We Used Neural Networks to Detect Clickbaits: You Won't Believe What Happened Next!

Ankesh Anand[1(✉)], Tanmoy Chakraborty[2], and Noseong Park[3]

[1] Indian Institute of Technology, Kharagpur, India
ankeshanand@iitkgp.ac.in
[2] University of Maryland, College Park, USA
tanchak@umiacs.umd.edu
[3] University of North Carolina, Charlotte, USA
npark2@uncc.edu

Abstract. Online content publishers often use catchy headlines for their articles in order to attract users to their websites. These headlines, popularly known as *clickbaits*, exploit a user's curiosity gap and lure them to click on links that often disappoint them. Existing methods for automatically detecting clickbaits rely on heavy feature engineering and domain knowledge. Here, we introduce a neural network architecture based on *Recurrent Neural Networks* for detecting clickbaits. Our model relies on distributed word representations learned from a large unannotated corpora, and character embeddings learned via Convolutional Neural Networks. Experimental results on a dataset of news headlines show that our model outperforms existing techniques for clickbait detection with an accuracy of 0.98 with F1-score of 0.98 and ROC-AUC of 0.99.

Keywords: Clickbait detection · Deep learning · Neural networks

1 Introduction

"Clickbait" is a term used to describe a news headline which will tempt a user to follow by using provocative and catchy content. They purposely withhold the information required to understand what the content of the article is, and often exaggerate the article to create misleading expectations for the reader. Some of the example of clickbaits are:

– "The Hot New Phone Everybody Is Talking About"
– "You'll Never Believe Who Tripped and Fell on the Red Carpet"

Clickbaits work by exploiting the insatiable appetite of humans to indulge their curiosity. According to the Loewenstein's information gap theory of curiosity [1], people feel a gap between what they know and what they want to know, and curiosity proceeds in two basic steps – first, a situation reveals a painful gap in our knowledge (that's the headline), and then we feel an urge to fill this gap and ease that pain (that's the click). Clickbaits clog up the social media news

J.M. Jose et al. (Eds.): ECIR 2017, LNCS 10193, pp. 541–547, 2017.
DOI: 10.1007/978-3-319-56608-5_46

streams with low-quality content and violate general codes of ethics of journalism. Despite a huge amount of backlash and being a threat to journalism [2], their use has been rampant and thus it's important to develop techniques that automatically detect and combat clickbaits.

There is hardly any existing work on clickbait detection except Potthast et al. [3] (specific to the Twitter domain) and Chakraborty et al. [4]. The existing methods rely on a rich set of hand-crafted features by utilizing existing NLP toolkits and language specific lexicons. Consequently, it is often challenging to adapt them to multi-lingual or non-English settings since they require extensive linguistic knowledge for feature engineering and mature NLP toolkits/lexicons for extracting the features without severe error propagation. Extensive feature engineering is also time consuming and sometimes corpus dependent (for example features related to tweet meta-data are applicable only to Twitter corpora).

In contrast, recent research has shown that deep learning methods can minimize the reliance on feature engineering by automatically extracting meaningful features from raw text [5]. Thus, we propose to use distributed word embeddings (in order to capture lexical and semantic features) and character embeddings (in order to capture orthographic and morphological features) as features to our neural network models.

In order to capture contextual information outside individual or fixed sized window of words, we explore several Recurrent neural network (RNN) architectures such as Long Short Term Memory (LSTM), Gated Recurrent Units (GRU) and standard RNNs. Recurrent Neural Network models have been widely adopted for their ability to model sequential data such as speech and text well.

Finally, to evaluate the efficacy of our model, we conduct experiments on a dataset consisting of clickbait and non-clickbait headlines. We find that our proposed model achieves significant improvement over the state-of-the-art results in terms of accuracy, F1-score and ROC-AUC score. We plan to open-source the code used to build our model to enable reproducibility and also release the training weights of our model so that other developers can build tools on top of them.

2 Model

The network architecture of our model as illustrated in Fig. 1 has the following structure:

- **Embedding Layer:** This layer transforms each word into embedded features. The embedded features are a concatenation of the word's Distributed word embeddings and Character level word embeddings. The embedding layer acts as input to the hidden layer.
- **Hidden Layer:** The hidden layer consists of a Bi-Directional RNN. We study different types of RNN architectures (described briefly in Sect. 2.2). The output of the RNN is a fixed sized representation of its input.
- **Output Layer:** In the output layer, the representation learned from the RNN is passed through a fully connected neural network with a sigmoid output node that classifies the sentence as clickbait or non-clickbait.

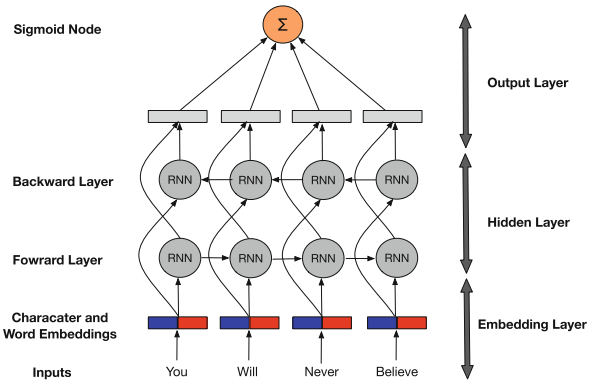

Fig. 1. BiDirectional RNN architecture for detecting clickbaits

2.1 Features

Two types of features are used in this experiment.

Distributed Word Embeddings: Distributed word embeddings map words in a language to high dimensional real-valued vectors in order to capture hidden semantic and syntactic properties of words. These embeddings are typically learned from large unlabeled text corpora. In our work, we use the pre-trained 300 dimensional word2vec embeddings which were trained on about 100B words from the Google News dataset using the Continuous Bag of Words architecture [6].

Character Level Word Embeddings: Character level word embeddings [7] have been used in several NLP tasks recently in order to incorporate character level inputs to build word embeddings. Apart from being able to capture orthographic and morphological features of a word, they also mitigate the problem of out-of-vocabulary-words as we can embed any word by its characters through character level embedding. In our work, we first initialize a vector for every character in the corpus. Then we learn the vector representation for any word by applying 3 layers of 1-dimensional CNN [8] with Rectified Linear Unites (ReLU) non-linearity on each vector of character sequence of that word and finally max-pooling across the sequence for each convolutional feature.

2.2 Recurrent Neural Network Models

Recurrent Neural Network (RNN) is a class of artificial neural networks which utilizes sequential information and maintains history through its intermediate layers. A standard RNN has an internal state whose output at each time-step is dependent on that of the previous time-steps. Expressed formally, given an input sequence x_t, a RNN computes it's internal state h_t by:

$$h_t = g(Uh_{t-1} + W_x x_t + b)$$

where g is a non-linear function such as $tanh$. U and W_x are model parameters and b is the bias vector.

Long Short Term Memory (LSTM): Standard RNNs have difficulty preserving long range dependencies due to the vanishing gradient problem [9]. In our case, this corresponds to interaction between words that are several steps apart. The LSTM is able to alleviate this problem through the use of a gating mechanism. Each LSTM cell computes its internal state through the following iterative process:

$$
\begin{aligned}
i_t &= \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \\
f_t &= \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f) \\
c_t &= f_t \odot c_{t-1} + i_t \odot tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \\
o_t &= \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o) \\
h_t &= o_t \odot tanh(c_t)
\end{aligned}
$$

where σ is the sigmoid function, and i_t, f_t, o_t and c_t are the input gate, forget gate, output gate, and memory cell activation vector at time step t respectively. $\odot$ denotes the element-wise vector product. W matrices with different subscripts are parameter matrices and b is the bias vector.

Gated Recurrent Unit (GRU): A gated recurrent unit (GRU) was proposed by Cho et al. [10] to make each recurrent unit adaptively capture dependencies of different time scales. Similarly to the LSTM unit, the GRU has gating units that modulate the flow of information inside the unit, however, without having a separate memory cells. A GRU cell computes it's internal state through the following iterative process:

$$
\begin{aligned}
z_t &= \sigma(W_z x_t + U_z h_{t-1}) \\
r_t &= \sigma(W_r x_t + U_r h_{t-1}) \\
\tilde{h}_t &= tanh(W_h x_t + U(r_t \odot h_{t-1})) \\
h_t &= (1 - z_t)\tilde{h}_{t-1} + z_t h_t
\end{aligned}
$$

where z_t, r_t, $\tilde{h}_t$ and h_t are respectively, the update gate, reset gate, candidate activation, and memory cell activation vector at time step t. W_h, W_r, W_z, U_r and U_z are parameters of the GRU and $\odot$ denotes the element-wise vector product.

In our experiments, we use the Bi-directional variants of these architectures since they are able to capture contextual information in both forward and backward directions.

3 Evaluation

Dataset: We evaluate our method on a dataset of 15,000 news headlines released by Chakraborty et al. [4] which has an even distribution of 7,500 clickbait headlines and 7,500 non-clickbait headlines. The non-clickbait headlines in the dataset

were sourced from Wikinews, and clickbait headlines were sourced from BuzzFeed, Upworthy, ViralNova, Scoopwhoop and ViralStories. We perform all our experiments using 10-fold cross validation on this dataset to maintain consistency with the baseline methods.

Training Setup: For training our model, we use the mini-batch gradient descent technique with a batch size of 64, the ADAM optimizer for parameter updates and Binary Cross Entropy Loss as our loss function. To prevent overfitting, we use the dropout technique [11] with a rate of 0.3 for regularization. During training, the character embeddings are updated to learn effective representations for this specific task. Our implementation is based on the Keras [12] library using a TensorFlow backend.

Comparison of Different Architectures: We first evaluate the performance of different RNN architectures using Character Embeddings (CE), Word Embeddings (WE) and a combination of both (CE+WE). Table 1 shows the result obtained by various RNN models on different metrics (specifically Accuracy, Precision, Recall, F1, and ROC-AUC scores) after 10-fold cross validation.

Table 1. Performance of various RNN architectures after 10-fold cross validation. The 'Bi' prefix means that the architecture is Bi-directional.

Model	Accuracy	Precision	Recall	F1-Score	ROC-AUC
BiRNN (CE)	0.9629	0.9513	0.9757	0.9633	0.9929
BiRNN (WE)	0.9650	0.9722	0.9573	0.9647	0.9935
BiRNN (CE+WE)	0.9666	0.9530	0.9787	0.9655	0.9938
BiGRU (CE)	0.9661	0.9833	0.9482	0.9634	0.9945
BiGRU (WE)	0.9769	0.9761	0.9778	0.9770	0.9965
BiGRU (CE+WE)	0.9774	0.9662	**0.9893**	0.9776	0.9979
BiLSTM (CE)	0.9673	**0.9849**	0.9492	0.9667	0.9950
BiLSTM (WE)	0.9787	0.9759	0.9815	0.9787	0.9970
BiLSTM (CE+WE)	**0.9819**	0.9839	0.9799	**0.9819**	**0.9980**

We observe that BiLSTM(CE+WE) model slightly outperforms other models, and the BiLSTM architecture in general performs better than BiGRU and BiRNN. If we look at performance of an individual architecture using three different set of features, model using a combination of word embeddings and character embeddings consistently gives the best results, closely followed by model with only word embeddings.

Comparison with Existing Baselines: Finally, we compare our model with state-of-the-art results on this dataset as reported in Chakraborty et al. [4]. The models reported in [4] use a combination of structural, lexical and lexicon based features. In Table 2, we notice that our BiLSTM(CE+WE) model shows

Table 2. Comparison of our model with the baseline methods.

Model	Accuracy	Precision	Recall	F1-score	ROC-AUC
Chakraborty et al. (2016) (SVM)	0.93	0.95	0.90	0.93	0.97
Chakraborty et al. (2016) (Decision Tree)	0.90	0.91	0.89	0.90	0.90
Chakraborty et al. (2016) (Random Forest)	0.92	0.94	0.91	0.92	0.97
BiLSTM(CE+WE)	**0.98**	**0.98**	**0.98**	**0.98**	**0.99**

more than 5% improvement in terms of both accuracy and F1-score and more than 2% in terms of the ROC-AUC score over the best performing baseline (i.e. Chakraborty et al. [4] (SVM)).

4 Conclusion

In this paper, we introduced three different variants of Bidirectional Recurrent Neural Network model for detecting clickbaits using distributed word embeddings and character-level word embeddings. We showed that these models achieve significant improvement over the state-of-the-art in detecting clickbaits without relying on heavy feature engineering. In future, we would like to qualitatively visualize the internal states of our model and incorporate attention mechanism into our model.

References

1. Loewenstein, G.: The psychology of curiosity: a review and reinterpretation. Psychol. Bull. **116**(1), 75 (1994)
2. Dvorkin, J.: Why click-bait will be the death of journalism (2016). http://to.pbs.org/2gQ6mCN
3. Potthast, M., Köpsel, S., Stein, B., Hagen, M.: Clickbait detection. In: Ferro, N., Crestani, F., Moens, M.-F., Mothe, J., Silvestri, F., Nunzio, G.M., Hauff, C., Silvello, G. (eds.) ECIR 2016. LNCS, vol. 9626, pp. 810–817. Springer, Cham (2016). doi:10.1007/978-3-319-30671-1_72
4. Chakraborty, A., Paranjape, B., Kakarla, S., Ganguly, N.: Stop clickbait: Detecting and preventing clickbaits in online news media. In: ASONAM, pp. 9–16. San Fransisco, USA (2016)
5. Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., Kuksa, P.: Natural language processing (almost) from scratch. J. Mach. Learn. Res. **12**, 2493–2537 (2011)
6. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. arxiv preprint (2013). arXiv:1301.3781
7. dos Santos, C.N., Zadrozny, B.: Learning character-level representations for part-of-speech tagging. In: ICML, pp. 1818–1826 (2014)
8. Le Cun, B.B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W., Jackel, L.D.: Handwritten digit recognition with a back-propagation network. In: Advances in neural information processing systems. Citeseer (1990)

9. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
10. Cho, K., Van Merriënboer, B., Bahdanau, D., Bengio, Y.: On the properties of neural machine translation: encoder-decoder approaches. arxiv preprint (2014). arXiv:1409.1259
11. Srivastava, N., Hinton, G.E., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. **15**(1), 1929–1958 (2014)
12. Chollet, F.: Keras (2015). https://github.com/fchollet/keras