Assignment 3: Project Preparation
Sentiment Analysis

Thora Daneyko

July 28, 2017

1 Previous work

1.1 Handling syntactic information

For sentiment analysis, especially for single sentences or short messages such as tweets, it is not really sufficient to simply count the occurrences of tokens associated with a certain emotion. Negation and conjunctions such as 'but' can inverse the polarity of a word or phrase and adverbs may weaken or strengthen the intensity of the emotions conveyed by an expression. Therefore, it is also important to take the syntactic structure of the text to be classified into account, especially the way in which emotional words or phrases are embedded into the sentence and relate to each other.

Because of this, many neural network implementations rely on parse trees to account for the syntactic structure of each sentence. Socher et al. (2013), for instance, feed these to a Recursive Neural Tensor Network (RNTN) which extracts the sentiment of a sentence by recursively combining the polarities of smaller constituents. However, due to its recursive nature, this model has a time complexity of at least $\mathcal{O}(n^2)$ and will thus be very slow on longer sentences. Also, one might not always have matching parse trees at hand. Luckily, it turns out that simpler non-recursive networks, namely Convolutional Neural Networks (CNNs), are able to extract syntactic information on their own.

1.1.1 Feed-Forward CNN

Such a network was proposed by Dos Santos and Gatti (2014). They design a feed-forward network with two convolutional layers. Words are represented as embeddings on two different levels: Once as a regular word-level embedding, which is pre-trained using `word2vec`, to capture their syntactic and semantic properties, and once as a character-level embedding to capture morphological similarities. These will mark informative parts of speech such as adverbs, and can also help to decompose hash tags containing multiple words such as `#SoSad` or `#ILikeIt`. Since words differ in their length, the character-level vector is obtained via the first convolutional layer, where the most relevant features of the variable length character embeddings are extracted using max-pooling. The

final representation of a word is then the concatenation of its word-level and character-level embedding. The second convolutional layer is used to derive a representation of a sentence. Similarly to how they obtain the character-level embedding, the vector representations of the words of the sentence are fed to the convolutional layer which combines the most relevant features into a sentence vector.

This sentence vector is then input to a regular feed-forward neural network with one hidden layer applying the hyperbolic tangent as an activation function. A softmax operation is applied to the resulting scores to obtain negative log-likelihoods for the tags. This likelihood is then minimized using stochastic gradient descent.

The resulting network outperforms state-of-the-art networks such as the RNTN of Socher et al. (2013) on both binary (85.7 % acc.) and more fine-grained (48.3 % acc.) sentiment distinctions on the Stanford Sentiment Treebank. For classifying Twitter data, the character-level embeddings provide an accuracy improvement of 1.2 percentage points, yielding 86.4 % overall accuracy (compared to 85.2 % for word-level embeddings only). Investigation of the local features selected by the sentence-level convolutional layer reveal that the network is sensitive to negation. These results indicate that explicit syntactic information is not necessary for handling negation and that using character embeddings in addition to word embeddings is beneficial for classifying Twitter data.

1.1.2 Recurrent CNN

Usually, the convolutional layer considers fixed size windows around all words in a sequence (e.g. a sentence) from which it extracts the most relevant features. In this case, the window size has a significant effect on the effectiveness of the method, as Lai et al. (2015) point out: "[S]mall window sizes may result in the loss of some critical information, whereas large windows result in an enormous parameter space". Recurrent Neural Networks (RNNs), on the other hand, are able to take all surrounding context into account, not just a window of fixed size, and could thus be able to retain more of the important information that might get lost in traditional CNNs. Hence, Lai et al. (2015) propose a new model, the Recurrent CNN (RCNN) which combines the recurrent layers of an RNN with the convolutional layers of the CNN, and test it for a range of classification tasks, amongst others sentiment analysis.

All word representations are initialized to word embeddings pre-trained using the skip-gram model. For each word in the sentence, the words left to it and the words right to it are fed to a recurrent layer with hyperbolic tangent activation to extract a representation of the left and right context, respectively. The representation of a word then is the concatenation of its left context, its own embedding and its right context. The resulting vector is fed to another linear layer and transformed by another tanh function. In this way, each word representation also stores information about all surrounding words.

To obtain a fixed size representation for the whole sentence, the context-sensitive word vectors of all words in that sentence are then fed to a max-pooling layer,

just as in Dos Santos and Gatti's (2014) model, to extract the most informative features. From the resulting sentence vector, another linear layer extracts the label scores, which are then transformed into probabilities by a softmax function. These log-likelihoods are maximized during training by stochastic gradient descent.

Their model was tested on four different datasets with four different classification tasks. One of these was the Stanford Sentiment Treebank containing movie reviews labeled with five sentiment distinctions. This corresponds to the "fine-grained" distinctions also evaluated by Dos Santos and Gatti (2014). On these data, they reached an F1 score of 47.21, which is worse than the best reported state-of-the-art system (48.70) and Dos Santos and Gatti (2014) (48.30), but better than the parse-tree dependent system of Socher et al. (2013). It also improves on the performance of a non-recurrent CNN (46.35) which indicates that the recurrent layer is beneficial for this task, perhaps due to better capturing the contextual information than a window-based convolutional approach. When experimenting with different window sizes of a regular CNN on the classification of the 20Newsgroups dataset, they also observed that the performance of the CNN depended heavily on the context window size.

1.2 Sentiment-specific word embeddings

Word embeddings such as those produced by `word2vec` usually capture syntactic and semantic information, such that words that are used in similar syntactic constructions and can be found in similar semantic contexts are close to each other in the vector space. For sentiment analysis, this is not very helpful, since opposite polarity words such as "good" and "bad" will have very similar representations, since they occur in the same constructions and contexts. However, they hint at completely different sentiments. Because of this, Tang et al. (2014) have developed a method to extract sentiment-specific word embeddings (SSWE). Neural networks that use SSWEs as word representations achieve a higher F1 score than all state-of-the-art sentiment analysis systems, and the same network based on SSWEs performs much better than when using `word2vec` or comparable embeddings. To capture syntactic modifications such as negation, they also create bigram and trigram embeddings, which consistently improves the strength of their SSWEs. Overall, their system reaches 84.98 F1 score without additional features and 86.48 when enriched with the handcrafted features used by many of the state-of-the-art systems.

2 Model outline

Since I find it hard to estimate how time-consuming the implementation is going to be[1], I propose a basic version of my model as well as several additions and modifications that I will attempt to include once my basic model is working. Figure 1 shows an overview over the envisioned components.

[1] Or, as Kurt Eberle likes to point out on each of his slides: "Take into account: Hofstadter's Law: It always takes longer than you expect, even when you take into account Hofstadter's Law".

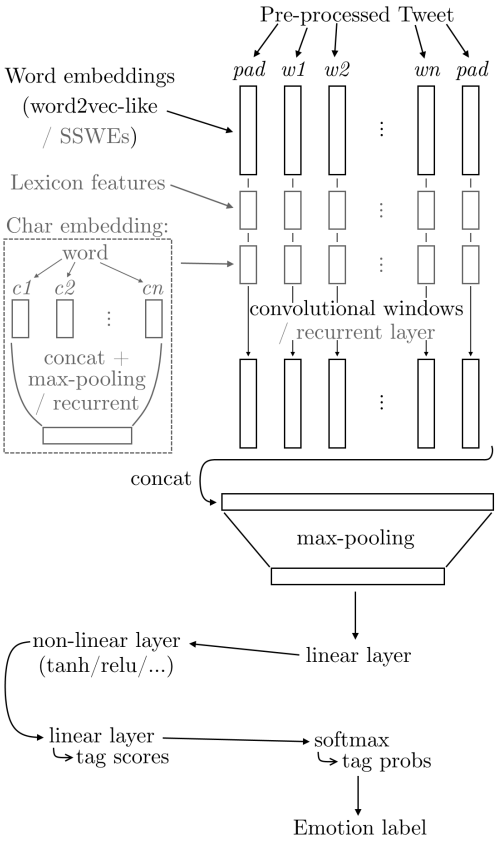

Figure 1: The maximal architecture of my model. Grayed out parts are optional additions/modifications.

2.1 Basic pre-processing

Before considering any model building, however, I need to outline some basic pre-processing steps. The Tweets come as raw, unprocessed sentences, so they first need to be tokenized. NLTK is the natural choice for this, since it also includes a specialized tokenizer for Tweets that can handle emoticons and hash tags. I would furthermore like to convert all non-hash-tag tokens to lower-case to keep the amount of words low. For hash tags, it could be beneficial to keep the capitalization to give the model the opportunity to learn the compositional nature of hash tags such as `#SoSad` or `#ILikeIt` itself. Since writing in all-caps is an indicator for the emotion anger, I will add a feature that encodes whether the original word was in all-caps or not to the word representation. Other types of capitalization should not matter much for emotion detection.

To represent these words in the model, I will initially just use regular word embeddings as they are produced by `word2vec` and similar algorithms, i.e. those that capture syntactic and contextual similarity, because these are more wi-

despread and thus easier to obtain or train. The resulting word vectors can of course be enhanced by additional features such as the all-caps feature via concatenation.

2.2 Basic model structure

The literature agrees that a convolutional approach is the best way to deal with sentiment analysis in deep learning. Therefore, my first step would be to implement a simple feed-forward CNN similar to that proposed by Dos Santos and Gatti (2014).

Since we want to classify Tweets and not single words, we need to get a Tweet representation from our individual word vectors. Hence, we will pass the vector representations of the words in the Tweet through a simple window-based convolutional layer (for this, we need to add some padding to our Tweet) to first obtain context-sensitive representations of the single words and then get a representation of the whole Tweet with its most important features by applying max-pooling on the concatenation of these word representations.

The resulting Tweet vector can then be fed to a linear layer which produces the input to our non-linear hidden layer. I will experiment with different activation functions, starting out with the hyperbolic tangent since this is the one used in the literature presented above. The output of this hidden layer is then handed to a second linear layer that produces the output scores for the six emotion categories given the current Tweet. I then apply softmax to get the tag probabilities and label the Tweet with the winning emotion.

2.3 Enhancements

This model can be enhanced in several ways.

First, to capture the hash tags, I will try to enrich my word representations with character-level embeddings run through a max-pooling layer, since this has proven to boost performance on Tweet classification (Dos Santos and Gatti 2014).

After that, I can hopefully further improve my model by replacing part of its convolutional layers by a recurrent structure. For the character-level embeddings, I could simply run a bi-directional recurrent unit over the individual characters of a word, as we did in the second assignment. Also, the convolutional windows that extract context-sensitive word vectors can be replaced by the recurrent system proposed by Lai et al. (2015) to get a richer context representation.

In a final step, I could attempt to replace my word embeddings by the SSWEs presented by Tang et al. (2014). In their paper, they note that they "release the sentiment-specific word embedding learned from 10 million tweets, which can be adopted off-the-shell in other sentiment analysis tasks", however, I could not find them on the web, otherwise I could start out directly with these instead of first training my own embeddings or downloading regular embeddings from the

web. There seems to be a Python implementation of their algorithm by Attardi (2015), so maybe I could try to create the SSWEs myself.

During any step, I can probably further improve the performance by including emotion lexicons such as the NRC Word-Emotion Association Lexicon (S. Mohammad 2010) and the NRC Hashtag Emotion Lexicon (S. M. Mohammad and Kiritchenko 2015), adding discrete `true/false` features to my word representations encoding whether it is listed for one of the six emotion categories in these lexicons.

References

Attardi, Giuseppe (2015). *Sentiment Specific Word Embeddings*. URL: `https://github.com/attardi/deepnl/wiki/Sentiment-Specific-Word-Embeddings`.

Dos Santos, Cícero Nogueira and Maira Gatti (2014). "Deep Convolutional Neural Networks for Sentiment Analysis of Short Texts." In: *COLING*, pp. 69–78.

Lai, Siwei, Liheng Xu, Kang Liu, and Jun Zhao (2015). "Recurrent Convolutional Neural Networks for Text Classification." In: *AAAI*. Vol. 333, pp. 2267–2273.

Mohammad, Saif (2010). *NRC Word-Emotion Association Lexicon*. URL: `http://saifmohammad.com/WebPages/NRC-Emotion-Lexicon.htm`.

Mohammad, Saif M. and Svetlana Kiritchenko (2015). "Using Hashtags to Capture Fine Emotion Categories from Tweets". In: *Computational Intelligence* 31.2, pp. 301–326.

Socher, Richard, Alex Perelygin, Jean Y. Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts (2013). "Recursive deep models for semantic compositionality over a sentiment treebank". In: *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*. Vol. 1631, p. 1642.

Tang, Duyu, Furu Wei, Nan Yang, Ming Zhou, Ting Liu, and Bing Qin (2014). "Learning Sentiment-Specific Word Embedding for Twitter Sentiment Classification." In: *ACL (1)*, pp. 1555–1565.