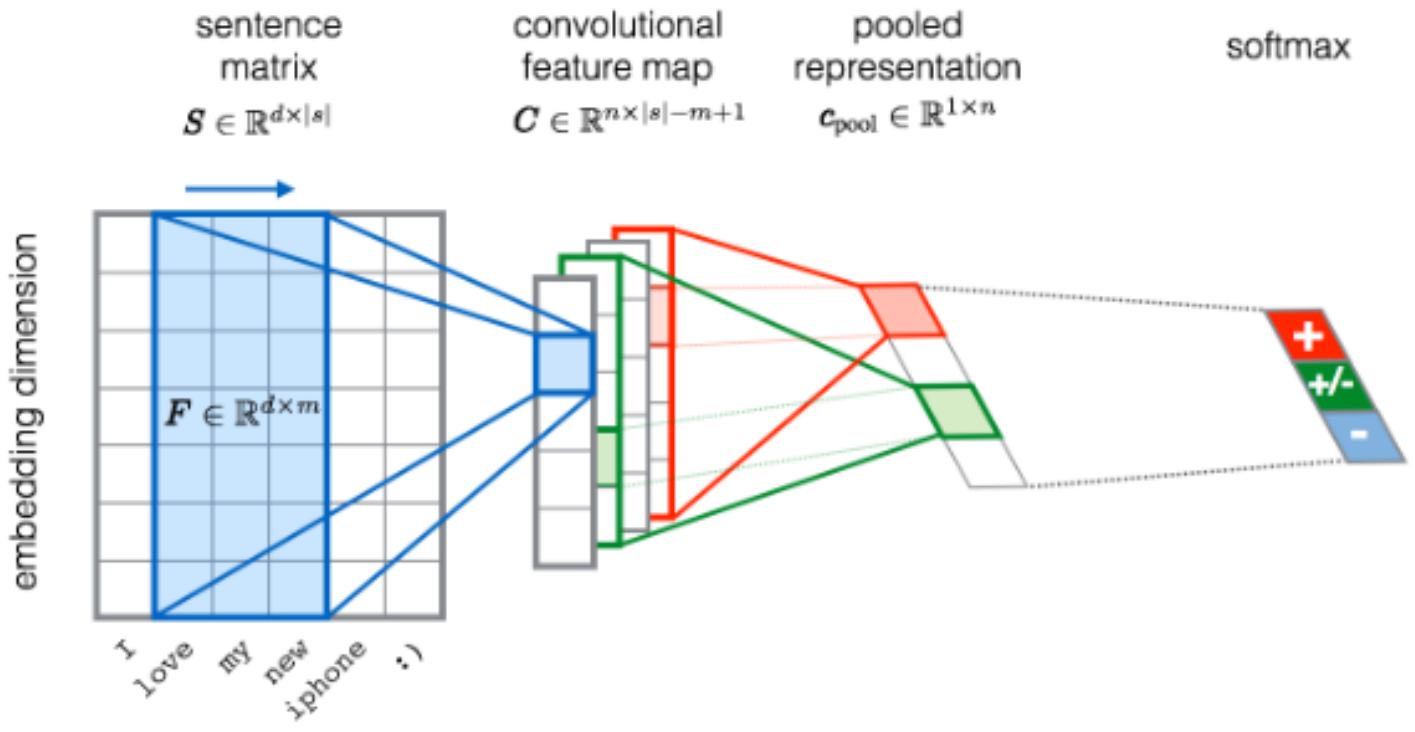
[3] While some people leave the embedding layer fixed during training, others allow the parameters to change.

Twitter Sentiment Classification with CNN

- A convolutional neural network architecture for Twitter sentiment classification is developed in [Severyn and Moschitti, 2015].
- Each tweet is represented as a matrix whose columns correspond to the words in the tweet, preserving the order in which they occur.
- The words are represented by dense vectors or embeddings trained from a large corpus of unlabeled tweets using word2vec.
- The network is formed by the following layers: an input layer with the given tweet matrix, a single convolutional layer, a rectified linear activation function, a max pooling layer, and a softmax classification layer.

Twitter Sentiment Classification with CNN

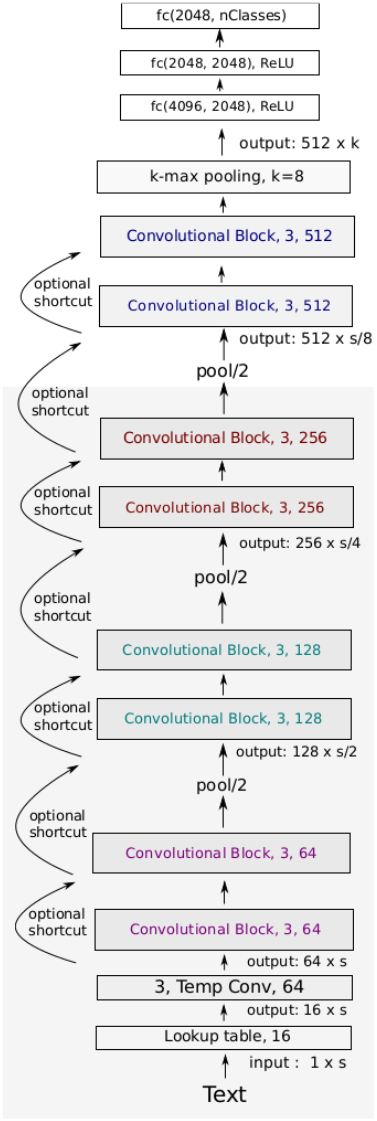
- The weights of the neural network are pre-trained using emoticon-annotated data, and then trained with the hand-annotated tweets from the SemEval competition.
- Experimental results show that the pre-training phase allows for a proper initialization of the network's weights, and hence, has a positive impact on classification accuracy.



Very Deep Convolutional Networks for Text Classification

- CNNs architectures for NLP are rather shallow in comparison to the deep convolutional networks which have pushed the state-of-the-art in computer vision.
- A text processing neural architecture (VDCNN) that operates directly at the character level and uses only small convolutions and pooling operations is proposed in [Conneau et al., 2017].
- Character-level embeddings are used instead of word-embeddings.
- Characters are the lowest atomic representation of text.
- The performance of this model increases with depth: using up to 29 convolutional layers, authors report improvements over the state-of-the-art on several public text classification tasks.
- Most notorious improvements are achieved on large datasets.
- One of the first words showing the the benefits of depth neural architectures for NLP.

Very Deep Convolutional Networks for Text Classification



Questions?

Thanks for your Attention!

References I


Conneau, A., Schwenk, H., Barrault, L., and Lecun, Y. (2017).
Very deep convolutional networks for text classification.
In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, volume 1, pages 1107–1116.


Goldberg, Y. (2017).
Neural network methods for natural language processing.
Synthesis Lectures on Human Language Technologies, 10(1):1–309.


LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998).
Gradient-based learning applied to document recognition.
Proceedings of the IEEE, 86(11):2278–2324.


Severyn, A. and Moschitti, A. (2015).
Twitter sentiment analysis with deep convolutional neural networks.
In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 959–962, New York, NY, USA. ACM.