ТЕКСТОВЫЙ КЛАССИФИКАТОР НА ОСНОВЕ ИЕРАРХИЧЕСКИХ СЕТЕЙ СО ВНИМАНИЕМ ДЛЯ КЛАССИФИКАЦИИ ДОКУМЕНТОВ
TEXT CLASSIFIER FOR HIERARCHICAL ATTENTION NETWORKS FOR DOCUMENT CLASSIFICATION

Kotlyarova Ekaterina (kotlyarova.ev@phystech.edu)
State Research Institute of Aviation System, Moscow, Russia

Abstract

Text classification is a classical problem. The goal is to classify documents into a fixed number of predefined categories, given a variable length of text bodies. It is widely used in in such fields and applications as sentimental analysis (IMDB, YELP reviews classification), stock market sentimental analysis, smart reply used by GOOGLE. This is a very active research area both in academia and industry. In this work, the comparison of three neural networks — CNN, RNN and Hierarchical Attention Networks — is done.

Keywords: text classifier, convolutional neural network, hierarchical attention network, lstm

1 Inroduction

Firstly, this work is based on paper "Hierarchical Attention Networks for Document Classification", which is written by Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola1, Eduard Hovy, 2016. Secondly, on paper "Convolutional Neural Networks for Sentence Classification", by Yoon Kim, 2014. And thirdly bidirectional LSTM and one level attentional RNN from "Feed-forward networks with attention can solve some long-term memory problems" (Colin Raffel, 2016) is used in this review.

2 Dataset

As the dataset is used one from the Kaggle tutorial competition "Bag of Words Meets Bags of Popcorn". The labeled data set consists of 50,000 IMDB movie reviews, specially selected for sentiment analysis. The sentiment of reviews is binary, meaning the IMDB rating less than 5 results in a sentiment score of 0, and rating upwards of 7 have a sentiment score of 1. No individual movie has more than 30 reviews. The 25,000 review labeled training set does not include any of the same movies as the 25,000 review test set. In addition, there are another 50,000 IMDB reviews provided without any rating labels.

3 Convolutional Neural Networks for Sentence Classification

In the present work, a simple CNN was trained. It consisted of one layer of convolution on top of word vectors obtained from an unsupervised neural language model. Despite little tuning of hyperparameters, this simple model achieves good results.

The model architecture, shown in figure 1, is a slight variant of the CNN architecture of Collobert et al. (2011). Let $x_i \in \mathrm{R}$ be a k-dimensional word vector corresponding to the i-t word in the sentence. A sentence of length n (padded where necessary) is represented as:

$$x_{1:n} = x_1 \oplus x_2 ... \oplus x_n \tag{1}$$

where $\oplus$ is the concatenation operator. In general, let $x_{i:i+j}$ refer to the concatenation of words x_i, x_{i+1}, ..., x_{i+j}. A convolution operation involves a filter $\mathrm{w} \in R^{hk}$, which is applied to a window of h words to produce a new feature. For example, a feature c_i is generated from a window of words $x_{i:i+h-1}$ by:

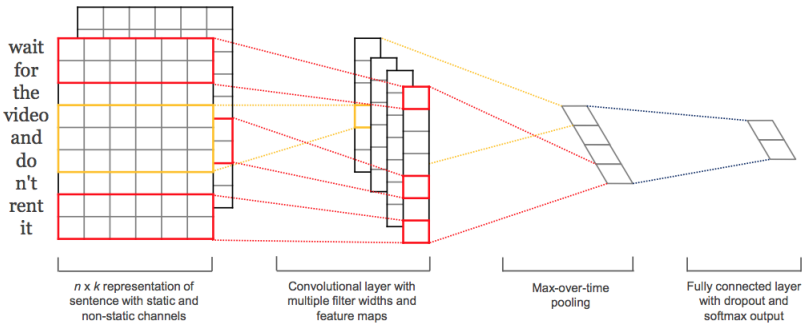

Figure 1: Model architecture with two channels for an example sentence (cf. (Yoon Kim, 2014) Figure 1)

$$c = f(w \cdot x_{i:i+h-1} + b) \tag{2}$$

with c $\in R^{n-h+1}$. Then apply a max-overtime pooling operation (Collobert et al., 2011) over the feature map and take the maximum value $\widehat{c} = \max\{c\}$ The idea is to capture the most important feature one with the highest value for each feature map. This pooling scheme naturally deals with variable sentence lengths.

The process can be described by specific feature extracted from one filter. The model uses multiple filters (with varying window sizes) to obtain multiple features. These features form the penultimate layer and are passed to a fully connected softmax layer whose output is the probability distribution over labels.

3.1 My implementation

First, a very simple convolutional architecture was used here. Simply use total 128 filters with size 5 and max pooling of 5 and 35. Here's how was solved the classification problem:

1. Convert all text samples in the dataset into sequences of word indices. A "word index" would simply be an integer ID for the word. We will only consider the top 20,000 most commonly occuring words in the dataset, and we will truncate the sequences to a maximum length of 1000 words.

2. Prepare an "embedding matrix" which will contain at index i the embedding vector for the word of index i in our word index.

3. Load this embedding matrix into a Keras Embedding layer, set to be frozen (its weights, the embedding vectors, will not be updated during training).

4. Build on top of it a 1D convolutional neural network, ending in a softmax output over our categories.

The accuracy achieved is 89 percent. Deeper convolutional neural network was investigated in Yoon Kim's paper [ref] with multiple filters applied. This can be easily implemented using Keras Merge Layer. Thus the result can be improved to 90.3 percent. Both architectures are represented on Figure 2.

4 Bidirectional LSTM and one level attentional RNN

4.1 Bidirectional LSTM

By using LSTM encoder, we intent to encode all information of the text in the last output of recurrent neural network before running feed forward network for classification. This is very similar to neural translation machine and sequence to sequence learning. See the Figure 3:

model fitting - simplified convolutional neural network

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 1000)	0
embedding_1 (Embedding)	(None, 1000, 100)	8410000
conv1d_1 (Conv1D)	(None, 996, 128)	64128
max_pooling1d_1 (MaxPooling1	(None, 199, 128)	0
conv1d_2 (Conv1D)	(None, 195, 128)	82048
max_pooling1d_2 (MaxPooling1	(None, 39, 128)	0
conv1d_3 (Conv1D)	(None, 35, 128)	82048
max_pooling1d_3 (MaxPooling1	(None, 1, 128)	0
flatten_1 (Flatten)	(None, 128)	0
dense_1 (Dense)	(None, 128)	16512
dense_2 (Dense)	(None, 2)	258

(a) A simplified Convolutional

model fitting - more complex convolutional neural network

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 1000)	0	
embedding_1 (Embedding)	(None, 1000, 100)	8410000	input_1[0][0]
conv1d_1 (Conv1D)	(None, 998, 128)	38528	embedding_1[0][0]
conv1d_2 (Conv1D)	(None, 997, 128)	51328	embedding_1[0][0]
conv1d_3 (Conv1D)	(None, 996, 128)	64128	embedding_1[0][0]
max_pooling1d_1 (MaxPooling1D)	(None, 199, 128)	0	conv1d_1[0][0]
max_pooling1d_2 (MaxPooling1D)	(None, 199, 128)	0	conv1d_2[0][0]
max_pooling1d_3 (MaxPooling1D)	(None, 199, 128)	0	conv1d_3[0][0]
merge_1 (Merge)	(None, 597, 128)	0	max_pooling1d_1[0][0] max_pooling1d_2[0][0] max_pooling1d_3[0][0]
conv1d_4 (Conv1D)	(None, 593, 128)	82048	merge_1[0][0]
max_pooling1d_4 (MaxPooling1D)	(None, 118, 128)	0	conv1d_4[0][0]
conv1d_5 (Conv1D)	(None, 114, 128)	82048	max_pooling1d_4[0][0]
max_pooling1d_5 (MaxPooling1D)	(None, 3, 128)	0	conv1d_5[0][0]
flatten_1 (Flatten)	(None, 384)	0	max_pooling1d_5[0][0]
dense_1 (Dense)	(None, 128)	49280	flatten_1[0][0]
dense_2 (Dense)	(None, 2)	258	dense_1[0][0]

(b) A deeper Convolutional

Figure 2: Implementation of convolutional architectures

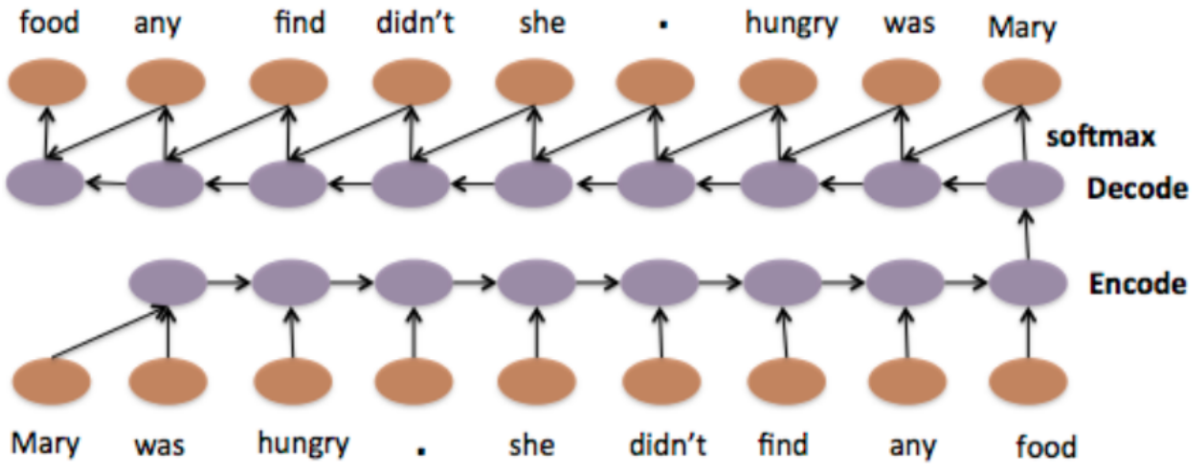

Figure 3: Standart Sequence to Sequence Model (cf. (Jiwei Li, 2015) Figure 1)

And now a quick overview of LSTM models. LSTM models (Hochreiter and Schmidhuber, 1997) are defined as follows: given a sequence of inputs $X = \{x_1, x_2, ..., x_{n_x}\}$, an LSTM associates each timestep with an input, memory and output gate, respectively denoted as i_t, f_t, and o_t. For notations, we disambiguate e and h where e_t denote the vector for individual text unite (e.g., word or sentence) at time step t while h_t denotes the vector computed by LSTM model at time t by combining e_t and h_{t-1}. The vector representation h_t for each time-step t is given by:

$$\begin{bmatrix} i_t \\ f_t \\ o_t \\ l_t \end{bmatrix} = \begin{bmatrix} \sigma \\ \sigma \\ \sigma \\ tanh \end{bmatrix} W \cdot \begin{bmatrix} h_{t-1} \\ e_t \end{bmatrix} \tag{3}$$

$$c_t = f_t \cdot c_{t-1} + i_t \cdot l_t \tag{4}$$

$$h_t^s = o_t \cdot c_t \tag{5}$$

where $W \in R^{4K\ times 2K}$ In sequence-to-sequence generation tasks, each input X is paired with a sequence of outputs to predict: $Y = \{y_1, y_2, ..., y_{n_y}\}$. An LSTM defines a distribution over outputs and sequentially predicts tokens using a softmax function:

$$P(Y|X) = \prod_{t \in [1, n_y]} p(y_t | x_1, x_2, ..., x_t, y_1, y_2, ..., y_{t-1}) = \prod_{t \in [1, n_y]} \frac{\exp f(h_{t-1}, e_{y_t})}{\sum_{y'} \exp(f(h_{t-1}, e_{y'}))} \quad (6)$$

$f(h_{t-1}, e_{y_t})$ denotes the activation between e_{h-1} and e_{y_t} where h_{t-1} is the representation outputted from LSTM at time t-1. Note that each sentence ends up with a special sympol <end>. Commonly, the input and output use two different sets of convolution parametres for capturing different compositional patterns.

In the decoding procedure, the algorithm terminates when an $< end >$ token is predicted. At each timestep, either a greedy approach or beam search can be adopted for word prediction. Greedy search selects the token with the largest conditional probability, the embedding of which is then combined with preceding output for next step token prediction. For beam search, (Sutskever et al., 2014) discovered that a beam size of 2 suffices to provide most of benefits of beam search.

4.2 One level attentional RNN

In the following, author is planning to implement an attention layer which is well studied in many papers including "Neural machine translation by joinly learning to alogn to translate" (Dzmitry Bahdanau, 2015). Particularly for this text classification task, was followed the implementation of "Feed-forward networks with attention can solve some long-term memory problems" (Colin Raffel, 2016).

Firstly, a overview of "attention". A recently proposed method for easier modeling of long-term dependencies is “attention”. Attention mechanisms allow for a more direct dependence between the state of the model at different points in time. Following the definition from (Bahdanau et al., 2014), given a model which produces a hidden state h_t, at each time step, attention-based models compute a “context” vector ct as the weighted mean of the state sequence h by

$$c_t = \sum_{j=1}^{T} a_{tj} h_j \quad (7)$$

where T is the total number of time steps in the input sequence and tj is a weight computed at each time step t for each state h_j. These context vectors are then used to compute a new state sequence s, where s_{t-1}, c_t and the model output at t-1. The weightings a_{tj} are then computed by

$$e_{tj} = a(s_{t-1}, h_j), a_{tj} = \frac{\exp e_{tj}}{\sum_{k=1}^{T} \exp e_{tk}} \quad (8)$$

where a is a learned function which can be thought of as computing a scalar importance value for h_j given the value of h_j and the previous state s_{t1}. This formulation allows the new state sequence s to have more direct access to the entire state sequence h. Attention-based RNNs have proven effective in a variety of sequence transduction tasks, including machine translation (Bahdanau et al., 2014), image captioning (Xu et al., 2015), and speech recognition (Chan et al., 2015; Bahdanau et al., 2015). Attention can be seen as analogous to the “soft addressing” mechanisms of the recently proposed Neural Turing Machine (Graves et al., 2014) and End-To-End Memory Network (Sukhbaatar et al., 2015) models.

4.3 My implementation

See the Figure 3. Was used LSTM layer in Keras to implement this. Other than forward LSTM, here author is planning to use bidirectional LSTM and concatenate both last output of LSTM outputs. Keras has provide a very nice wrapper called bidirectional. The best peformance is about 90.4 percent, the architecture on Figure 4 (a).

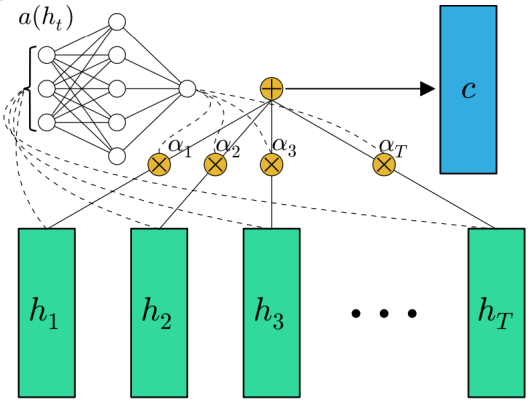

Figure 4: Schematic of our proposed "feed-forward" attention mechanism (cf. (Cho, 2015) Figure 1). Vectors in the hidden state sequence ht are fed into the learnable function $a(h_t)$ to produce a probability vector α. The vector c is computed as a weighted average of h_t, with weighting given by α

model fitting - Bidirectional LSTM

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 1000)	0
embedding_1 (Embedding)	(None, 1000, 100)	8410000
bidirectional_1 (Bidirection	(None, 200)	160800
dense_1 (Dense)	(None, 2)	402

(a) Text classification using LSTM

model fitting - simplified convolutional neural network

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 1000)	0
embedding_1 (Embedding)	(None, 1000, 100)	8410000
conv1d_1 (Conv1D)	(None, 996, 128)	64128
max_pooling1d_1 (MaxPooling1	(None, 199, 128)	0
conv1d_2 (Conv1D)	(None, 195, 128)	82048
max_pooling1d_2 (MaxPooling1	(None, 39, 128)	0
conv1d_3 (Conv1D)	(None, 35, 128)	82048
max_pooling1d_3 (MaxPooling1	(None, 1, 128)	0
flatten_1 (Flatten)	(None, 128)	0
dense_1 (Dense)	(None, 128)	16512
dense_2 (Dense)	(None, 2)	258

(b) Attention Network

Figure 5: My implementation of LSTM and sentence level Attentional RNN architectures

In the following, author is planning to implement an attention layer which is well studied in many papers. Particularly for this text classification task, was followed the implementation of "Feed-forward networks with attention can solve some long-term memory problems" by Colin Raffel. To implement the attention layer, need to build a custom Keras layer. Everyone can read how to do it in Keras documentation. Then following code is pretty much the same as the previous one (LSTM) except an attention layer on top of Output. See the layers on Figure 4 (b).

5 Hierarchical Attention Networks

The overall architecture of the Hierarchical Attention Network (HAN) is shown in Fig. 4. It consists of several parts: a word sequence encoder, a word-level attention layer, a sentence encoder and a sentence-level attention layer. The details of attention layer are described in the following section.

5.1 Hierarchical Attention

Assume that a document has L sentences s_i and each contains T_i words. W_{it} with $t \in [1, T]$ represents the words in the ith sentence. The proposed model projects the raw document into a vector representation, on which we build a classifier to perform document classification. In the following, will present how build the document level vector progressively from word vectors by using the hierarchical structure.

Word Encoder Given a sentence with words $W_{it}, t \in [0, T]$, we first embed the words to vectors through an embedding matrix $W_e, x_{ij} = W_e w_{ij}$. Then use bidirectional GRU (Bahdanau et al.,

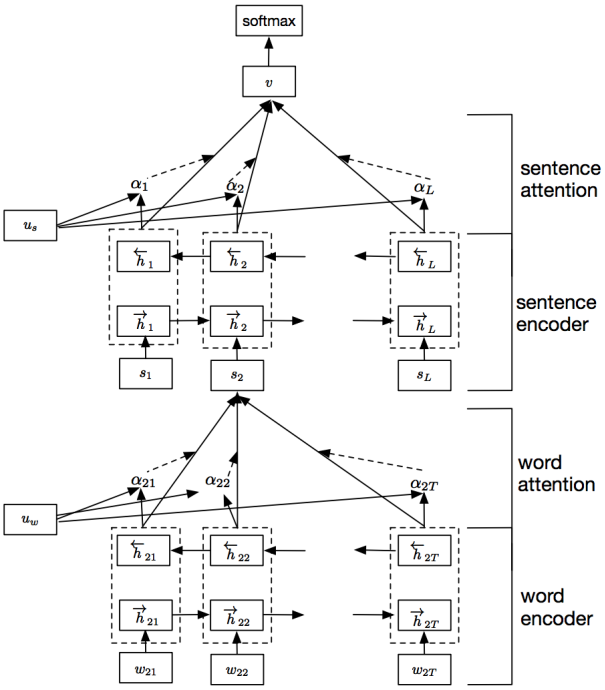

Figure 6: Hierarchical Attention Network (cf. (Zichao Yang, 2016) Figure 2).

2014) to get annotations of words by summarizing information from both directions for words, and therefore incorporate the contextual information in the annotation.

Word Attention Not all words contribute equally to the representation of the sentence meaning. Hence, introduce attention mechanism to extract such words that are important to the meaning of the sentence and aggregate the representation of those informative words to form a sentence vector. Specifically,

$$u_{it} = \tanh(W_w h_{it} + b_w) \tag{9}$$

$$\alpha_{it} = \frac{\exp(u_{it}^T u_w)}{\sum_t \exp(u_{it}^T u_w)} \tag{10}$$

$$s_i = \sum_t \alpha_{it} h_{it} \tag{11}$$

That is, first feed the word annotation hit through a one-layer MLP to get u_{it} as a hidden representation of hit, then measure the importance of the word as the similarity of u_{it} with a word level context vector u_w and get a normalized importance weight it through a softmax function. After that, compute the sentence vector s_i (we abuse the notation here) as a weighted sum of the word annotations based on the weights. The context vector u_w can be seen as a high level representation of a fixed query "what is the informative word" over the words like that used in memory networks (Sukhbaatar et al., 2015; Kumar et al., 2015). The word context vector u_w is randomly initialized and jointly learned during the training process.

Sentence Encoder Given the sentence vectors s_i, can get a document vector in a similar way. In this work used a bidirectional GRU to encode the sentences.

Sentence Attention To reward sentences that are clues to correctly classify a document, again used attention mechanism and introduce a sentence level context vector us and use the vector to measure the importance of the sentences. This yields

$$u_{it} = \tanh(W_s h_i + b_s) \tag{12}$$

Results List	
Type of neural network	Results of my implementation
Simple Convolutional Network	89 %
Deeper Convolutional Network	90.3 %
Bidirectional LSTM	90.4 %
One level attentional RNN	90.4 %

Table 1: Results.

$$\alpha_{it} = \frac{\exp(u_{it}^T u_s)}{\sum_i \exp(u_i^T u_s)} \tag{13}$$

$$u = \sum_i \alpha_i h_i \tag{14}$$

5.2 My implementation

Like in previous parts, was needed to build a custom Keras layer. Following the paper, Hierarchical Attention Networks for Document Classification (Zichao Yang, 2016), author also added a dense layer taking the output from GRU before feeding into attention layer. In the following implementation (Figure 6), there're two layers of attention network built in, one at sentence level and the other at review level. The best performance is pretty much still cap at 90.4 percent. Look at implemetation on Figure 7.

```
model fitting - Hierachical LSTM
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         (None, 15, 100)           0
_________________________________________________________________
time_distributed_1 (TimeDist (None, 15, 200)           8570800
_________________________________________________________________
bidirectional_2 (Bidirection (None, 200)               240800
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 402
=================================================================
```

Figure 7: My implementation of of hierarchical attention network.

6 Conclusion

The result is a bit disappointing. It was not possible to achieve a better accuracy although the training time is much faster, comparing to different approaches from using convolutional, bidirectional RNN, to one level attention network. Maybe the dataset is too small. However, given the potential power of explaining the importance of words and sentences, Hierarchical attention network could have the potential to be the best text classification method.

Results at Table 1.

References

Yoon Kim. Convolutional Neural Networks for Sentence Classification. New York University, 2014.

Colin Raffel, Daniel P. W. Ellis. Feed-forward networks with attention can solve some long-term memory problems. LabROSA, Columbia University, 2016.

Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, Eduard Hovy. Hierarchical Attention Networks for Document Classification. Carnegie Mellon University and Microsoft Research, Redmond, 2016.

Jiwei Li, Minh-Thang Luong and Dan Jurafsky. Hierarchical Neural Autoencoder for Paragraphs and Documents. Computer Science Department, Stanford University, Stanford, CA 94305, USA, 2015.

Keras documentation.
https://keras.io/layers/writing-your-own-keras-layers/