Similarity-Aware Deep Attentive Model for Clickbait Detection

Team Name: Ego Et Al

Tanmay Bhatt - 2020112017
Pranav Manu - 2020112019
Anjali Singh - 2020102004
Abhinav Siddharth - 2020112007

Introduction

Clickbaits are a type of web links designed to entice users to enter specific web-pages or videos. We come across clickbait text and images multiple times each day. Usually, such links will lead to non-informative and misleading articles which causes the reader to stray from the topic which they were actually looking for. Hence, it becomes imperative to come up with methods for automated clickbait detection such that users can be warned against potential clickbait material.

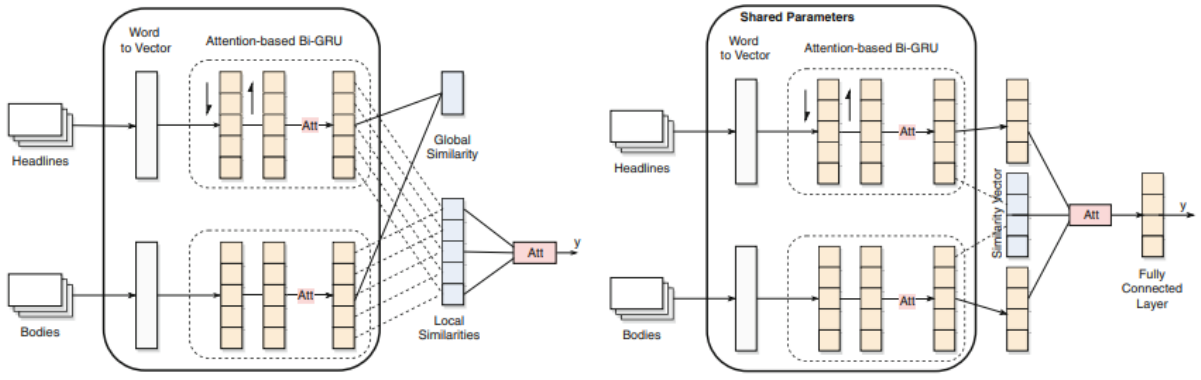
Clickbait detection has been a fairly sought after topic in recent years and hence considerable amount of work has been put forward toward the same. The first few approaches extracted linguistic features from both the header and the body text and passed these features on traditional classifiers like logistic regression, bayes classifier, random forest, etc. Existing studies have several limitations (i) they consider the similarity as features in a linear manner and therefore, lack the expressiveness when compared to non-linear methods; (ii) current efforts on leveraging such similarities typically use the partial/local information, such as quantifying the similarity between the titles and the top five sentences of content; on the other hand, they overlooking the hidden global information in the entire content.

The objective of our project is to build a deep similarity-aware attentive model for capturing the discriminative information from local and global similarities. In other words, given a headline and an article body, the model should be able to classify the text as clickbait or not. Clickbaits can be understood using a few examples such as, “You’ll never believe what happened after..”, “How to achieve some unbelievable results with this one weird trick!”, “This is the biggest mistake you can make…”, etc.

We use a combination of bi-GRU and attention network to learn headline and body representations and compute a similarity vector based on the global and local similarities between the headline and body vectors.

Model

We propose a model that will exploit both the local and global similarities between the header text and the body text. Moreover, the similarities are modeled as vectors so that they can be augmented with other features for future prediction easily. Given a set of titles, H = {h_1, h_2, ..., h_N} and their bodies, B = {b_1, b_2, ..., b_N} and the labels, Y = {y_1, y_2, ..., y_N}; y_i = 1 if the headline is a clickbait. The framework in the paper includes (i) Learning latent representations, (ii) Learning the similarities, (iii) Using the similarities for further predictions.



Understanding the GRU Model

GRU is a simplified version of Long Short-Term Memory (LSTM), which is an improved version of recurrent neural networks (RNN's). RNN's are unable to learn long term dependencies as effectively.

Recurrent neural networks (RNN) consider past information or memory for classification, but RNNs face an issue called vanishing gradient problem and are unable to learn long-term dependencies. GRU is a simplified version of Long-Short term memory as it has a fewer number of gates (gates are neural network layers that determine which information should be kept or forgotten). GRU also has lesser matrix and tensor operation compared to LSTM making its training faster.

GRU works using 2 gates: update gate and reset gate. The update gate is responsible for determining what information about a new entry is retained or forgotten. In contrast, the reset gate determines how much the past data will be forgotten.

Learning the Latent Representation

Firstly, pre-processing of text information is performed which involves the removal of all punctuations, stop words, converting to lower form, and word lemmatization, for which we used the 'nltk' module (Natural Language Toolkit).

The next step is to transform the clean input to word vectors, for which we used the "gensim" module. After performing pre-processing, the next step involves applying bi-directional GRU (Gated Recurrent Unit) which is a simplified version of Long Short-Term Memory (LSTM) which is an improved version of RNNs. RNNs are unable to learn long term dependencies as effectively as GRU can. GRU is used to get annotations of words by summarizing information from both the directions of

a word. We get hidden representations by concatenating forward and backward hidden states.

Given a b_i, we first get a set of word embedding vectors, $w_{i,t}$, $t \in [1, N]$, T_i is the number of words in body i. The forward GRU reads the sentence from w_{i1} to w_{iT_i} and a backward GRU which reads the sentence from w_{iT_i} to w_{i1}. We use a bidirectional GRU to get information from the words by summarizing information from both directions in the sentence.

$$\overrightarrow{w'_{it}} = \overrightarrow{GRU}(w_{it}), t \in [1, T_i]$$
$$\overleftarrow{w'_{it}} = \overleftarrow{GRU}(w_{it}), t \in [T_i, 1]$$

Then, we get the hidden representation w'_{it} by concatenating the forward and backward hidden states. Latent representation is :

$$w'_{it} = [\overrightarrow{w'_{it}}, \overleftarrow{w'_{it}}]$$

The hidden representation w'_{it}, summarizes the information of the whole sentence centered around w_{it}.

All words do not contribute equally to the sentence meaning, thus an attention mechanism is used to extract words which are important to the meaning of the sentence.

$$u_t = tanh(W_w w'_{it} + b_w)$$
$$a_t = \frac{exp(u_t^T u_w)}{\Sigma_t exp(u_t^T u_w)}$$
$$L_{b_i} = \Sigma_t a_t w'_{it}$$

We first pass the hidden representation through a one layer MLP to get an importance measure u_t, then we 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 a_t through a softmax function. After that we get a sentence vector/ latent representation of all the words in the sentence as a weighted combination of the word annotations.

This is how we get the latent representation of the bodies L_B. Attention mechanism is used to extract important words and further on, the representation of the words are aggregated to get the latent representations.

Learning the Similarities

Next we find global similarities as cosine between L_H and L_B. The similarity $r(H, B)$ is a constant within [0, 1], a higher value of which stands for a higher level of consistency between the titles and bodies. Higher the value, higher would be the consistency.

$$r(H, B) = cosine(L_H, L_B) = \frac{L_H^T L_B}{\| L_H \| \| L_B \|}$$

For using only global similarity to predict the clickbait, we use: -

$$R(H, B) = softmax[r(H, B), (1 - r(H, B))]$$

We use cross entropy for measuring the loss:

$$\mathcal{L} = -\Sigma_{Y=0,1} Y \log P(Y|H, B)$$

Then, the optimization goal is to minimize this loss. We use Adam as our optimization method.

$$\underset{\Theta}{\operatorname{argmin}} \mathcal{L} + \lambda \| \Theta \|_2$$

Local similarities are found by taking similarities of all blocks in a vector format. Attentive mechanism is used to select the most useful local similarities for final predictions.

$$LS(H, B) = (r(L_{H,1}, L_{B,1}), ..., r(L_{H,K}, L_{B,K}))^T$$

Then, we apply the self-attention mechanism for getting the attention values (which serve as self-learned weight values).

$$A = softmax(V_a \tanh(W_a LS(H, B)^T)) \quad W_a \in \mathbb{R}^K \; V_a \in \mathbb{R}^{K \times K}$$

A is the attention matrix and W_a and V_a are two weight matrices. Then, prediction for the clickbait is calculated as $\hat{y} = argmax_y(P)$.

$$P = softmax(W_P(A \times LS(H, B)) + b_P)$$

Learning for Prediction

Once we have the similarities, we can combine them with standard clickbait indicators like writing

style and text quality. In order to combine these, we make use of the latent representations.

$$L'_H = f(W_H L_H + b_H)$$
$$L'_B = f(W_B L_B + b_B)$$

We then calculate self attention values which are representative of the writing style and text quality and combine it with the latent representations to get a final combination layer on which a multilayer perceptron will be applied to get the prediction.

Layer Visualization

```
==========================================================================================================================================
Layer (type:depth-idx)                   Input Shape               Output Shape              Param #                   Mult-Adds
==========================================================================================================================================
LSD                                      [20, 96, 100]             [20, 200]                 --                        --
├─SimilarityAware: 1-1                   [20, 96, 100]             [20, 200]                 --                        --
│    └─GRU: 2-1                          [20, 96, 100]             [20, 96, 200]             121,200                   232,704,000
│    └─Linear: 2-2                       [20, 96, 200]             [20, 96, 1]               201                       4,020
│    └─Tanh: 2-3                         [20, 96, 1]               [20, 96, 1]               --                        --
│    └─Softmax: 2-4                      [20, 96, 1]               [20, 96, 1]               --                        --
├─SimilarityAware: 1-2                   [20, 96, 100]             [20, 200]                 --                        --
│    └─GRU: 2-5                          [20, 96, 100]             [20, 96, 200]             121,200                   232,704,000
│    └─Linear: 2-6                       [20, 96, 200]             [20, 96, 1]               201                       4,020
│    └─Tanh: 2-7                         [20, 96, 1]               [20, 96, 1]               --                        --
│    └─Softmax: 2-8                      [20, 96, 1]               [20, 96, 1]               --                        --
==========================================================================================================================================
Total params: 242,802
Trainable params: 242,802
Non-trainable params: 0
Total mult-adds (M): 465.42
==========================================================================================================================================
Input size (MB): 1.54
Forward/backward pass size (MB): 6.17
Params size (MB): 0.97
Estimated Total Size (MB): 8.68
==========================================================================================================================================
```

```
==========================================================================================================================================
Layer (type:depth-idx)                   Input Shape               Output Shape              Param #                   Mult-Adds
==========================================================================================================================================
LSDA                                     [20, 96, 100]             [20, 2]                   5,025                     --
├─LSD: 1-1                               [20, 96, 100]             [20, 200]                 --                        --
│    └─SimilarityAware: 2-1              [20, 96, 100]             [20, 200]                 --                        --
│    │    └─GRU: 3-1                     [20, 96, 100]             [20, 96, 200]             121,200                   232,704,000
│    │    └─Linear: 3-2                  [20, 96, 200]             [20, 96, 1]               201                       4,020
│    │    └─Tanh: 3-3                    [20, 96, 1]               [20, 96, 1]               --                        --
│    │    └─Softmax: 3-4                 [20, 96, 1]               [20, 96, 1]               --                        --
│    └─SimilarityAware: 2-2              [20, 96, 100]             [20, 200]                 --                        --
│    │    └─GRU: 3-5                     [20, 96, 100]             [20, 96, 200]             121,200                   232,704,000
│    │    └─Linear: 3-6                  [20, 96, 200]             [20, 96, 1]               201                       4,020
│    │    └─Tanh: 3-7                    [20, 96, 1]               [20, 96, 1]               --                        --
│    │    └─Softmax: 3-8                 [20, 96, 1]               [20, 96, 1]               --                        --
├─myLinear: 1-2                          [20, 1, 25]               [20, 25, 25]              25                        500
├─Tanh: 1-3                              [20, 25, 25]              [20, 25, 25]              --                        --
├─myLinear: 1-4                          [20, 25, 25]              [20, 25, 25]              625                       12,500
├─Softmax: 1-5                           [20, 625]                 [20, 625]                 --                        --
├─Linear: 1-6                            [20, 25]                  [20, 2]                   52                        1,040
├─Softmax: 1-7                           [20, 2]                   [20, 2]                   --                        --
==========================================================================================================================================
Total params: 248,529
Trainable params: 248,529
Non-trainable params: 0
Total mult-adds (M): 465.43
==========================================================================================================================================
Input size (MB): 1.54
Forward/backward pass size (MB): 6.38
Params size (MB): 0.97
Estimated Total Size (MB): 8.89
==========================================================================================================================================
```

Comparison with traditional Machine learning models

We have applied the GRU model for classifying the text as clickbait or not when given the text headers and body. We shall now compare the proposed method with the following classification approaches: Support Vector Machine (SVM), Logistic Regression (LR) and Random Forest Classifier (RFC).

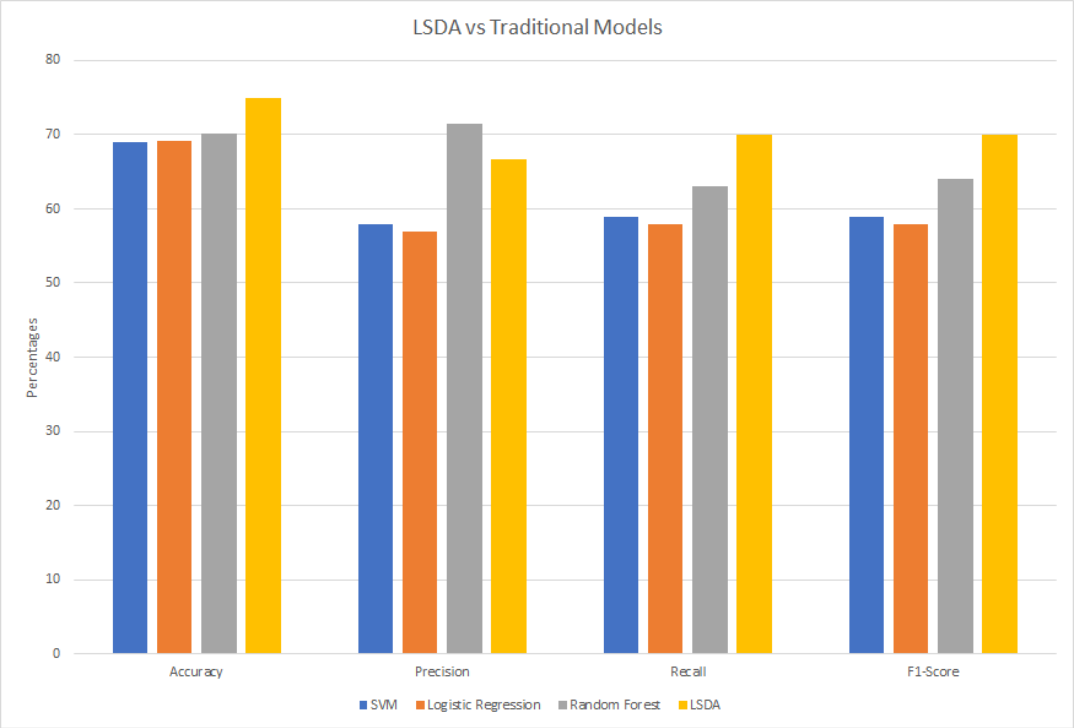
Methodology:-

1. The header and body tensors of the training and testing dataset were first flattened such that for each sample the features are present in the same dimension, where the total number of features in each header or body sample is equal to the number of vectors in the vector representation of all words times the word2vec length. Each sample is represented by: (H,B,y).
2. The header and the body vectors are both concatenated together to form a new merged vector of the form (X, y).
3. Now the SVM, LR and RFC models are trained on the training data and later evaluated on the test set and the accuracy of the models is recorded.

Trained and Tested on Clickbait Dataset - 2017. These are all weighted-averages.

Methods	Accuracy	Precision	Recall	F1-Score
SVM	61.78%	58%	59%	59%
Logistic Regression	64.22%	57%	58%	58%

Random Forests	72.35%	71.5%	63%	64%
LSDA	75.02%	66.66%	69.99%	69.98%



Datasets

The datasets considered here are the same as the ones mentioned in the paper, namely

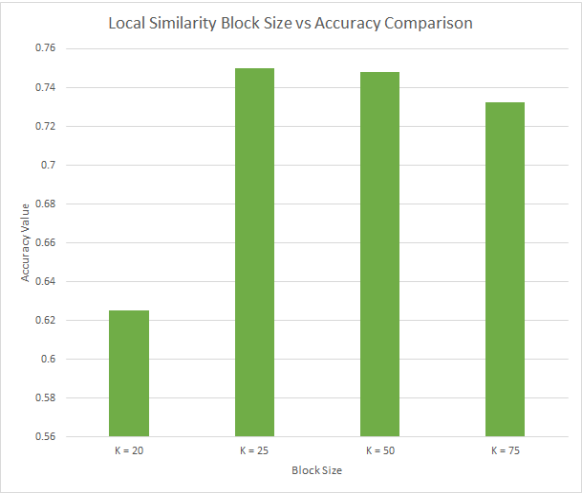
1. **Clickbait Challenge 2017** - This set contains around 20,000 labeled pairs of posts with each post being judged by 5 judges, each of whom gives a score between 0 and 1. Higher the score, higher the chance of the post being clickbait. We take the mean of all 5 verdicts and quantize it with 0.5 as the threshold. 1 means clickbait and 0 means not-clickbait.

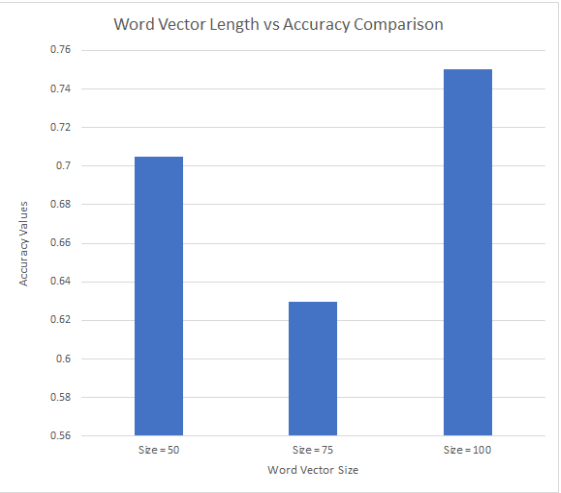
2. **FNC Dataset** - Fake News Challenge in 2017, where data describes pairs of titles and bodies and are labeled as 'agree', 'disagree', 'discuss' and 'unrelated'. The data labeled as 'unrelated' is considered as a clickbait. The dataset contains around 50,000 pairs of titles and bodies for training and around 25,000 pairs for the testing.

We have trained and tested our model on the clickbait challenge 2017 dataset and could not run the model on the FNC dataset due to hardware constraints. (All our machines, as well as Google Colab ran out of RAM)

Outputs

Figure 1 shows the outputs for the accuracy of our model when compared to other traditional models like SVM, Logistic Regression and Random Forests. Figure 2 represents a comparison of accuracy with different word to vector lengths.





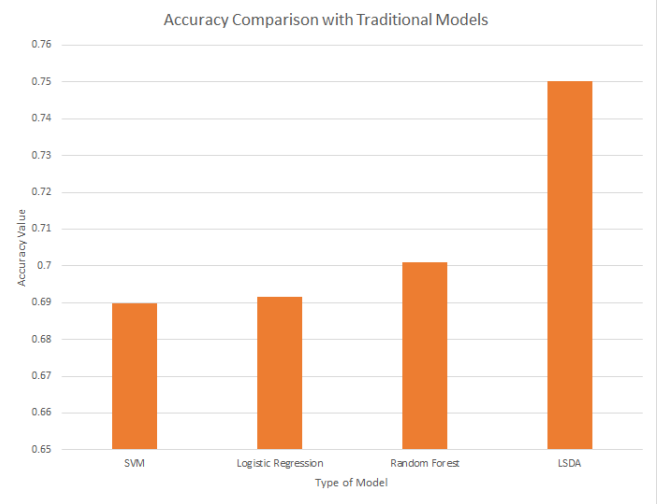
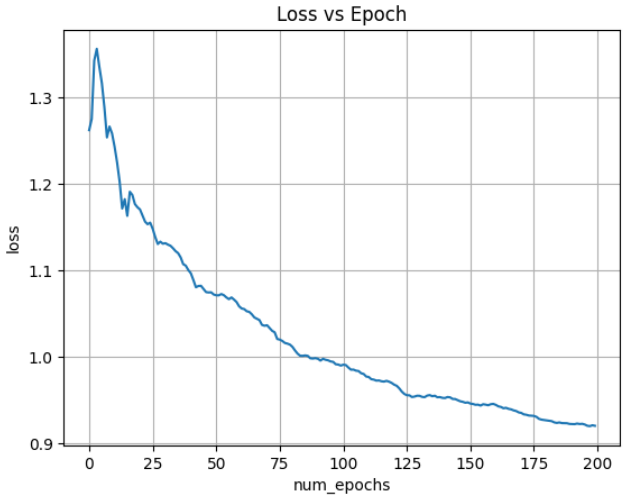


Figure 3 shows comparison in accuracy with different traditional models like SVM, Logistic Regression and Random Forests.



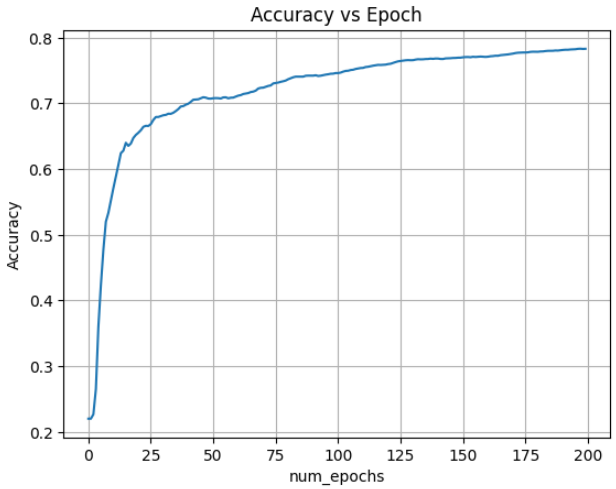
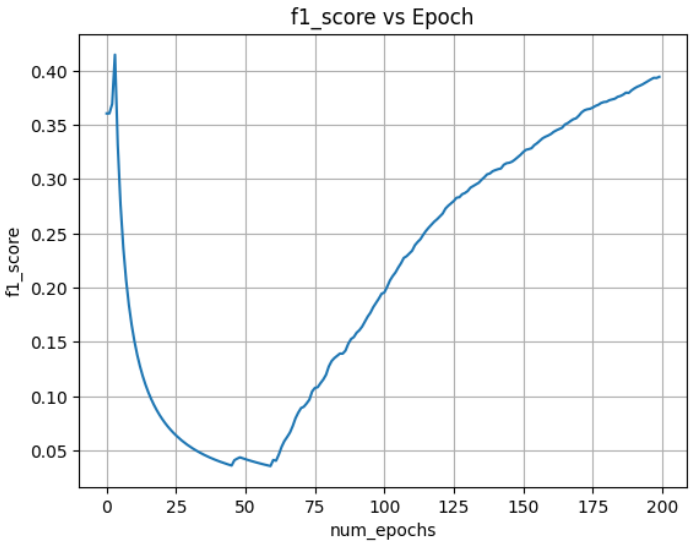


Figure 4, 5 and 6 represent the plots of loss, accuracy and F1 score, respectively, with different values of epochs.



Shortcomings/Limitations of the Model

The model we are considering here has some limitations which are: -

- The model doesn't consider patterns or styles of the titles. For example, the sentence, "You WON'T believe what happened next?", would be converted to lowercase and an important cue for clickbait would be lost.
- The model also ignores pictorial information while detecting clickbait text.
- The hyperparameter K, which is used to make blocks for finding local similarity needs to be tuned in order to get optimal performance.
- If the word to vector size is too small, the output of the deep learning model built overflows. This is called the Gradient Explosion Problem caused due to large updates to the neural network model while training.

Conclusion

In our project, we successfully implemented the similarity method to solve the problem of clickbait detection by exploiting global as well as local similarities, as opposed to the traditional feature engineering which lack the properties in representing the matching information between titles and targeted bodies and have gotten a rough accuracy value of 75.0255%. The accuracy on the training set we got is 81.99%. We have presented a local similarity-aware deep attentive model that learns both local similarities and raw input features to make predictions in an attentive manner.

Division of Work

- Pranav Manu: LSD and LSDA Model building, training and analysis
- Tanmay Bhatt: Preprocessing, word vectorization and model analysis
- Anjali Singh: Preprocessing, results compilation and model analysis
- Abhinav Siddharth: Model testing, tuning and model analysis

References

Chung, J., Gulcehre, C., Cho, K., Bengio, Y.: Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.35

He, T., Droppo, J.: Exploiting lstm structure in deep neural networks for speech recognition. In: 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). pp. 5445–5449. IEEE (2016)

Assis, Marcos VO, et al. "A GRU deep learning system against attacks in software defined networks." *Journal of Network and Computer Applications* 177 (2021): 102942.

Bahdanau, D., Cho, K., Bengio, Y.: Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473 (2014)

Biyani, P., Tsioutsiouliklis, K., Blackmer, J.: 8 amazing secrets for getting more clicks: detecting clickbaits in news streams using article informality. In: AAAI, pp. 94–100 (2016)

Yang, Zichao, et al. "Hierarchical attention networks for document classification." Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies. 2016.