The Report committee for Yan Jiang

Certifies that this is the approved version of the following report:

Using Machine Learning for Stance Detection

SUPERVISING COMMITTEE:

Matt Lease, Supervisor

Unmil P Karadkar

Using Machine Learning for Stance Detection

by

Yan Jiang

Report

Presented to the Faculty of the Graduate School

of the University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree(s) of

Master of Science in information studies

The University of Texas at Austin

December 2018

Abstract

Using Machine Learning for Stance Detection

by

Yan Jiang, M.S.INFO.ST

The University of Texas at Austin, 2018

Supervisor: Matt Lease

In this report, we focus on the task of stance detection with the power of machine learning techniques to determine the relative stance between a headline and an associated article. We explore the performance of various models on the fake news challenge dataset, including both traditional methods and neural networks. We find that the Enhanced Sequential Inference Model proposed for Natural Language Inference achieves great performance on the task of stance detection. To address the imbalanced class distribution problem, we use Focal Loss to train our neural networks.

Table of Contents

1. Introduction

In this report, we explore how machine learning and deep learning techniques could be used to detect fake news. Wikipedia defines fake news as a type of propaganda that consists of deliberate disinformation or hoaxes spread through traditional print, news media or online social media [1]. With the popularity of the Internet, fake news is easy to transmit and be conceived. Fake news may cause some problems; for example, people don't know whether to believe a piece of news or not.

To combat fake news with artificial intelligence methods, Pomerleau and Rao [2] organized the Fake news challenge Stage I (FNC-1): Stance Detection. The organizers of this competition believed that accessing the veracity of a news story is a complex and cumbersome task, even for trained experts, so a better way is to break it into steps or stages [2]. This challenge focuses on the first stage of the veracity checking, called the Stance Detection. More specifically, the stance could be one of the following values: Agree, Disagree, Discusses, and Unrelated. The first three could be merged into one class: Related. One of the examples is as follows:

Headline	Hundreds of Palestinians flee floods in Gaza as Israel opens dams
Article	Hundreds of Palestinians were evacuated from their homes Sunday morning after Israeli authorities opened a number of dams near the border, flooding the Gaza Valley in the wake of a recent severe winter storm. ...
Stance	Agree

Table 1: One example of (headline, article) pair in the FNC-1 dataset.

This report is structured as follows: We first review previous related work in chapter 2. Then in chapter 3, we cover some details about the FNC-1 dataset and discuss proper evaluation metrics for this dataset. Thereafter in chapter 4, we describe the machine learning concepts and models for the fake news challenge. In chapter 5, we provide the implementation details of the models we use and report the obtained results. In chapter 6, we design further experiments to explore our best model and analyze of the results. Lastly, in chapter 7, we summarize our work.

2. Related Work

2.1 Natural Language Inference

Natural Language Inference (NLI) is a task that focuses on inference about entailment and contradiction between a premise and a hypothesis. Bowman et al. [3] published a human annotated dataset called Stanford Natural Language Inference (SNLI) for learning natural language inference. The SNLI dataset is composed of 570,152 high-quality sentence pairs. With the availability of such a large-scale dataset, many complicated neural networks are proposed for the NLI task. These models can be divided into two categories: sentence-vector based models and compare-aggregate models. The former use a Siamese architecture [4]. For these models, firstly, we use two encoders for the premise and hypothesis. Then we concatenate the encodings. Finally, a Multilayer Perceptron classifier is used to decide the relationship between two documents. Various neural networks have been used for sentence encoder, such as Long-Short Term Memory [3], Gated Recurrent Unit [5], Convolutional Neural Network [6]. These models usually transform a sentence into a fixed-length vector.

The compare-aggregate models [7] usually utilize attention mechanisms to learn word alignments and then aggregate the information. Rocktaschel et al. [8] introduced neural attention-based models, which allow the model to pay more attention to detailed tokens of the sentence pair. To apply attention for the NLI task, Parikh et al. [9] proposed a relatively simple but effective model that only uses word embeddings and attention. Inspired by the idea of Parikh and other previous works, Chen et al. [10] created an enhanced sequential inference model that learns alignments between sentence pairs and aggregates them with another bidirectional long-short term memory layer. Gong et al. [11] came up with a deep interactive inference network that uses a convolutional network to extract information from the sentence pairs.

2.2 Document Representation

Transforming a piece of text into a fixed-length vector is one of the fundamental tasks in Natural Language Processing. Due to the simplicity and effectiveness, the bag-of-words, N-gram, and term frequency-inverse document frequency models are the most commonly used representations for documents. Documents can also be represented as a set of topics. Latent Semantic Indexing and Latent Dirichlet Allocation are two popular topic models. Apart from statistical based models, Quoc et al. [12] proposed a paragraph vector that learns a vector representation for a sentence or a paragraph. The idea is to predict the following words in a paragraph given a paragraph vector and several word vectors. Kiros et al. [13] introduced the skip-thought method which encodes one sentence to predict the surrounding sentences.

2.3 Text Classification

The goal of text classification is to label a document. Traditional representations of a document, like the bag-of-words or term frequency-inverse document frequency, are usually fed into a Support Vector Machine classifier with a linear kernel or a tree-based learner. Yang et al. [14] designed the Hierarchical Neural Network that utilizes attention mechanisms to find out the essential words and sentences in the documents.

3. Fake New Challenge

In this chapter, we describe the fake news challenge dataset and discuss appropriate metrics for this dataset.

3.1 FNC-1 dataset

The FNC-1 dataset consists of a training set with 49,972 (headline, article) pairs and a test set with 25,413 (headline, article) pairs. In the training set, there are 1,689 unique headlines and 1,648 unique articles. In the test set, there are 894 unique headlines and 904 unique articles. The class distribution in the training set and test set is reported in Table 2.

	Agree	Disagree	Discuss	Unrelated
Training set	7.4%	1.7%	17.8%	73.1%
Test set	7.5%	2.7%	17.6%	72.2%

Table 2: The class distribution in the training set and test set.

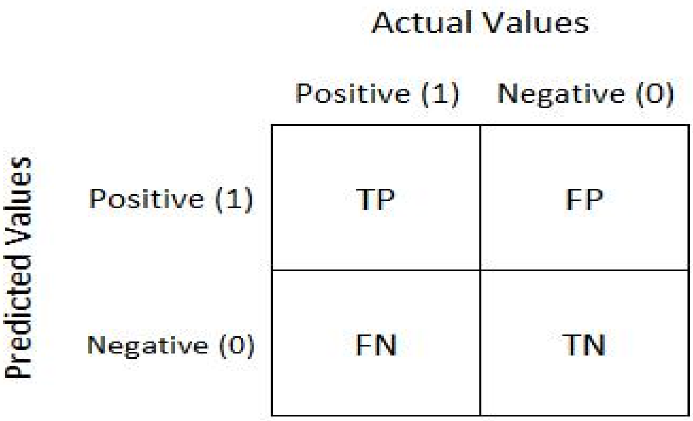
The FNC-1 dataset cannot be split into a training set and an evaluation set because this would cause overlap between the headlines and articles. Instead, the challenge organizers suggested a solution by randomly selecting articles from the dataset. The evaluation set is composed of the selected articles and associated headlines. This ensures that the model trained on the training set never “sees” the articles in the evaluation set. However, the model can “see” the headlines in the evaluation set. By adopting this method, we get a training set with 40,350 (headline, article) pairs and an evaluation set with 9,622 (headline, article) pairs.

Significantly, the FNC-1 dataset is highly imbalanced. Most of the (headline, article) pairs fall into the Unrelated class, while only 2 percent of the data is labeled as Disagree. More details can be found in Table 2. This imbalanced class distribution may pose limitations to the methods, especially evaluation metrics.

3.2 Evaluation Metrics

A confusion matrix is a table that visualizes the performance of a classifier. Each row of the matrix represents the instance in a predicted class while each column represents the instances in an actual class [15]. In the binary classification setting, the confusion matrix looks as follows:

3.2.1 Confusion Matrix



Actual Values

Predicted Values	Positive (1)	Negative (0)
Positive (1)	TP	FP
Negative (0)	FN	TN

Figure 1: A confusion matrix for binary classification

TP: labeled as positive and predicted as positive

FP: labeled as negative and predicted as positive

FN: labeled as positive and predicted as negative

TN: labeled as negative and predicted as negative

With a one-vs-rest strategy, it’s easy to extend the confusion matrix to a multiclass setting. For example, regarding all the samples with label Agree as positive samples and all other samples as negative [16].

3.2.2 Accuracy

With a confusion matrix available, accuracy is defined as follows in binary classification:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Accuracy measures the percentage of correct predictions. In multiclass classification, accuracy equals the total number of data samples divided by the number of samples correctly predicted by the algorithm. For the FNC-1 dataset, a system that simply returns Unrelated receives an accuracy score of 72.2% on the test set. A model that predicts Unrelated for all the Unrelated samples and Discuss for all the Related samples gets an accuracy score of 89.8% on the test set (this is an easy baseline). Because of this highly imbalanced property of the Fake News Challenge Dataset, accuracy is not an appropriate metric for this task.

3.2.3 FNC Score

Being aware of this imbalanced class distribution problem, the organizers of this challenge proposed a weight accuracy metric called FNC Score, which awards 0.25 points to each (headline, article) pair with correct predictions. Another 0.75 points are awarded if the label of a correct pair falls into one of the following categories: Agree, Disagree, Discuss. Finally, even if the system outputs a wrong label, it could still get 0.25 points as long as the prediction and the label are both in the Related class (Agree, Disagree, Discuss). FNC Score does take class distribution into consideration. However, since distinguishing between Related and Unrelated data pairs is fairly simple for a variety of classifiers such as support vector machines, even a trivial system that always outputs Discuss for the related data pairs can get a FNC score of 0.83, which beats the performance of winning teams [17]. Consequently, FNC Score is also not a good metric for this task.

3.2.4 Precision, Recall, and F Score

For a binary classification problem, Precision, Recall, F_1 and F score are defined as follows:

$$\Pr ecision = \frac{TP}{TP + FP}$$

$$\operatorname{Re} call = \frac{TP}{Tp + FN}$$

$$F_1 = 2 * \frac{\Pr ecison * \operatorname{Re} call}{\Pr ecision + \operatorname{Re} call}$$

$$F_\beta = \frac{\left(1 + \beta^2\right) \Pr ecision * \operatorname{Re} call}{\beta^2 * \Pr ecision + \operatorname{Re} call}$$

F_1 is a special case of F score where $\beta = 1$. Two other commonly used F scores are F_2 score, which favors Recall and $F_{0.5}$ which favors Precision. Since we care about both precision and recall, F_1 is chosen.

In the multiclass case, there are some variants of the F_1 score; macro F_1 and micro F_1 are two of the most popular ones. Their definitions are as follows:

Micro F_1 : Calculates metrics globally by counting the total true positives, false negatives and false positives.

Macro F_1 : Calculates a F_1 score for each category, and then average these F_1 scores.

The Micro F_1 score is highly related to the most common classes, while Macro F_1 treats each category equally. Considering the imbalanced class distribution of the FNC-1 dataset, Macro F_1 is clearly a better choice. Only a system that performs great on all of the four classes can get a good Macro F_1 score. To explore the human performance on the FNC-1 dataset, Hanselowski et al. [17] asked 5 human annotators to manually label 200 instances; the results are listed in Table 3.

	Macro-F_1	Agree-F_1	Disagree-F_1	Discuss-F_1	Unrelated-F_1
Human Performance	.754	.588	.667	.765	.997

Table 3: Human performance on the FNC-1 dataset in terms of F_1 score.

4. Concepts and Models

In this chapter, we describe some concepts and models for the stance prediction task. This chapter is constructed as follows. First, we introduce some methods that are commonly used to represent text documents. Then, we cover some details about classifiers used in this report. We also study the ensemble strategy and investigate methods to tackle the imbalanced class problem. Finally, we discuss “overfitting” and multiclass classification.

4.1 Statistical Based Text Representation

Since most of the machine learning algorithms considered here require a fix-length vector as input, how to effectively transform a word, a sentence, a paragraph or a document into a vector is one of the important topics in the natural language processing community today. Some proposed models showed to perform well are discussed below.

4.1.1 Bag-of-words Model

The bag-of-words (BOW) model is a simplifying representation that represents a text as a bag of its words. For instance, the BOW model represents the sentence “Monica likes to shop. Rachel likes to shop too.” as {‘Monica’: 1, “likes”: 2, “to”: 2, “shop”: 2, “Rachel ”: 1, “too”: 1}. This type of feature is called term frequency, the number of terms that appear in the text. As simple and clear as the BOW model is, it has some limitations. First, the BOW model disregards spatial information in the text, e.g., word co-occurrence. Apart from that, common words like “I”, “a”, and “the” almost always have a high term frequency. In addition, the BOW model ignores the order of the words, which is quite important because sometimes the order of the words may change the meaning of the whole text. For example: “That is great.” and “Is that great?” have different meanings while their BOW representations are exactly the same. Thus, the BOW model fails to capture the semantic or syntactic relationship between two words.

4.1.2 N-Gram Model

As an alternative of the BOW model, the N-Gram model can store spatial information in the text. For example, for a sentence “Monica likes to shop. Rachel likes to shop too.”, a bi-gram model transforms it into {“Monica likes”: 1, “likes to”: 2, “to shop”: 2, “Rachel likes”:1, “shop too”: 1}. By storing word co-occurrence information, n-gram models usually outperform BOW models. One thing to notice is that when n equals 1 (also called unigram), a n-gram model equals a BOW model. As n increases, the vocabulary size of the model increases. The above example shows a word n-gram model. In some cases, we might need to do character n-grams. Instead of regarding each word as a single unit, character n-gram models take a character as a single unit. Apart from that, the logic stays unchanged.

4.1.3 Term Frequency-Inverse Document Frequency Model

Term frequency-inverse document frequency (TFIDF) is another statistical based representation for text. As the name suggests, term frequency is the number of terms that appear in the text. The BOW model and the N-gram model only take term frequency into consideration, which means commonly used words like “I”, and “the” always have a huge influence in the representation. The TFIDF model introduces Inverse Document Frequency, which measures how much information a word provides, i.e., a common word appears in plenty of documents while a rare word only exists in some particular documents. Intuitively speaking, a word that appears a lot of times in a document but only shows up in several documents is considered to be an important word and thus has a significant weight. By combing the term frequency and inverse document frequency, TFIDF reduces the effect of the commonly used words or stop words. There exist many variants of TFIDF models. A commonly used one is defined as follows:

$$TF(t, D) = Count(terms\_number\_in\_the\_document)$$

$$IDF(t, D) = \log \frac{Count(total\_document\_number)}{Counte(document\_number\_where\_t\_appears)}$$

$$TFIDF = TF * IDF$$

4.1.4 Cosine Similarity

After obtaining the representations of two documents using a BOW/N-gram/TFIDF model, we can calculate the similarity between two documents with pairwise metrics. One of the most commonly used one is cosine similarity:

$$\cos(\vec{u}, \vec{v}) = \frac{\vec{u} \cdot \vec{v}}{\| \vec{u} \| * \| \vec{v} \|},$$

Where $\cdot$ represents dot multiplication and $\| vector \|$ gets the L2-norm of the vector. Cosine similarity values range from 0 to 1. Values close to 0 represent dissimilar documents while values close to 1 represent very similar documents.

4.1.5 Latent Semantic Indexing

Latent Semantic Indexing (LSI) [18] is a topic model that uses a matrix decomposition technique called singular value decomposition (SVD) to reduce the columns of the TFIDF matrix constructed from multiple documents. In this TFIDF matrix, each row represents a document while each column corresponds to the TFIDF weight of a term. After applying SVD, each column in the reduced matrix represents a semantic topic. Document similarity can then be computed by calculating the cosine similarity between two LSI vectors.

4.2 Word Embedding

4.2.1 Word2Vec

The problem of the aforementioned representations is that they fail to model the semantic meaning of a word. Word2Vec [19] is a group of shallow neural networks that take a large corpus of text as its input and output word embeddings. There are two types of architecture: continuous bag-of-words or skip-gram, where the former models predict the current word from a window of surrounding context words and the latter models uses the current word to predict the surrounding window of context words. By considering the context information, Word2Vec models can capture semantic and syntactic relationships between words. A good example is as follows:

$$V(queen) - V(woman) \approx V(king) - V(man)$$

There is some research about why the word2vec model works so well. Goldberg and Levy [20] argued that the loss function of the word2vec model makes words occurring in similar contexts likely to have similar embeddings. For example, if the training corpus contains many sentences like “The cat sits on the floor” and “The dog sits on the floor”, the word2vec model starts to realize that “cat” and “dog” may refer to similar things and map them closely in the word embedding space.

4.2.2 Glove

Glove is another kind of model for distributed word representation which combines the advantages of global matrix factorization methods and local context window methods [21].

4.2.3 Fasttext

Bojanowshki et al. [22] argued the importance of sub-word information and proposed a model that learn representations for character n-grams. The word embedding is represented as the sum of the n-gam vectors. Because word formation follows rules in plenty of languages, using character level information improves the vector representations, especially when the word is rare in the training corpus. As simple as this idea is, this extension of the SkipGram model works pretty well on various languages. The simplicity also enables the model to be trained fast.

4.2.4 More Embeddings

The big success of word embedding has led it to become a basic component of most natural language models today. Recently, researchers have been working on getting more powerful representation vectors. Embeddings from Language Models (Elmo) proposed by Peter et al. [23] improves various state-of-the-art models for plenty of tasks, including question answering, textual entailment, named entity extraction, and sentiment analysis. Unlike traditional word type embeddings, Elmo assigns each token a representation that is the function of the entire input sentence.

OpenAI demonstrated that Generative Pre-Training (GPT) helps many models gain performance for a large variety of natural language processing tasks [24]. One thing to notice is that GPT uses a left-to-right architecture, where every token can only attend to previous tokens in the the Transformer.

Recently, Devlin et al. [25] introduced a new bidirectional language model, Bidirectional Encoder Representations from Transformers (BERT), which achieves new state-of-the-art performances on a wide range of natural language processing tasks, including natural language inference, question answering, named entity recognition, etc. The representations are trained by jointly conditioning on both left and right contexts.

All of these representations are pretrained on a large text corpus with a deep language model and require fine-tuning for a supervised downstream task.

4.3 Models

4.3.1 Logistic Regression

In machine learning concept, logistic regression is a linear classifier that uses a logistic function to estimate the probability. The basic formulation is as follows:

$$P(y=1) = \frac{1}{1+e^{-z}}$$

$$z = \sum_{i=0}^{n} w_i x_i$$

Where W is the learnable weight and X is the input. If the probability is bigger than 0.5, logistic regression predicts 1. Otherwise, logistic regression outputs 0.

4.3.2 Multinomial Naive Bayes

Multinomial Naive Bayes is a probabilistic classifier that based on Bayes’ theorem with a naive assumption that features are independent [26]. The classifier makes the predictions by returning the class with the maximum posterior probability given a document d. We denote this as P(c|d). Bayes' theorem is joined by:

$$P(c \mid d) = \frac{P(d \mid c)P(c)}{P(d)},$$

We can ignore P(d) since it's a common to all of the classes. Given a dataset, we can estimate P(c). Naive Bayes is built based on the BOW representations that ignores the word order. In addition, Naive Bayes assumes that features are independent, which means we can break down P(d|c):

$$P(d \mid c) = \prod_i P(w_i \mid c),$$

Where $P(w_i \mid c)$ stands for the probability of a word appears when the label is c. $P(w_i \mid c)$ is easy to calculate by dividing the total appear times of a word in class c by the total word counts in class c.

4.3.3 Support Vector Machine

Support Vector Machine (SVM) [27] is a model that maps data samples in space so that instances from separate classes are divided by a gap that is as wide as possible. By applying kernel tricks [28], the SVM can be used as a non-linear classifier.

4.3.4 Random Forest

The Random Forest [29] is an ensemble learning method. By applying a bagging technique to tree learners, each decision tree in the forest is trained with a different subset of data randomly sampled from the training set. To take a step further, we can train each decision trees with a random subset of features. The basic assumption behind the Random Forest model is that a single tree might overfit the training set, but the average of many trees do not as long as these trees are not correlated. The bagging and random subset of features are key, because each tree in the forest is trained with a different subset of data samples and a different subset of features [30].

4.3.5 Gradient Boosting Decision Trees

The Gradient Boosting Decision Trees (GBDT) [31] is a prediction model in the form of an ensemble of weak models. A weak model is a type of learner that perform slightly better than a random guess. Given training data X, labels y, the goal of epoch n is to find a mapping function or a learner that minimize the residual $y-\sum_{i=1}^{n-1} f_i(x)$. The final prediction of a GBDT with m trees is the sum of predictions from each tree: $y_{pred}=\sum_{i=1}^{m} f_m(x)$. The boosting algorithm forces the model to focus more on the misclassified samples predicted by the previous learners. There are two variants of GBDT, Xgboost and LightGBM, that work well on most tasks.

4.3.6 Multilayer Perceptron

A multilayer perceptron (MLP) is a class of feedforward neural networks with several layers of nodes: an input layer, one or more hidden layers and an output layer [32]. By applying non-linear activation in the hidden layers, an MLP can distinguish data that is not linearly separable. The most common activation include the sigmoid function, $\sigma(x)=\frac{1}{1+e^{-x}}$, the hyperbolic tangent function, $\tanh(x)=\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$, and the rectifier function, $\mathrm{Re}\,lu(x)=\max(0,x)$. With linear activation, any layers of the MLP can be equally replaced by a two-layer MLP. These non-linear activation functions give the MLP more representation power.

4.3.7 Recurrent Neural Network

A recurrent neural network (RNN) is a class of neural networks designed to deal with sequence data. Given an input sequence $X=[x_1,x_2,...,x_t,...,x_N]$, for each time step t,

$$h_t=f(W_h x_t+U_h h_{t-1}+b_h),$$

where h_t is the hidden state of a RNN cell, x_t is the input, h_{t-1} is the hidden state calculated in the last time step, and f is the activation function. The final hidden state of a RNN h_N is often used as the representation of the whole sequence.

When dealing with long sequence input, RNN has the problem of gradient vanishing or gradient exploding. Artificial neural networks are trained with gradient-based methods and backpropagation, which updates the weights proportionally to the partial derivative of the loss function. The vanishing gradient problem means that the gradient of the front layers becomes extremely small. The gradient of the front layers is calculated using the chain rule which multiplies a small number many times when the gradient of the top layers is small. This vast drop of the gradient makes the training process for the front layers very slow, or even stops the training process. The exploding gradient problem is similar, except that the gradient of the front layers becomes extremely large. The exploding gradient problem makes the training process unstable. To deal with the exploding gradient problem, we can set a maximum value and clip the gradient to it.

4.3.8 Long-Short Term Memory

To address the problems of vanishing and exploding gradient, Hochreiter and Schmidhuber [33] proposed the Long-Short Term Memory (LSTM) Network. Compared to a traditional RNN, a LSTM unit is composed of a cell, an input gate, a forget gate, and an output gate. Simply speaking, the input gate controls how much new information should be added to the cell memory state. The forget gate controls how much old information should be discarded. The output gate controls how much information should be sent to the output. With the memory state and these three gates, a LSTM unit has more power to regulate the information flowing into and out of the cell. What’s more, the forget gate allows errors to flow backwards easily, and thus a LSTM network can process several hundreds or thousands of steps. More specifically, a LSTM cell is defined as follows:

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i)$$

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f)$$

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o)$$

$$c_t = f_t \otimes c_{t-1} + i_t \otimes \tanh(W_c x_t + U_c h_{t-1} + b_c)$$

$$h_t = o_t \otimes \tanh(c_t),$$

where σ is the sigmoid function, tanh is the hyperbolic tangent function, and $\otimes$ stands for element-wise multiplication. c_t is the memory state and h_t is the output of the LSTM cell at time step t. W_*, U_* and b_* are the learnable parameters of the LSTM network.

4.3.9 Gated Recurrent Unit

The Gated Recurrent Unit (GRU) is another gating mechanism introduced by Kyunghyun Cho et al. [34]. The GRU can be viewed as a simplified variant of a LSTM with fewer parameters. The GRU has been shown to have similar performance with a LSTM on some tasks.

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z)$$

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r)$$

$$h_t = z_t \otimes h_{t-1} + (1 - z_t) \otimes \tanh(W_h x_t + U_h (r_t \otimes h_{t-1}) + b_h),$$

where σ is the sigmoid function, tanh is the hyperbolic tangent function, and $\otimes$ stands for element-wise multiplication. W_*, U_* and b_* are the learnable parameters of the GRU network.

4.3.10 Bi-directional RNN

A Bi-directional RNN models the sequence based on past and future contexts. To achieve this, one can simply concatenate the outputs of two RNNS, one processing the sequence from the beginning to end of the text, and the other one from the end to the beginning of the text. This simple trick is shown to be effective in many tasks [35].

4.3.11 Convolutional Neural network

A Convolutional Neural Network (CNN) [36] is a class of feed-forward neural networks that use a variation of MLP that only receives a restricted subarea of the previous layer. Because of the translation invariance characteristics of images, CNNs perform greatly on tasks like image classification. To make CNNs work for text, Kim et al. [37] introduced a TextCNN structure that used CNNs for sentence classification and achieved great success. The TextCNN uses a 1D CNN with various kernel sizes to capture local information between words. However, this architecture fails to capture long-distance word interactions.

4.3.12 Embedding Bag

Inspired by the idea of Fasttext [38], a straightforward approach is to use the mean of word embeddings to represent a headline and an article. The formulation is as follows:

$$h = \frac{1}{n}\sum Glove(h_i)$$
$$a = \frac{1}{m}\sum Glove(a_i)$$

Then we can concatenate h and a to get a 1 * 2d vector and feed it to a MLP classifier.

4.3.13 Independent Encoding

Because of the simplicity, the Embedding Bag model ignores some information like word order and synthetics that may be vital for the stance detection. To address this, we use two independent LSTMs to encode the headlines and the articles. After passing a headline to a LSTM encoder, the final hidden state is returned as the semantic representation of the headline. A similar encoding is generated for the corresponding article using an independent LSTM. These encodings are concatenated and then passed to a MLP classifier with one hidden layer and the RELU activation. We show the structure of independent encoding (IE) as follows:

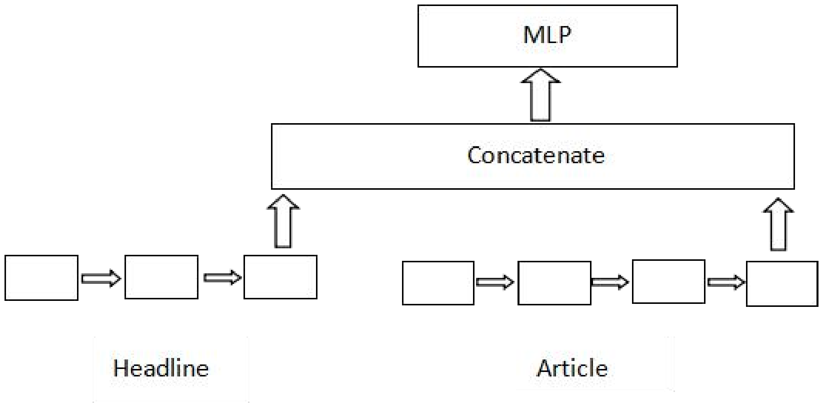


Figure 2: Illustration of the Independent Encoding.

4.3.14 Conditional Encoding

Rocktschel et al. [39] introduced conditional encoding to deal with the RTE task. We can condition the encoding of the article on the encoding of the headline or vice-versa. For the former situation, this can be simply implemented by passing the final state of the headline encoder as the initial state for the article encoder. The final hidden state of the article encoder is then fed to a MLP classifier. For the latter situation, the same logic applies. The following chart can represent one type of conditional encoding (CE):

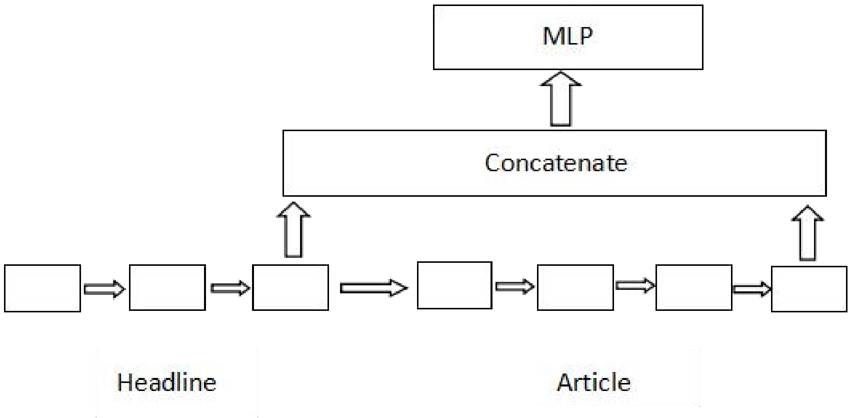


Figure 3: Illustration of the Conditional Encoding.

4.3.15 Bidirectional Conditional Encoding

We can also apply the bi-direction trick to the conditional encoding. Instead of using one direction LSTMs, bidirectional conditional encoding uses bidirectional LSTMs as the encoders. As in conditional encoding, the final hidden state of the first bidirectional LSTM is passed as an initial state for the second bidirectional LSTM. One thing to notice about bidirectional conditional encoding is that the final state of the article/headline encoder is two vectors. We concatenate these two vectors and feed it into a MLP classifier.

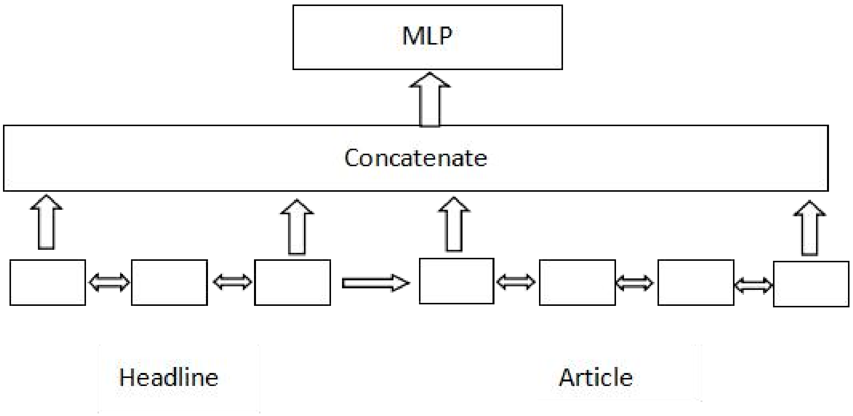


Figure 4: Illustration of the Bidirectional Conditional Encoding.

4.3.16 Stacked Conditional Encoding

These aforementioned models use a one layer RNN. It is likely that these shallow models are not powerful enough to capture the information key to stance detection. To extract high-level interaction between an article and a headline, we employ the stacked two layers of conditional encoding structure. The first layer LSTM incorporates contextual information into representation vectors, and the second layer LSTM uses them to extract high-level semantic meanings. One thing to notice is that the input of the second layer comes from the original word embedding and vectors with contextual information, which gives LSTM the power to select significant features for stance detection. We plot the structure of stacked conditional encoding

(StackedCE) as follows:

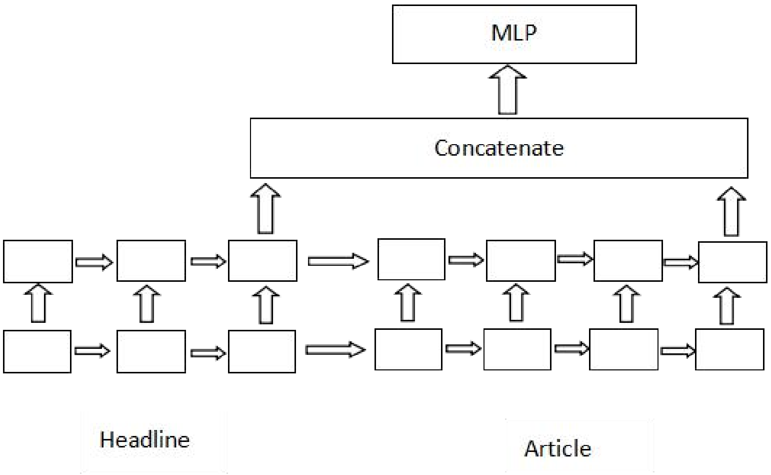


Figure 5: Illustration of the Stacked Conditional Encoding.

4.3.17 Enhanced Sequential Inference Model

Considering the similarity between the stance detection task and the NLI task, it is worth exploring the performance of the state-of-the-art NLI models on the FNC-1 dataset. The Enhanced Sequential Inference Model (ESIM) [10] achieves 88.0 accuracy on the test set of the Standford Natural Language Inference (SNLI) dataset. Because of the excellent performance, ESIM is used as a baseline by many later researchers for the NLI task and other related NLP topics. Since we conducted extensive experiments with ESIM, we cover the basic structure and ideas behind this well-performed model. ESIM is composed of embedding layers, input encoding part, local inference modeling, and inference composition. For the stance detection task, ESIM takes a headline, article as input and predicts a stance.

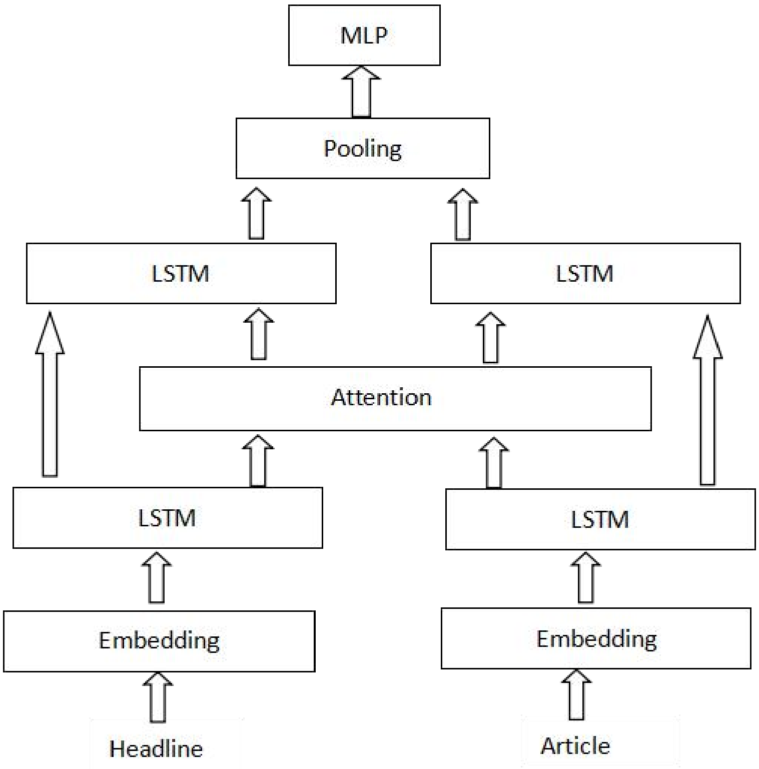


Figure 6: Illustration of the ESIM.

The original version of ESIM simply uses a word embedding layer that maps a token to a vector. To better model the input, we also include a named entity recognition (NER) embedding, a part-of-speech (POS) embedding, and an exact match flag. The NER embedding maps a NER tag to a vector, and the POS embedding maps a POS tag to a vector. For each word in a claim or an article, if it appears both in the claim and in the article, the exact match flag is set to 1. Otherwise, the exact match flag is set to 0. For each word, we concatenate the word embedding, NER embedding, POS embedding, and the exact match flag as our final word representation.

In annotation, we have a headline $H = [h_1, h_2, ...h_n]$ with length n and an article $A = [a_1, a_2, ...a_m]$ with length m. The final word feature $\bar{h_i} / \bar{a_i}$ for the word h_i / a_i is detailed as follows:

$$\bar{h}_i = [wordEmb(h_i); posEmb(h_i); nerEmb(h_i); flag(h_i)]$$

$$\bar{a}_j = [wordEmb(a_j); posEmb(a_j); nerEmb(a_j); flag(a_j)]$$

where wordEmb represents word embedding, posEmb represents POS embedding, nerEmb represents NER embedding, flag() represents the exact match flag, and [;] is the concatenation operation. The word embedding is pretrained, while the POS embedding and NER embedding is learned through the training process.

We also use a bidirectional LSTM (BiLSTM) as our input encoding layer, which incorporates contextual information into the word feature. With this encoding layer, we can encode a headline for each token.

$$\vec{h}_i = \overrightarrow{LSTM}(\vec{h}_{i-1}, \bar{h}_i)$$

$$\overleftarrow{h}_i = \overleftarrow{LSTM}(\overleftarrow{h}_{i+1}, \bar{h}_i)$$

Meanwhile, we can apply the same encoder to an article.

$$\vec{a}_j = \overrightarrow{LSTM}(\vec{a}_{j-1}, \bar{a}_j)$$

$$\overleftarrow{a}_j = \overleftarrow{LSTM}(\overleftarrow{a}_{j+1}, \bar{a}_j)$$

To model local inference, ESIM takes a soft alignment layer that calculates the similarity between two words.

$$attn_{i,j} = \cos(\overleftrightarrow{h}_i, \overleftrightarrow{a}_j) \quad ,$$

where $\overleftrightarrow{h}_i$ is the concatenation of $\vec{h}_i$ and $\overleftarrow{h}_i$, $\overleftrightarrow{a}_j$ is the concatenation of $\vec{a}_j$ and $\overleftarrow{a}_j$, and cos() stands for the cosine function. After computing the attention weight, we can obtain the local relevance of a headline and an article. For a token in a headline, the relevant semantics in the article is computed with the following equation:

$$\tilde{h}_i = \sum_{j=1}^{m} \frac{\exp(attn_{i,j})}{\sum_{k=1}^{m} \exp(attn_{i,k})} \overleftrightarrow{a}_j ,$$

where $\tilde{h}_i$ is a weighted sum of the $\overleftrightarrow{a}_j$. Intuitively, for each token in a headline, the relevant information in the article is extracted. We also perform this operation on each token in an article to extract relevant information from the headline.

$$\tilde{a}_j = \sum_{i=1}^{n} \frac{\exp(attn_{i,j})}{\sum_{k=1}^{n} \exp(attn_{k,j})} \overleftrightarrow{h}_i$$

To further enhance local inference, difference and element-wise multiplication are introduced in the model. The expectation is that such operations could help sharpen local inference information between token pairs and capture local relationships such as contradiction. Finally, the output of this component is generated by concatenating the original vectors $\overleftrightarrow{h}_i$ and $\tilde{h}_i$ with difference and element-wise product.

$$h_{enh,i} = [\overleftrightarrow{h}_i; \tilde{h}_i; \overleftrightarrow{h}_i - \tilde{h}_i; \overleftrightarrow{h}_i \otimes \tilde{h}_i]$$

$$a_{enh,j} = [\overleftrightarrow{a}_j; \tilde{a}_j; \overleftrightarrow{a}_j - \tilde{a}_j; \overleftrightarrow{a}_j \otimes \tilde{a}_j],$$

where $\otimes$ stands for element-wise multiplication and [;] is the concatenation operation. The proposers of the ESIM model argue that this enhancement improves the model by modeling some high-order interaction.

To capture local inference information, another bidirectional LSTM is introduced here.

$$\overrightarrow{h}_{c,i} = \overrightarrow{LSTM}(\overrightarrow{h}_{c,i-1}, h_{enh,i})$$

$$\overleftarrow{h}_{c,i} = \overleftarrow{LSTM}(\overleftarrow{h}_{c,i+1}, h_{enh,i})$$

$$\overrightarrow{a}_{c,j} = \overrightarrow{LSTM}(\overrightarrow{a}_{c,j-1}, a_{enh,j})$$

$$\overleftarrow{a}_{c,j} = \overleftarrow{LSTM}(\overleftarrow{a}_{c,j+1}, a_{enh,j})$$

With a pooling layer, ESIM can convert the above vectors into a fix-length vector. Instead of using the summation, average-pooling and max-pooling are used together and concatenated to get the final vector, calculated as follows:

$$v_{h,ave} = \sum_{i=1}^{n} \frac{\overleftrightarrow{h}_{c,i}}{n}$$

$$v_{h,\max} = \max(\overleftrightarrow{h}_{c,i})$$

$$v_{a,ave} = \sum_{j=1}^{m} \frac{\overset{\leftrightarrow}{a}_{c,j}}{n}$$

$$v_{a,\max} = \max(\overset{\leftrightarrow}{a}_{c,j})$$

$$v = [v_{h,ave}; v_{a,\max}; v_{h,ave}; v_{h.\max}]$$

We build the final MLP classifier with v as the input. The MLP has a hidden layer with RELU activation and softmax output layer. The entire model is trained end-to-end.

4.4 Ensemble Learning

Ensemble learning is a method that uses multiple learning algorithms to obtain better predictive performance. Empirically, the more diversity among the models, the better the results are. The diversity can come from different inputs to the model, a different set of features to use, or even just a different random seed. Commonly used ensemble strategies include bootstrap aggregating (bagging), boosting, and stacking. Bagging ensures diversity by training each model using a randomly sampled subset of the training set. Boosting forces the new trained model to put more emphasis on the data samples that previous learners misclassified. Stacking involves training a multilayer algorithm that the high layers classifier are trained on based on the predictions from the low layers. Stacking has been showed to be successful on many tasks [40].

4.5 Imbalance

One of the biggest obstacles of the fake news challenge is how to deal with highly imbalanced class distribution. An Imbalanced class distribution is a common phenomenon in the real world. In this section, we cover various strategies for training and evaluating a classifier on an imbalanced dataset.

4.5.1 Balancing the class distribution

One straightforward way is to balance the class distribution. We can over-sample the minority classes or under-sample the majority classes. In the fake news challenge setting, we cannot directly use over-sampling algorithms like Synthetic Minority Over-sampling Technique (SMOTE) [41]. One alternative is to over-sample the minority classes by picking samples at random with replacement. One thing to notice about over-sampling is that the risk of overfitting goes up. Under-sampling is relatively simple to implement. We can randomly drop some data points from the majority classes. However, this means that we waste some data samples.

4.5.2 Adjusting class weight

Another approach is to adjust the loss function. A loss function or cost function is a a function that measures the distance between the true labels and predictions. One commonly used loss function for binary classification is logarithmic loss: $\log loss = -\sum_{i=1}^{m}(y_i \log p_i + (1-y_i)\log(1-p_i))$, where $y_i = 1$ or $y_i = 0$ and p_i stands for the predicted probability of the data point to be a positive sample. It is trivial to extend the logistic loss to multiclass classification, $-\sum_{c=1}^{M} y_{o,c}\log(p_{o,c})$, also known as the cross-entropy loss. The assumption is that a good model gives a big probability value to the right class label. Other popular loss functions include hinge loss, $hinge = \max(0,1-z)$ and mean squared error (MSE), $mse = \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y}_i)$. Hinge loss is used for a support vector machine and MSE is used for regression problems. Loss functions like log loss assume that each class contributes to the total loss equally. In other words, every class is equally important. By assigning higher weights to rare categories, we force the model to pay more attention to the minority classes. However, we need to decide these weights for each class. One naive approach is to set the weights to be inversely proportional to the class ratios in the dataset. The basic idea is to make data samples in each class contribute to the total loss equally.

4.5.3 Focal Loss

Based on the cross-entropy loss, Lin et al. [42] proposed a new loss function, Focal Loss, that focuses more on hard, misclassified examples. The definition is as follows:

$$FL(p_t) = -(1 - p_t)^{\gamma} \log(p_t)$$

Where γ is a tunable parameter that adjusts the rate of well-classified examples. One thing to notice is that when $\gamma = 0$, Focal loss is equal to cross-entropy loss. For the hard-to-classify samples, p_t is usually small and $(1 - p_t)^{\gamma}$ is close to 1 and thus the loss stays almost unaffected. For the easy-to-classify examples, $(1 - p_t)^{\gamma}$ goes to 0 and the loss is down-weighted. Compared to manually adjusting the class weight, Focal Loss automatically pays more attention to the misclassified examples.

4.6 Strategies to reduce Overfitting

Overfitting refers to the phenomenon that a learner performs perfectly on the training set while generalizing poorly on unseen data. Overfitting happens when a model tries to “memorize” the training data rather than “learning” or “mining” useful patterns. To prevent overfitting, many strategies are available (e.g., model selection, early stopping, weight decay, dropout). The basic idea is to penalize complex models or evaluate the performance of a model on unseen data [43].

4.6.1 Model selection

There exist a variety of learners that are suitable for different types of tasks. Depending on the task and the dataset we have, we may choose a different algorithm. After choosing a learner, a slight change of one single parameter (even a different random seed) may lead to a brand new model. Since most of the learners can be trained to perform perfectly on the train set, we need a brand new data set or a set of unseen data to evaluate the generalization ability of a model. One natural approach is to pick out some data samples from the train set randomly. This new data set is often called the validation set or development set. The sampling ratio typically takes the values of 0.1, 0.2 or 0.3. With a validation set available, we can tune the hyper-parameters of a leaner. The model that has the best performance on the validation set is selected, which is the process of model selection.

4.6.2 Early stopping

Early stopping is a popular strategy used in the training process of GBDT and neural networks. It uses the learner's performance on a validation set as a monitor. If the performance of a model on the validation set stops improving for a few epochs, the training stops. The performance drop is a signal that the model is starting to overfit. The simplicity and efficacy make early stopping very helpful.

4.6.3 Weight decay

Weight decay is a method to reduce overfitting in neural networks by penalizing bigger weights or complex models. To achieve that, it adds a term in the loss function, which usually takes the form of multiplying the norm of all the weights by a constant number (usually 1e-6 to 1e-9). The assumption is that compared to single models, complex models are more likely to overfit.

4.6.4 Dropout

Dropout is a commonly used regularization strategy in neural networks. It works by randomly dropping some units in each layer when training a neural network, which can prevent complex co-adaptations on training data [44].

4.7 Multiclass classification

Multiclass classification refers to the problem of classifying data samples into one of three or more classes. While some algorithms are designed to allow the prediction for more than two classes, others classifiers can be turned into multinomial classifiers by some strategies.

4.7.1 One-vs-all or One-vs-rest

One-vs-all [16] is a method that transforms a multi-class problem into multiple binary problems (classifying the data samples into two categories). For each class, it trains a classifier that regards data samples of that class as positive samples and other samples as negatives. After applying the One-vs-all strategy we get C classifiers, where C is the total number of classes. When making predictions, all of these C classifiers return a confidence score for the decisions. For each data sample, the classifier with the highest confidence score is chosen to predict the final label.

4.7.2 One-vs-one

One-vs-one [16] is a strategy that learns a classifier to distinguish data samples between any two different classes, which means we get C*(C-1) classifiers after applying One-vs-one for a C way multiclass classification.

5. Experiments and Results

In this chapter, we describe the experiment that we have conducted along with the results obtained. All of the models are trained on the Fake News Challenge dataset [2]. The performance on the validation set is used to select the best models. We report the best macro F_1 [17] values on the test set. To better compare the performance of different models and see how various models perform on different classes, we also report the class-wise F_1 [17] values here.

5.1 Winning solutions

First, we briefly introduce the ideas behind the top 3 wining solutions. The top one team Sloat in the SWEN [45] used a weighted average model of a GBDT and a deep CNN. Their processing code generates a set of five features extracted from headlines and articles, including counting features, TFIDF features, SVD features, Word2Vec features, and sentiment features. These features are concatenated together to train their GBDT model. Their CNN model uses several layers of 1D convolutional layers to extract features and feed these features into a MLP classifier.

The second-ranked team Athene [46] uses an ensemble of different MLP classifiers. The final predictions are based on the votes. Features like unigrams, topic models based on non-negative matrix factorization, latent Dirichlet allocation, and latent semantic indexing are used to train their MLPs.

Using a MLP classifier trained on term frequency vectors and the cosine similarity between a pair of TFIDF vectors of a headline and the associated article, team UCL Machine Reading [47] won third place of the FNC-1 competition. To reduce the size of the network, they set the vocabulary size of the TFIDF transformer to 5000. In other words, only the 5000 most frequent terms in the training set are used for prediction.

Table 4 shows the results of the top three systems (results are from [17]).

	Macro-F_1	Agree-F_1	Disagree-F_1	Discuss-F_1	Unrelated-F_1
Sloat in the SWEN	.582	.539	.035	.760	.994
Athene	.604	.487	.151	.780	.996
UCL Machine Reading	.583	.479	.114	.747	.989

Table 4: Performance for the top 3 wining teams.

5.2 Implementation details and results

Considering the simplicity and effectiveness of the TFIDF representations, we conducted experiments to see how the TFIDF vectors work for FNC-1. The features we used are similar to the features used in the team UCL Machine Reading. Instead of using term frequency vectors, we use TFIDF vectors. Apart from that, n-grams (up to 3) are also used when obtaining TFIDF vectors. The TFIDF transformer is trained with all of the headlines and articles. Finally, the max features of the TFIDF transformer are set to 10,000. The cosine similarity between a pair of TFIDF vectors is kept. For the classifiers, we choose SVM, Logistic Regression, MLP, Random Forest, and GBDT, all of which are introduced in the Concepts and Models chapter. SVM and Logistic Regression use the default parameters. Our MLP has one hidden layer with unit size equals to 100, the maximum training iteration is set to 10, and early stopping is enabled. Other than that, the other default parameters are used. The Random Forest Classifier consists of 50 trees, while GBDT consists of 100 trees with learning rate set to 0.1. The processing and training code are implemented with sklearn [48], because of the convenience and the clear documentation. To get a GBDT, LightGBM is used. Table 5 shows the results.

	Macro-F_1	Agree-F_1	Disagree-F_1	Discuss-F_1	Unrelated-F_1
SVM	.544	.47	.01	.73	.96
Logistic Regression	.530	.43	.00	.74	.95
MLP	.503	.41	.01	.66	.93
Random Forest	.461	.35	.00	.58	.91
LightGBM	.562	.51	.01	.76	.97

Table 5: Performance for the models trained with TFIDF features.

From the above results, we observe something interesting. Firstly, all of the models do an excellent job of distinguishing between related pairs and Unrelated pairs, which shows the effectiveness of the TFIDF representations. In addition, all of the existing models, including the models in the winning solutions, perform poorly on the Disagree class (the best system only gets a F_1 score of .151). This may be caused by the limited number of Disagree samples in the dataset (only about 850 pairs).

To show that neural networks can perform stance detection, we trained many RNN based models on the FNC-1 dataset, including IE, CE, StackedCE, and ESIM. In addition, we trained an Embedding Bag model that only uses the word embedding information. The details of these models are introduced in the Concept and Models Section. All of the deep learning models are implemented with Pytorch [49].

The embedding layer used by all of the LSTM based neural networks are exactly the same, a word vector concatenated with a POS vector, a NER vector, and a match flag. We used pretrained 100D Glove 6B vectors [21] to initialize our word embedding. NLTK [50] is used to do tokenization and extract the POS and NER tags of each token in a headline or an article. Each POS or NER tag is mapped to a 10-dimensional vector. The POS embedding and NER embedding is initialized using a normal distribution with zero mean and standard deviation of 1. All of the embeddings are updated during the training process. We write the preprocessing code to make sure that the headlines/articles are of the same length in the same batch. On top of that, any headline with more than 50 words is truncated to 50 words, while article length is cut to 100 words. Shorter texts are padded with zeros. For optimization, we use Adam [51] with learning rate set to 1e-3. The first momentum is set to be 0.9 and the second is set to 0.999. The weight decay is set to 1e-6. All hidden states of BiLSTMs have 100 dimensions. To prevent overfitting, a dropout of 0.5 is applied to the hidden layer of the MLP classifier. Focal loss with gamma of 5 is used as the loss function to be minimized.

With the above default settings, the performance of various models on the test set is reported in the following table:

	Macro-F_1	Agree-F_1	Disagree-F_1	Discuss-F_1	Unrelated-F_1
Embedding Bag	.313	.08	.00	.36	.81
IE	.498	.41	.02	.59	.97
CE, H->A	.528	.45	.12	.57	.97
CE, A->H	.515	.42	.07	.59	.97
StackedCE, H->A	.484	.39	.01	.56	.97
StackedCE, A->H	.456	.40	.04	.43	.96
ESIM	.600	.53	.19	.70	.97

Table 6: Performance of various models on the test set.

From Table 6, we can see that the extracted information by directly feeding LSTMs a pair of (headline, article) is not powerful enough to solve a complex problem like stance detection. Compared to independent encoding, conditional encoding on the headlines or articles does help, but the improvement is limited. Stacking two layers of conditional encoding actually damages the performance. The reason is not clear and worth exploring further. We leave this for future work. One interesting thing is that ESIM performs quite well. The macro F_1 value is similar to the top 3 winning systems. Although ESIM was proposed for NLI, it seems to be a choice for stance detection. The great performance of ESIM is not surprising since NLI and stance detection are two types of tasks with many commonality, e.g. the format of the input is a pair of texts, and the goal is to predict a label that explains the relationship between the text pair.

6. Further Experiments and Analysis

Since ESIM is one of our best models, we design several experiments to further explore the potential of this architecture.

6.1 Article Length

The maximum length for a headline is set to 50, which is longer than any headline in the dataset. All of the information in the headlines are considered, and the article length controls how many words in the article to be used by our neural networks. The articles/headlines length distributions of the FNC-1 training set test set are as follows:

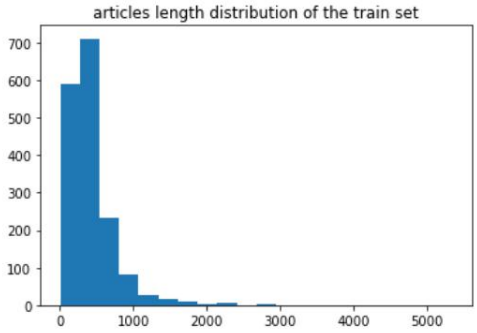


Figure 7: Histogram of articles length distribution of the training set.

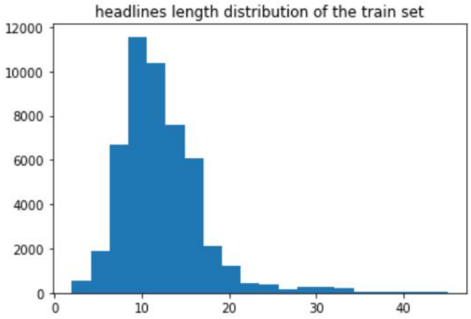


Figure 8: Histogram of headlines length distribution of the training set

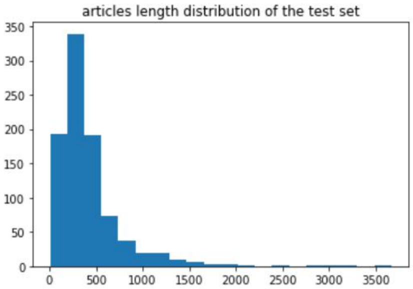


Figure 9: Histogram of articles length distribution of the test set

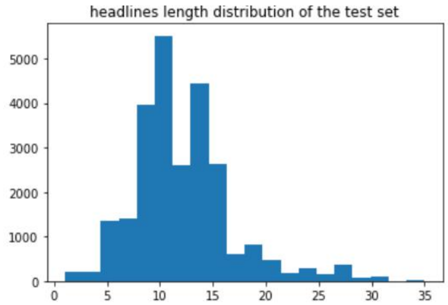


Figure 10: Histogram of headlines length distribution of the test set

We tune the ESIM over the following article body length: 50, 80, 100, 120, 150. The results are summarized in Table 7.

	Macro-F_1	Agree-F_1	Disagree-F_1	Discuss-F_1	Unrelated-F_1
Article length: 50	.554	.51	.02	.71	.91
Article length: 80	.552	.51	.05	.68	.97
Article length: 100	.600	.53	.19	.70	.97
Article length: 120	.561	.50	.12	.65	.97
Article length: 150	.575	.50	.14	.68	.97

Table 7: Performance vs Article length

From the results, we can see that the F_1 score of the Disagree class is affected mostly by the length of the article body length, which may happen when the first few sentences in the article seem to agree or discuss a headline while the following sentences contradict the attitude in the first few sentences. In addition, we observe that the ESIM with short article body performs well. The first 100 words are good enough to determine the stance perspective of a (headline, article) pair. This is understandable since people tend to state their opinions in the first few paragraphs.

6.2 POS/NER embedding dimensions

The dimensions of the POS or NER embedding in the ESIM model is also a hyper-parameter to tune. In theory, as the dimension increases, the importance of POS and NER tags increases, while the importance of word embeddings decreases.

	Macro-F_1	Agree-F_1	Disagree-F_1	Discuss-F_1	Unrelated-F_1
Embedding size: 5	.559	.51	.13	.64	.96
Embedding size: 10	.600	.53	.19	.70	.97
Embedding size: 15	.575	.48	.16	.69	.97
Embedding size: 20	.573	.51	.10	.71	.98

Table 8: Performance vs Embedding size

Table 8 shows how the performance of ESIM varies according to the embedding dimensions of POS vectors and NER vectors. From the experiments training ESIM with different embedding dimensions for POS and NER tags, we observe no significant improvement on the test set. Given the relatively small vocabulary size of POS tags and NER tags (the vocabulary size for POS tag is 36 and the vocabulary size for NER tag is 6), setting the embedding dimensions to 10 seems reasonable.

6.3 Focal Loss vs Cross-Entropy Loss

To show the efficiency of Focal Loss for this imbalanced dataset, we train an ESIM with cross-entropy Loss (class weight is set to be Agree: 3, Disagree: 3, Discuss: 3, Unrelated: 1) and an ESIM with Focal Loss. Then we compare the performance of these two models. Table 9 reports the results and shows that Focal Loss is a great loss function for the FNC-1 dataset. Notice that the F1 score for the Disagree class trained with Cross-Entropy Loss equals zero. Because of the extremely small portion of the Disagree data samples in the FNC-1 dataset, their contribution to the total loss is so small that Cross-Entropy loss ignores it. One can argue that for a different class weight (for instance, setting the class weight for the Disagree class to 10), models trained with Cross-Entropy loss may outperform models trained with Focal Loss for the FNC-1 dataset. One significant advantage of Focal Loss is that it can automatically put more focus on hard-to-classify data samples.

	Macro-F_1	Agree-F_1	Disagree-F_1	Discuss-F_1	Unrelated-F_1
Cross Entropy Loss	.512	.45	.00	.63	.97
Focal Loss	.600	.53	.19	.70	.97

Table 9: Performance vs Loss function

We further explore how the choice of gamma affects the ESIM's performance on the test set. We summarize the results in Table 10. We can see that Focal Loss with a small gamma value achieves similar performance with Cross-Entropy Loss. As gamma increases, the Focal Loss penalizes misclassified instances more heavily.

	Macro-F_1	Agree-F_1	Disagree-F_1	Discuss-F_1	Unrelated-F_1
Gamma: 3	.514	.43	.00	.66	.96
Gamma: 5	.600	.53	.19	.70	.97
Gamma: 6	.572	.51	.13	.68	.97

Table 10: Performance vs Gamma.

6.4 Character Embedding

When exploring the dataset, we found that some of the words appearing in the dataset are not in the vocabulary of the Glove pretrained vectors, which is called the out of vocabulary (OOV) problem. To deal with OOV, we add a character embedding that maps a character into a 30 dimensions vector. The model design follows the textCNN structure proposed by Kim et al. [36]. First, we pad all the tokens to the same length. Then a CNN with max pooling is used to extract character level information. The kernel size is set to be (2, 3, 4) to capture bi-gram, tri-gram, and four-gram information.

We report the results in Table11. Character embedding seems to make the model perform worse on the Disagree class.

	Macro-F_1	Agree-F_1	Disagree-F_1	Discuss-F_1	Unrelated-F_1
With char-embedding	.587	.51	.14	.73	.98
Without char-embedding	.600	.53	.19	.70	.97

Table 11: Performance of ESIM with or without char-embedding.

6.5 LSTM vs GRU

Considering the fact that the FNC-1 dataset is relatively small (around 50,000 rows) and ESIM is a very complex system (about 41m parameters), we reduce the total number of parameters by replacing all of the LSTM units with GRU units. This replacement boosts the performance of the ESIM from 0.600 to 0.606 in terms of macro F_1 score. The improvement is quite small, indicating the possibility of overfitting.

6.6 Adding Dropout Layer

One of the straightforward and effective approaches to prevent overfitting is to use a dropout layer. After the concatenating operation that combines word embedding, POS embedding, NER embedding, and Exact Match flag, we add a dropout layer on top of the concatenated vectors. The keep rate is set to be 0.8, which means that our ESIM has a 20% percent of the chance to set the element to be 0. There is a possibility that our model hiddenly rely on a small set of features, such as POS tags. This added dropout layer forces the model to extract information from all the features, because POS tags may be dropped during training process. The results in Table12 show that ESIM overfits the FNC-1 dataset.

	Macro-F_1	Agree-F_1	Disagree-F_1	Discuss-F_1	Unrelated-F_1
ESIM with LSTM	.600	.53	.19	.70	.97
ESIM with GRU	.606	.55	.21	.69	.97
ESIM with GRU + Dropout	.611	.56	.23	.68	.97

Table 12: Performance of ESIM with different settings.

6.7 Ensemble

Since ESIM is such a complex model that takes a lot of time and computation to train, we want simple ensemble strategies. This means that stacking is out of our consideration. One naive way is to train multiple ESIM models with different random seeds. Although this means we need to train a model several times, this is a pretty popular method. Another approach is to save models after each epoch during the training phase. After collecting several models, the average or weighted average strategy is used to get the final predictions.

	Macro-F_1	Agree-F_1	Disagree-F_1	Discuss-F_1	Unrelated-F_1
2 ESIM models	.614	.56	.23	.69	.97
2 ESIM models + LightGBM	.617	.57	.23	.70	.97

Table 13: The results of ensemble learning.

Table 13 shows two examples of our experiments. The first row in Table 13 is a weighted average prediction of two ESIM models trained with different random seeds. The second row combines an additional GBDT model based on TFIDF features.

The weight of each classifier is related to the macro F_1 scores on the evaluation test. The higher the macro F_1 score, the bigger the weight. Although LightGBM performs worse than ESIM, the diversity of TFIDF features and LightGBM helps the system perform better. From Table13, we can also see that the improvement of the ensemble strategy on FNC-1 dataset is relatively small.

To find out why the ensemble strategy does not work as expected, we further analyze the predictions for an ESIM model and a LightGBM model. The correlation coefficients are as follows:

	LGB:Agree	LGB:Disagree	LGB:Discuss	LGB:Unrelated
ESIM : Agree	.736			
ESIM: Disagree		.377		
ESIM: Discuss			.797	
ESIM: Unrelated				.945

Table 14: Correlation of predictions between ESIM and LGB

The above result means that although ESIM and LGB are trained with different features, their predictions on the Agree, the Discuss and Unrelated class are highly correlated, which makes the ensemble loses it’s power.

6.8 Ideas for future work

Because of the good performance of ESIM on the FNC-1 dataset and the similarity between NLI and stance detection, one straightforward and effective direction is to build more systems that obtain state-of-the-art results on the SNLI dataset. In addition, considering the significance of pretrained word vectors for neural networks, instead of using 100 dimensional word representations, a 300 dimensional word embedding may be helpful for containing more information. Also, word embeddings trained with different methods may capture various semantic meanings. The performance comparisons between Glove, Word2Vec, and Fasttext for the FNC-1 dataset are worth exploring. Our word embedding layer could be replaced by a deep contextualized word representation, such as ELMO, GPT and BERT; there are fairly easy to use and showed to improve models’ performance for many tasks [22, 23, 24].

7. Conclusion

In this report, we have investigated algorithms trained with TFIDF features and word embedding for the stance detection task. The results obtained show that TFIDF vectors are effective representations of headlines and articles. The winning team in the FNC-1 uses a large set of hand-engineered features, including TFIDF similarities, SVD features, n-gram, sentiment features, word2vec features. However, a GBDT trained with TFIDF vectors and their similarities gets a Macro F_1 score of 0.562, which is comparable to the best systems.

We also study how neural networks work for the stance detection task. Neural networks with simple structure have poor performance. One interesting thing is that the ESIM model proposed for Natural Language Inference produces our best results, which can be explained by the similarity between NLI and stance detection tasks. This provides us with directions for future work. We are eager to see how other state-of-the-art systems for the NLI task perform on the FNC-1 dataset. All of the existing solutions perform poorly on the Disagree class. We maintain that the reason is the limited number of data samples in the FNC-1 dataset. To fully see the potential of neural networks for this task, more data samples are needed, especially for the minority classes.

The fact that models trained with different text features output highly correlated disappointing results indicates that stance detection is not an easy task. We hope more researchers focus on this area and propose better methods for this task.

References:

[1] Wikipedia contributors, "Fake news," Wikipedia, The Free Encyclopedia, https://en.wikipedia.org/w/index.php?title=Fake_news&oldid=871325018 (accessed December 3, 2018).

[2] D. Pomerleau and D. Rao. Fake news challenge. http://www.fakenewschallenge.org/. Visited on 2018-12-01

[3] Bowman, Samuel R., Gabor Angeli, Christopher Potts, and Christopher D. Manning. "A large annotated corpus for learning natural language inference." arXiv preprint arXiv:1508.05326 (2015).

[4] Bromley, Jane, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. "Signature verification using a" siamese" time delay neural network." In Advances in neural information processing systems, pp. 737-744. 1994.

[5] Vendrov, Ivan, Ryan Kiros, Sanja Fidler, and Raquel Urtasun. "Order-embeddings of images and language." arXiv preprint arXiv:1511.06361 (2015).

[6] Mou, Lili, Rui Men, Ge Li, Yan Xu, Lu Zhang, Rui Yan, and Zhi Jin. "Natural language inference by tree-based convolution and heuristic matching." arXiv preprint arXiv:1512.08422(2015).

[7] Wang, Shuohang, and Jing Jiang. "Learning natural language inference with LSTM." arXiv preprint arXiv:1512.08849 (2015).

[8] Rocktäschel, Tim, Edward Grefenstette, Karl Moritz Hermann, Tomáš Kočiský, and Phil Blunsom. "Reasoning about entailment with neural attention." arXiv preprint arXiv:1509.06664 (2015).

[9] Parikh, Ankur P., Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. "A decomposable attention model for natural language inference." arXiv preprint arXiv:1606.01933 (2016).

[10] Chen, Qian, Xiaodan Zhu, Zhenhua Ling, Si Wei, Hui Jiang, and Diana Inkpen. "Enhanced lstm for natural language inference." arXiv preprint arXiv:1609.06038 (2016).

[11] Gong, Yichen, Heng Luo, and Jian Zhang. "Natural language inference over

interaction space." arXiv preprint arXiv:1709.04348 (2017).

[12] Le, Quoc, and Tomas Mikolov. "Distributed representations of sentences and documents." In International Conference on Machine Learning, pp. 1188-1196. 2014.

[13] Kiros, Ryan, Yukun Zhu, Ruslan R. Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. "Skip-thought vectors." In Advances in neural information processing systems, pp. 3294-3302. 2015.

[14] Yang, Zichao, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. "Hierarchical attention networks for document classification." In Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pp. 1480-1489. 2016.\

[15] Powers, David Martin. "Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation." (2011).

[16] Witten, Ian H., Eibe Frank, Mark A. Hall, and Christopher J. Pal. Data Mining: Practical machine learning tools and techniques. Morgan Kaufmann, 2016.

[17] Hanselowski, Andreas, Avinesh PVS, Benjamin Schiller, Felix Caspelherr, Debanjan Chaudhuri, Christian M. Meyer, and Iryna Gurevych. "A Retrospective Analysis of the Fake News Challenge Stance Detection Task." arXiv preprint arXiv:1806.05180 (2018).

[18] Deerwester, Scott, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. "Indexing by latent semantic analysis." Journal of the American society for information science 41, no. 6 (1990): 391-407.

[19] Mikolov, Tomas, Kai Chen, Greg Corrado, and Jeffrey Dean. "Efficient estimation of word representations in vector space." arXiv preprint arXiv:1301.3781 (2013).

[20] Goldberg, Yoav, and Omer Levy. "word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method." arXiv preprint arXiv:1402.3722 (2014).

[21] Pennington, Jeffrey, Richard Socher, and Christopher Manning. "Glove: Global vectors for word representation." In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), pp. 1532-1543. 2014.

[22] Bojanowski, Piotr, Edouard Grave, Armand Joulin, and Tomas Mikolov. "Enriching word vectors with subword information." arXiv preprint

arXiv:1607.04606 (2016).

[23] Peters, Matthew E., Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. "Deep contextualized word representations." arXiv preprint arXiv:1802.05365 (2018).

[24] Radford, Alec, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. "Improving language understanding by generative pre-training." URL https://s3-us-west-2. amazonaws. com/openai-assets/research-covers/language-unsupervised/language_ understanding_paper. pdf (2018).

[25] Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. "Bert: Pre-training of deep bidirectional transformers for language understanding." arXiv preprint arXiv:1810.04805 (2018).

[26] Hand, David J., and Keming Yu. "Idiot's Bayes—not so stupid after all?." International statistical review 69, no. 3 (2001): 385-398.

[27] Cortes, Corinna, and Vladimir Vapnik. "Support-vector networks." Machine learning 20, no. 3 (1995): 273-297.

[28] Boser, Bernhard E., Isabelle M. Guyon, and Vladimir N. Vapnik. "A training algorithm for optimal margin classifiers." In Proceedings of the fifth annual workshop on Computational learning theory, pp. 144-152. ACM, 1992.

[29] Ho, Tin Kam. "Random decision forests." In Document analysis and recognition, 1995., proceedings of the third international conference on, vol. 1, pp. 278-282. IEEE, 1995.

[30] Wikipedia contributors, "Random forest," Wikipedia, The Free Encyclopedia, https://en.wikipedia.org/w/index.php?title=Random_forest&oldid=869972790 (accessed December 3, 2018).

[31] Mason, Llew, Jonathan Baxter, Peter L. Bartlett, and Marcus R. Frean. "Boosting algorithms as gradient descent." In Advances in neural information processing systems, pp. 512-518. 2000.

[32] Rosenblatt, Frank. Principles of neurodynamics. perceptrons and the theory of brain mechanisms. No. VG-1196-G-8. CORNELL AERONAUTICAL LAB INC BUFFALO NY, 1961.

[33] Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." Neural computation 9, no. 8 (1997): 1735-1780.

[34] Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate." arXiv preprint arXiv:1409.0473 (2014).

[35] Graves, Alex, and Jürgen Schmidhuber. "Framewise phoneme classification with bidirectional LSTM and other neural network architectures." Neural Networks 18, no. 5-6 (2005): 602-610.

[36] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." In Advances in neural information processing systems, pp. 1097-1105. 2012.

[37] Kalchbrenner, Nal, Edward Grefenstette, and Phil Blunsom. "A convolutional neural network for modelling sentences." arXiv preprint arXiv:1404.2188 (2014).

[38] Bag of Tricks for Efficient Text Classification

[39] Augenstein, Isabelle, Tim Rocktäschel, Andreas Vlachos, and Kalina Bontcheva. "Stance detection with bidirectional conditional encoding." arXiv preprint arXiv:1606.05464 (2016).

[40] Wolpert, David H. "Stacked generalization." Neural networks5, no. 2 (1992): 241-259.

[41] Chawla, Nitesh V., Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. "SMOTE: synthetic minority over-sampling technique." Journal of artificial intelligence research16 (2002): 321-357.

[42] Lin, Tsung-Yi, Priyal Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. "Focal loss for dense object detection." IEEE transactions on pattern analysis and machine intelligence(2018).

[43] Wikipedia contributors, "Overfitting," Wikipedia, The Free Encyclopedia, https://en.wikipedia.org/w/index.php?title=Overfitting&oldid=870581179 (accessed December 3, 2018).

[44] Hinton, Geoffrey E.; Srivastava, Nitish; Krizhevsky, Alex; Sutskever, Ilya; Salakhutdinov, Ruslan R. (2012). "Improving neural networks by preventing

co-adaptation of feature detectors".

[45] Baird, Sean, Doug Sibley, and Yuxi Pan. "Talos targets disinformation with fake news challenge victory." (2017).

[46] Hanselowski, Andreas, P. V. S. Avinesh, Benjamin Schiller, and Felix Caspelherr. Description of the system developed by team athene in the fnc-1. Technical report, 2017.

[47] Riedel, Benjamin, Isabelle Augenstein, Georgios P. Spithourakis, and Sebastian Riedel. "A simple but tough-to-beat baseline for the Fake News Challenge stance detection task." arXiv preprint arXiv:1707.03264 (2017).

[48] Pedregosa, Fabian, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel et al. "Scikit-learn: Machine learning in Python." Journal of machine learning research 12, no. Oct (2011): 2825-2830.

[49] Paszke, Adam, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. "Automatic differentiation in pytorch." (2017).

[50] Bird, Steven, Ewan Klein, and Edward Loper. Natural language processing with Python: analyzing text with the natural language toolkit. " O'Reilly Media, Inc.", 2009.

[51] Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980(2014).