DataStories at SemEval-2017 Task 6: Siamese LSTM with Attention for Humorous Text Comparison

Christos Baziotis, Nikos Pelekis, Christos Doulkeridis
University of Piraeus - Data Science Lab
Piraeus, Greece
mpsp14057@unipi.gr, npelekis@unipi.gr, cdoulk@unipi.gr

Abstract

In this paper we present a deep-learning system that competed at SemEval-2017 Task 6 "#HashtagWars: Learning a Sense of Humor". We participated in Subtask A, in which the goal was, given two Twitter messages, to identify which one is funnier. We propose a Siamese architecture with bidirectional Long Short-Term Memory (LSTM) networks, augmented with an attention mechanism. Our system works on the token-level, leveraging word embeddings trained on a big collection of unlabeled Twitter messages. We ranked 2nd in 7 teams. A post-completion improvement of our model, achieves state-of-the-art results on #HashtagWars dataset.

1 Introduction

Computational humor (Stock and Strapparava, 2003) is an area in computational linguistics and natural language understanding. Most computational humor tasks focus on the problem of humor detection. However SemEval-2017 Task 6 (Potash et al., 2017) explores the subjective nature of humor, using a dataset of Twitter messages posted in the context of the TV show "@midnight". At each episode during the segment "Hashtag Wars", a topic in the form of a hashtag is given and viewers of the show post funny tweets including that hashtag. In the next episode, the show selects the ten funniest tweets and a final winning tweet.

In the past, computational humor tasks have been approached using hand-crafted features (Hempelmann, 2008; Mihalcea and Strapparava, 2006; Kiddon and Brun, 2011; Yang et al., 2015). However, these approaches require a laborious feature-engineering process, which usually leads to missing or redundant features, especially in the case of humor, which is hard to define and consequently hard to model. Recently, approaches using neural networks, that perform feature-learning, have shown great results (Chen and Lee, 2017; Potash et al., 2016; Bertero and Fung, 2016a,b) outperforming the traditional methods.

In this paper, we present a deep-learning system that we developed for subtask A - "Pairwise Comparison". The goal of the task is, given two tweets about the same topic, to identify which one is funnier. The labels are applied using the show's relative ranking. This is a very challenging task, because humor is subjective and the machine learning system must develop a sense of humor similar to that of the show, in order to perform well.

We employ a Siamese neural network, which generates a dense vector representation for each tweet and then uses those representations as features for classification. For modeling the Twitter messages we use Long Short-Term Memory (LSTM) networks augmented with a context-aware attention mechanism (Yang et al., 2016). Furthermore, we perform thorough text preprocessing that enables our neural network to learn better features. Finally, our approach does not rely on any hand-crafted features.

2 System Overview

2.1 External Data and Word Embeddings

We collected a big dataset of 330M English Twitter messages, which is used (1) for calculating word statistics needed for word segmentation and spell correction and (2) for training word embeddings. Word embeddings are dense vector representations of words (Collobert and Weston, 2008; Mikolov et al., 2013), capturing their semantic and syntactic information. We leverage our big Twitter dataset to train our own word embeddings, using GloVe (Pennington et al., 2014). The word embeddings are used for initializing the weights of the first layer (embedding layer) of our network.

Proceedings of the 11th International Workshop on Semantic Evaluations (SemEval-2017), pages 390–395,
Vancouver, Canada, August 3 - 4, 2017.

2.2 Text Preprocessing [1]

For preprocessing the text we perform the following steps: tokenization, spell correction, word normalization, word segmentation (for splitting hashtags) and word annotation (with special tags).

Tokenizer. Our tokenizer is able to identify most emoticons, emojis, expressions like dates (e.g. 07/11/2011, April 23rd), times (e.g. 4:30pm, 11:00 am), currencies (e.g. $10, 25mil, 50€), acronyms, censored words (e.g. s**t), words with emphasis (e.g. *very*) and more. This way we keep all these expressions as one token, so later we can normalize them, or annotate them (with special tags) reducing the vocabulary size and enabling our model to learn more abstract features.

Postprocessing. After the tokenization we add an extra postprocessing step, where we perform spell correction, word normalization, word segmentation (for splitting a hashtag to its constituent words) and word annotation. We use the Viterbi algorithm in order to perform spell correction (Jurafsky and Martin, 2000) and word segmentation (Segaran and Hammerbacher, 2009), utilizing word statistics (unigrams and bigrams) from our big Twitter dataset. Finally, we lowercase all words, and replace URLs, emails and user handles (@user), with special tags.

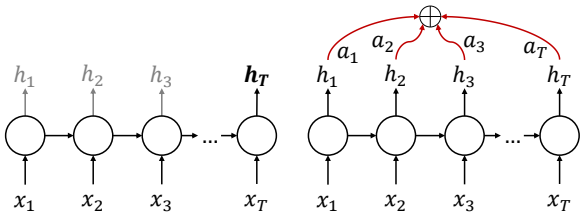
2.3 Recurrent Neural Networks

In computational humor tasks, the most popular approaches that utilize neural networks involve, Convolutional Neural Networks (CNN) (Chen and Lee, 2017; Potash et al., 2016; Bertero and Fung, 2016a) and Recurrent Neural Networks (RNN) (Bertero and Fung, 2016b). We model the text of the Twitter messages using RNNs, because CNNs have no notion of order, therefore losing the information of the word order. However, RNNs are designed for processing sequences, where the order of the elements matters. An RNN performs the same computation, $h_t = f_W(h_{t-1}, x_t)$, on every element of a sequence, where h_t is the hidden state at time-step t, and W the weights of the network. The hidden state at each time-step depends on the previous hidden states. As a result, RNNs utilize the information of word order and are able to handle inputs of variable length.

RNNs are difficult to train (Pascanu et al., 2013), because of the vanishing and exploding gradients problem, where gradients may grow or decay exponentially over long sequences (Bengio et al., 1994; Hochreiter et al., 2001). We overcome this limitation by using one of the more sophisticated variants of the regular RNN, the Long Short-Term Memory (LSTM) network (Hochreiter and Schmidhuber, 1997), which introduces a gating mechanism, that ensures proper gradient propagation through the network.

2.3.1 Attention Mechanism

An RNN can generate a fixed representation for inputs of variable length. It reads each element sequentially and updates its hidden state, which holds a summary of the processed information. The hidden state at the last time-step, is used as the representation of the input. In some cases, especially in long sequences, the RNN might not be able to hold all the important information in its final hidden state. In order to amplify the contribution of important elements (i.e. words) in the final representation, we use an attention mechanism (Rocktäschel et al., 2015), that aggregates all the intermediate hidden states using their relative importance (Fig. 1).



(a) Regular RNN (b) Attention RNN

Figure 1: Regular RNN and RNN with attention.

3 Model Description

In our approach, we adopt a Siamese architecture (Bromley et al., 1993), in which we create two identical sub-networks. Each sub-network reads a tweet and generates a fixed representation. Both subnetworks share the *same* weights, in order to project both tweets to the *same* vector space and thus be able to make a meaningful comparison between them. The Siamese sub-networks involve the Embedding layer, BiLSTM layer and Attention layer.

The network has two inputs, the sequence of words in the first tweet $X_1 = (x_1^1, x_2^1, ..., x_{T_1}^1)$, where T_1 the number of words in the first tweet, and the sequence words of the second tweet $X_2 = (x_1^2, x_2^2, ..., x_{T_2}^2)$, where T_2 the number of words of the second tweet.

[1] github.com/cbaziotis/ekphrasis

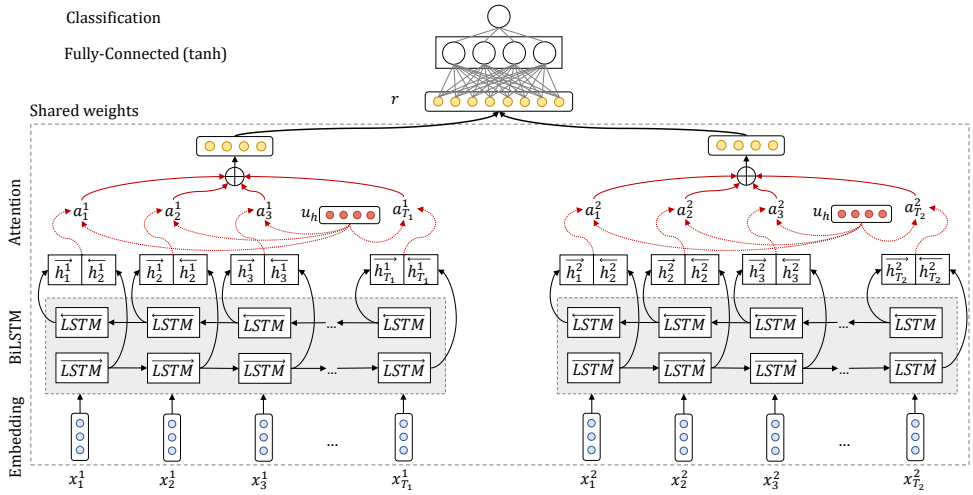


Figure 2: Siamese Bidirectional LSTM with context-aware attention mechanism.

Embedding Layer. We use an Embedding layer to project the words to a low-dimensional vector space R^E, where E is the size of the Embedding layer. We initialize the weights of the Embedding layer using our pre-trained word embeddings.

BiLSTM Layer. An LSTM takes as input the words of a tweet and produces the word annotations $H = (h_1, h_2, ..., h_T)$, where h_i is the hidden state of the LSTM at time-step i, summarizing all the information of the sentence up to x_i. We use bidirectional LSTM (BiLSTM) in order to get annotations for each word that summarize the information from both directions of the message. A bidirectional LSTM consists of a forward LSTM $\overrightarrow{f}$ that reads the sentence from x_1 to x_T and a backward LSTM $\overleftarrow{f}$ that reads the sentence from x_T to x_1. We obtain the final annotation for each word x_i, by concatenating the annotations from both directions,

$$h_i = \overrightarrow{h_i} \parallel \overleftarrow{h_i}, \quad h_i \in R^{2L} \tag{1}$$

where $\parallel$ denotes the concatenation operation and L the size of each LSTM.

Context-Attention Layer. An attention mechanism assigns a weight a_i to each word annotation, which reflects its importance. We compute the fixed representation r of the whole message as the weighted sum of all the word annotations using the attention weights. We use a context-aware attention mechanism as in (Yang et al., 2016). This attention mechanism introduces a context vector u_h, which can be interpreted as a fixed query, that helps to identify the informative words and it is randomly initialized and jointly learned with the rest of the attention layer weights. Formally,

$$e_i = tanh(W_h h_i + b_h), \quad e_i \in [-1, 1] \tag{2}$$

$$a_i = \frac{exp(e_i^\top u_h)}{\sum_{t=1}^{T} exp(e_t^\top u_h)}, \quad \sum_{i=1}^{T} a_i = 1 \tag{3}$$

$$r = \sum_{i=1}^{T} a_i h_i, \quad r \in R^{2L} \tag{4}$$

where W_h, b_h and u_h are the layer's weights.

Fully-Connected Layer. Each Siamese subnetwork produces a fixed representation for each tweet, r_1 and r_2 respectively, that we concatenate to produce the final representation r.

$$r = r_1 \parallel r_2, \quad r \in R^{4L} \tag{5}$$

We pass the vector r, to a fully-connected feed-forward layer with a $tanh$ (hyperbolic tangent) activation function. This layer learns a non-linear function of the input vector, enabling it to perform the complex task of humor comparison.

$$c = tanh(W_c r + b_c) \tag{6}$$

Output Layer. The output c of the comparison layer is fed to a final single neuron layer, that performs binary classification (logistic regression) and identifies which tweet is funnier.

3.1 Regularization

At first we adopt the simple but effective technique of dropout (Srivastava et al., 2014), in which we randomly turn-off a percentage of the neurons of a layer in our network. Dropout prevents co-adaptation of neurons and can also be thought as a form of ensemble learning, because for each training example a subpart of the whole

network is trained. Additionally, we apply dropout to the recurrent connections of the LSTM as suggested in (Gal and Ghahramani, 2016). Moreover, we add L_2 regularization penalty (weight decay) to the loss function to discourage large weights. Also, we stop the training of the network, after the validation loss stops decreasing (early-stopping). Lastly, we apply Gaussian noise and dropout at the embedding layer. As a result, the network never sees the exact same sentence during training, thus making it more robust to overfitting.

3.2 Training

We train our network to minimize the cross-entropy loss, using back-propagation with stochastic gradient descent and mini-batches of size 256, with the Adam optimizer (Kingma and Ba, 2014) and we clip the gradients at unit norm.

In order to find good hyper-parameter values in a relative short time, compared to grid or random search, we adopt the Bayesian optimization (Bergstra et al., 2013) approach. The size of the embedding layer is 300, the size of LSTM layers is 50 (100 for BiLSTM) and the size of the $tanh$ layer is 25. We insert Gaussian noise with $\sigma = 0.2$ and dropout of 0.3 at all layers. Moreover we apply dropout 0.2 at the recurrent connections of the LSTMs. Finally, we add L_2 regularization of 0.0001 at the loss function.

4 Results

Subtask A Results. The official evaluation metric of Subtask A is micro-averaged accuracy. Our team ranked 2nd in 7 teams, with score 0.632. A post-completion bug-fix improved significantly the performance of our model (Table 2).

	training	testing
hashtags	106	6
tweet pairs	109309	48285

Table 1: Dataset Statistics for Subtask A.

System	Acc Micro Avg
HumorHawk	0.675
DataStories (official)	0.632
Duluth	0.627
DataStories (fixed)	0.711

Table 2: The Results of our submitted and fixed models, evaluated on the official Semeval test set. The updated model would have ranked 1st.

#HastagWars Dataset Results. Furthermore, we compare the performance of our system on the #HastagWars dataset (Potash et al., 2016). Table 3 shows that our improved model outperforms the other approaches. The reported results are the average of 3 Leave-One-Out runs, in order to be comparable with (Potash et al., 2016). Figure 3 shows the detailed results of our model on the #HastagWars dataset, with the accuracy distribution over the hashtags.

System	Acc Micro Avg
LSTM (token) (Potash et al., 2016)	0.554 ($\pm$ 0.0085)
CNN (char) (Potash et al., 2016)	0.637 ($\pm$ 0.0074)
DataStories (fixed)	**0.696 ($\pm$ 0.0075)**

Table 3: Comparison on #HastagWars dataset.

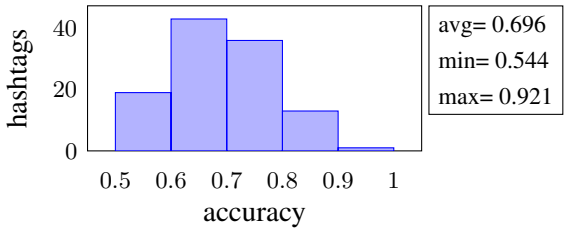


Figure 3: Detailed results on #HastagWars dataset.

Experimental Setup. For developing our models we used Keras (Chollet, 2015), Theano (Theano Dev Team, 2016) and Scikit-learn (Pedregosa et al., 2011). We trained our neural networks on a GTX750Ti(4GB), with each model taking approximately 30 minutes to train. Our source code is available to the research community[2].

5 Conclusion

In this paper we present our submission at SemEval-2017 Task 6 "#HashtagWars: Learning a Sense of Humor". We participated in Subtask A and ranked 2nd out of 7 teams. Our neural network uses a BiLSTM equipped with an attention mechanism in order to identify the most informative words. The network operates on the word level, leveraging word embeddings trained on a big collection of tweets. Despite the good results of our system, we believe that a character-level network will perform even better in computational humor tasks, as it will be able to capture the morphological characteristics of the words and possibly to identify word puns. We would like to explore this approach in the future.

[2] https://github.com/cbaziotis/datastories-semeval2017-task6

References

Yoshua Bengio, Patrice Y. Simard, and Paolo Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE Trans. Neural Networks* 5(2):157–166.

James Bergstra, Daniel Yamins, and David D. Cox. 2013. Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures. *Proceedings of ICML* 28:115–123.

Dario Bertero and Pascale Fung. 2016a. Deep learning of audio and language features for humor prediction. In *Proceedings of LREC*.

Dario Bertero and Pascale Fung. 2016b. A long short-term memory framework for predicting humor in dialogues. In *Proceedings of NAACL-HLT*. pages 130–135.

Jane Bromley, James W. Bentz, Léon Bottou, Isabelle Guyon, Yann LeCun, Cliff Moore, Eduard Säckinger, and Roopak Shah. 1993. Signature Verification Using A "Siamese" Time Delay Neural Network. *IJPRAI* 7(4):669–688.

Lei Chen and Chong Min Lee. 2017. Convolutional Neural Network for Humor Recognition. *arXiv preprint arXiv:1702.02584* .

François Chollet. 2015. Keras. https://github.com/fchollet/keras.

Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings ICML*. pages 160–167.

Yarin Gal and Zoubin Ghahramani. 2016. A theoretically grounded application of dropout in recurrent neural networks. In *Proceedings of NIPS*. pages 1019–1027.

Christian F. Hempelmann. 2008. Computational humor: Beyond the pun? *The Primer of Humor Research. Humor Research* 8:333–360.

Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, and Jürgen Schmidhuber. 2001. *Gradient Flow in Recurrent Nets: The Difficulty of Learning Long-Term Dependencies*. A field guide to dynamical recurrent neural networks. IEEE Press.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation* 9(8):1735–1780.

Daniel Jurafsky and James H. Martin. 2000. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall PTR, 1st edition.

Chloe Kiddon and Yuriy Brun. 2011. That's what she said: Double entendre identification. In *Proceedings of ACL*. pages 89–94.

Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* .

Rada Mihalcea and Carlo Strapparava. 2006. Learning to laugh (automatically): Computational models for humor recognition. *Computational Intelligence* 22(2):126–142.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Proceedings of NIPS*. pages 3111–3119.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the difficulty of training recurrent neural networks. In *Proceedings of ICML*. pages 1310–1318.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, and others. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* 12:2825–2830.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global Vectors for Word Representation. In *Proceedings of EMNLP*. volume 14, pages 1532–1543.

Peter Potash, Alexey Romanov, and Anna Rumshisky. 2016. # HashtagWars: Learning a Sense of Humor. *arXiv preprint arXiv:1612.03216* .

Peter Potash, Alexey Romanov, and Anna Rumshisky. 2017. SemEval-2017 Task 6: #HashtagWars: Learning a Sense of Humor. In *Proceedings of SemEval*.

Tim Rocktäschel, Edward Grefenstette, Karl Moritz Hermann, Tomáš Kočiskáżş, and Phil Blunsom. 2015. Reasoning about entailment with neural attention. *arXiv preprint arXiv:1509.06664* .

Toby Segaran and Jeff Hammerbacher. 2009. *Beautiful Data: The Stories Behind Elegant Data Solutions*. "O'Reilly Media, Inc.".

Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research* 15(1):1929–1958.

Oliviero Stock and Carlo Strapparava. 2003. Getting serious about the development of computational humor. In *Proceedings of IJCAI*. pages 59–64.

Theano Dev Team. 2016. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints* abs/1605.02688.

Diyi Yang, Alon Lavie, Chris Dyer, and Eduard H. Hovy. 2015. Humor Recognition and Humor Anchor Extraction. In *Proceedings of EMNLP*. pages 2367–2376.

Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. 2016. Hierarchical attention networks for document classification. In *Proceedings of NAACL-HLT*. pages 1480–1489.