# Gradient-Based Adversarial Training on Transformer Networks for Detecting Check-Worthy Factual Claims

**Anonymous COLING submission**

## Abstract

We present a study on the efficacy of adversarial training on transformer neural network models on the task of detecting check-worthy claims. In this work, we introduce the first adversarially-regularized, transformer-based claim spotter model that achieves state-of-the-art results on multiple challenging benchmarks. We obtain a $4.70$ point F1-score improvement over current state-of-the-art models on the ClaimBuster Dataset. In the process, we propose a method to apply adversarial training to transformer models, which has the potential to be generalized to many similar text classification tasks. Along with our results, we are releasing our codebase, and we also showcase our models' real-world usage via a live public API: *URL redacted for double-blind review.*

## 1 Introduction

The creation and propagation of misinformation has become an increasingly important issue for our society to tackle. Today, many falsehoods are spread via mediums that allow quick dissemination of information, including social media, news outlets, and televised programs. The distribution of objectively incorrect information can negatively impact the operation of our society in many spheres. Especially in the realm of political discourse, misinformation can shake public confidence in government institutions, [1] erroneously inform political judgements (Allcott and Gentzkow, 2017), and reinforce confidence in wrong information (Chan et al., 2017).

Although the number of fact-checking outlets has grown from 44 in 2014 to 226 in 2019, [2] many problematic claims still go unnoticed and unchecked due to the intense time commitment demanded by fact-checking. This problem is then exacerbated by the rapid rate at which new content surfaces via modern media channels (Pennycook and Rand, 2019). These challenges present an opportunity for *automated* fact-checking tools to help fact-checkers perform their duties more efficiently.

Claim-spotting is a crucial process through which relevant and factual claims are spotted from large streams of information on sources including newscasts, Twitter, and Facebook. This is paramount to ensuring that 1) check-worthy factual claims are not missed by fact-checkers and 2) unimportant or non-factual claims do not congest fact-checkers' intellectual bandwidth. Claim-spotting is an area that is highly suitable for machine learning to tackle.

Currently, no existing claim-spotter (Hassan et al., 2017a; Hassan et al., 2015; Jimenez and Li, 2018; Hansen et al., 2019; Favano et al., 2019) has applied transformers (Vaswani et al., 2017) to the claim-spotting task. The transformer is a new deep learning architecture that has recently yielded rapid progress in the natural language processing field. Particularly, Bidirectional Encoding Representations from Transformers (BERT) (Devlin et al., 2019) has achieved state-of-the-art performance on many challenging language understanding and classification benchmarks. However, BERT models have upwards of 300 million trainable parameters, making them susceptible to overfitting (Sun et al., 2019) when trained on limited amounts of data. To address this, we propose to incorporate *gradient-based adversarial training* into a BERT-based model as a regularization technique. No prior work has attempted to incorporate this type of adversarial training into transformer networks.

[1] `https://pewrsr.ch/2HoH0au` [2] `https://reporterslab.org/tag/fact-checking-database/`

Motivated by the above, we introduce the first adversarially-regularized, transformer-based claim-spotting model that achieves state-of-the-art results on challenging claim-spotting benchmarks. Our contributions are summarized as follows:

- We are the first to apply gradient-based adversarial training to transformer networks.
- We present the first transformer-based neural network architecture for claim-spotting.
- Our models are the first claim-spotters to be regularized by gradient-based adversarial training.
- Our models achieve state-of-the-art performance by a substantial margin on public benchmarks.
- We release a public codebase, dataset, and API: *URL redacted for double-blind review*.

## 2 BERT Claim Spotting Model

In this section, we present our approach to integrating adversarial training into the BERT architecture for the claim spotting task. To the best of our knowledge, our work is the first to apply gradient-based adversarial training (Goodfellow et al., 2014) to transformer networks.

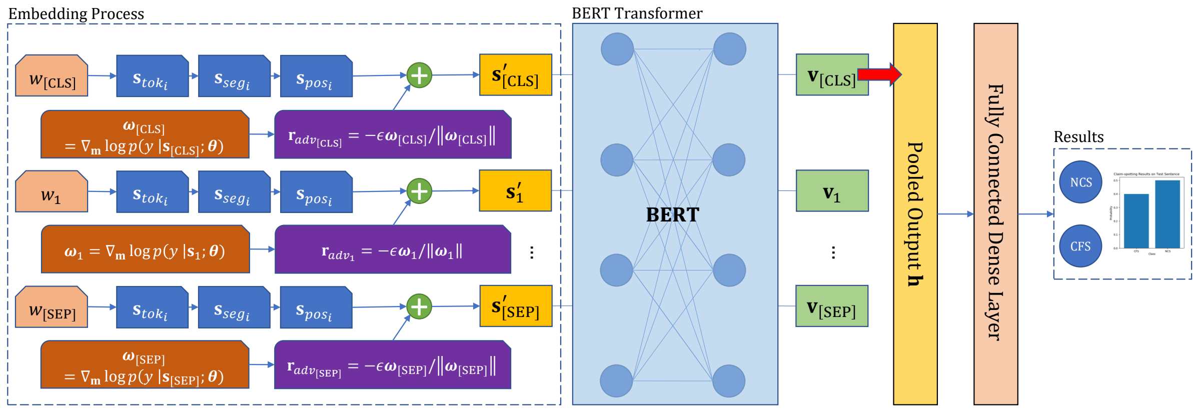

Embedding Process
BERT Transformer
$w_{[CLS]}$
$s_{tok_i}$
$s_{seg_i}$
$s_{pos_i}$
$s'_{[CLS]}$
$\omega_{[CLS]} = \nabla_m \log p(y \mid s_{[CLS]}; \theta)$
$r_{adv_{[CLS]}} = -\epsilon\omega_{[CLS]}/\|\omega_{[CLS]}\|$
$w_1$
$s'_1$
$\omega_1 = \nabla_m \log p(y \mid s_1; \theta)$
$r_{adv_1} = -\epsilon\omega_1/\|\omega_1\|$
$w_{[SEP]}$
$s'_{[SEP]}$
$\omega_{[SEP]} = \nabla_m \log p(y \mid s_{[SEP]}; \theta)$
$r_{adv_{[SEP]}} = -\epsilon\omega_{[SEP]}/\|\omega_{[SEP]}\|$
BERT
$v_{[CLS]}$
$v_1$
$v_{[SEP]}$
Pooled Output $\mathbf{h}$
Fully Connected Dense Layer
Results
NCS
CFS

Figure 1: Our Custom Adversarially Perturbed Claim Spotting Architecture

### 2.1 Preliminaries

#### 2.1.1 Task Definition

Detecting check-worthy factual claims has been studied as a binary/ternary classification task and a ranking task, as explained below. In this paper, we evaluate the performance of our models on the binary and ranking task definitions.

**Binary Classification Task**: In this work a sentence $\mathbf{w}$ is classified as one of two classes. This deviates from the previous definitions (Hassan et al., 2017b; Jimenez and Li, 2018; Hassan et al., 2017a; Hassan et al., 2015), but is consistent with the latest release of the ClaimBuster dataset. (Arslan et al., 2020b)

- **Non-Check-Worthy Sentence (NCS):** This class includes sentences that contain subjective or opinion-centered statements, questions, and trivial factual claims (e.g., The sky is blue).
- **Check-Worthy Factual Sentence (CFS):** This class contains claims that are both factual and salient to the general public. They may touch on statistics or historical information, among other things.

**Ranking Task**: To capture the importance of prioritizing the most check-worthy claims, a check-worthiness score (Hassan et al., 2017a) is defined for each sentence $\mathbf{w}$:

$$CWS = p(y = \text{CFS} \mid \mathbf{w}) \tag{1}$$

Where $CWS$ is the predicted probability that a given claim is in the CFS class.

#### 2.1.2 BERT Language Model

Bidirectional Encoder Representations from Transformers (BERT) (Devlin et al., 2019) is a transformer-based language modeling architecture that has recently achieved state-of-the-art performance on many language modeling and text classification tasks, including the Stanford Question Answering Dataset

(SQuAD) (Rajpurkar et al., 2016) and General Language Understanding Evaluation (GLUE) (Wang et al., 2018). We review BERT's relevant features below.

**Input/Output Representations:** Consider an arbitrary training sentence $\mathbf{w}$ with ground-truth label $y$. $\mathbf{w}$ is first tokenized using the WordPiece Tokenizer (Wu et al., 2016). Next, a [CLS] token is prepended to $\mathbf{w}$ to indicate the start of the sentence, a [SEP] token is appended to $\mathbf{w}$ to indicate the end of a sentence, and $\mathbf{w}$ is padded to a length of $T = 200$ using whitespace. Each resulting token is then converted into its corresponding index in the WordPiece vocabulary list. The resultant vector, denoted $\mathbf{x} \in \mathbb{R}^T$, is passed to the embedding layers.

**Three-Part Embeddings:** $\mathbf{x}$ is first transformed from a sparse bag-of-words form to a dense vector representation (Mikolov et al., 2013) through an embedding lookup matrix $\mathbf{V} \in \mathbb{R}^{Q \times H}$, where $Q$ is the size of the WordPiece vocabulary list and $H$ is the embedding dimensionality. The series of operations that applies $\mathbf{V}$ to $\mathbf{x}$ is called the *token* embedding layer, and its output is given as $\mathbf{s}_{tok} = \mathbf{V}_{x_t}, \forall x_t \in \mathbf{x}$, where $\mathbf{s}_{tok} \in \mathbb{R}^{T \times H}$. Additionally, BERT utilizes an *segment* embedding layer that signifies which parts of the input contain the input sentence, as the end of $\mathbf{x}$ may be padded with empty space. The output of this layer is denoted by $\mathbf{s}_{seg} \in \mathbb{R}^{T \times H}$. Finally, since vanilla transformers analyze all tokens in parallel and therefore cannot account for the sequential ordering of words, BERT introduces a randomly-initialized real-valued signal via the *positional* embedding layer to encode the relative order of words. The output of this layer is denoted $\mathbf{s}_{pos} \in \mathbb{R}^{T \times H}$. The final input, denoted $\mathbf{s}$, is the element-wise addition of the three separate embedding layers' outputs: $\mathbf{s} = \mathbf{s}_{tok} + \mathbf{s}_{seg} + \mathbf{s}_{pos}$. We denote the vector representation of the $t$th token in $\mathbf{x}$ to be $\mathbf{s}_t \in \mathbb{R}^H$.

**Transformer Encoder:** Using multiple stacked layers of attention heads (Vaswani et al., 2017), the BERT module encodes each input embedding $\mathbf{s}_t$ into a hidden vector $\mathbf{v}_t \in \mathbb{R}^H$, which is a hidden representation that incorporates context from surrounding words bidirectionally, as opposed to unidirectional encoders used in OpenAI GPT (Radford et al., 2018; Radford et al., 2019).

**Pooling Layer:** The pooling layer generates a representation for the entire sentence by applying a dense layer on top of the [CLS] token's hidden representation, resulting in $\mathbf{h} \in \mathbb{R}^H$. This sentence-level encoding vector can be used to perform many downstream tasks including claim-spotting.

## 2.2 Model Architecture

In this section, we outline how BERT is integrated with adversarial perturbations to create a claim-spotting model (Figure 1).

### 2.2.1 Embedding Process

All three types of embeddings are carried over from the BERT architecture. Each embedding layer still performs its original function, transforming a given word $\mathbf{x}$ into the embedding representation $\mathbf{s}$. The key difference in our architecture is the implantation of an addition gate through which adversarial perturbations $\mathbf{r}_{adv}$ are injected into $\mathbf{s}$ to create the perturbed embedding $\mathbf{s}'$.

### 2.2.2 BERT Transformer

Our work harnesses the power of the BERT architecture using transfer learning (Tan et al., 2018; Radford et al., 2018; Peters et al., 2018; Howard and Ruder, 2018), a process in which weights are loaded from a BERT language model that was pre-trained on billions of English tokens. Denote the number of transformer encoder layers as $L$, the hidden size as $H$, and the number of self-attention heads as $A$. The version of BERT used is $\text{BERT}_{\text{Base}}$ ($L = 12$, $A = 12$, $H = 768$), which has approximately 110-million parameters. Pretrained BERT model weights can be found on Google Research's Repository. [3] Base BERT was selected for superior speed during inference.

### 2.2.3 Fully-Connected Dense Layer

The dense layer is implemented as a fully-connected neural network that accepts input $\mathbf{h}$ and returns $|\mathbf{k}|$ un-normalized activations in $\mathbf{z} \in \mathbb{R}^{|\mathbf{k}|}$, where $\mathbf{k} = \{CFS, NCS\}$. $\mathbf{z}$ is passed through the softmax normalization function [4] to produce final output vector $\hat{\mathbf{y}}$, where each activation in $\hat{\mathbf{y}}$ represents a clas-

[3] `https://github.com/google-research/bert` [4] `en.wikipedia.org/wiki/Softmax_Function`

sification class. $\hat{\mathbf{y}}$ will be used to compute the check-worthiness score $CWS$ (Equation 1) and compute the predicted classification label as $\hat{y} = \operatorname{argmax} \hat{\mathbf{y}}$.

### 2.3 Standard Optimization Objective Function

In neural networks, an objective function, also known as the cost or loss function, is a differentiable expression that serves two purposes: to 1) quantify the disparity between the predicted and ground-truth probability distributions and 2) provide a function for gradient descent to minimize. Negative log-likelihood is a highly common choice for the cost function, because it has a nicely computable derivative for optimization via backpropagation, among other advantageous properties (Janocha and Czarnecki, 2017). Our standard negative log-likelihood loss function is formulated as the probability that the model predicts ground-truth $y$ given embedded inputs $\mathrm{s}$, parameterized by the model's weights $\boldsymbol{\theta}$:

$$\mathcal{L}_{reg} = -\frac{1}{N} \sum_{n=1}^{N} \log p(y^{(n)} \mid \mathrm{s}^{(n)}; \boldsymbol{\theta}) \tag{2}$$

where $N$ is the total number of training examples in a dataset. $\mathcal{L}_{reg}$ is used to compute adversarial perturbations in Section 2.4.

### 2.4 Computing Adversarial Perturbations

Gradient-based adversarial training is a regularization technique first introduced in (Goodfellow et al., 2014) that trains classifiers to be resistant to small perturbations to its inputs. Rather than passing regular embedded input $\mathrm{s}$ into a processing module such as a transformer or LSTM, adversarial training inputs $\mathrm{s}' = \mathrm{s} + \mathbf{r}_{adv}$. $\mathbf{r}_{adv}$ is typically a norm-constrained vector that modifies the input slightly to force the classifier to output incorrect predictions. Then, the disparity between the ground-truth ($y$) and perturbed prediction ($\hat{y}'$) is minimized through backpropagation, hence training the model to be resistant to adversarial perturbations. We are particularly interested in adversarial training's potential as a regularization technique (Shafahi et al., 2019; Dalvi et al., 2004; Nguyen et al., 2015; Shaham et al., 2018; Goodfellow et al., 2014; Miyato et al., 2016), as BERT networks are prone to overfitting when being fine-tuned on small datasets (Sun et al., 2019). To the best of our knowledge, **we contribute the first implementation of this technique on transformer networks**.

#### 2.4.1 Formal Definition of Adversarial Perturbations

We denote $\boldsymbol{\theta}$ as the parameterization of our neural network and $\mathbf{r}_{adv}$ as a vector perturbation that is added element-wise to $\mathrm{s}$ before it is passed to the transformer encoder. $\mathbf{r}_{adv}$ can be computed in several ways. Firstly, random noise may be added to disrupt the classifier. This is typically formalized as sampling $\mathbf{r}_{adv}$ from a Gaussian distribution as $\mathbf{r}_{adv} \sim \mathcal{N}(\mu, \sigma^2)$. Alternatively, we can compute perturbations that are *adversarial*, meaning that they increase the model's negative-log-likelihood error (Equation 2) by the theoretical maximum margin. This achieves the desired effect of generating a perturbation in the direction in which the model is most sensitive. In this case, $\mathbf{r}_{adv}$ is given as:

$$\mathbf{r}_{adv} = \underset{\mathbf{r}, ||\mathbf{r}|| \leq \epsilon}{\operatorname{argmax}} \; -\log p(y \mid \mathbf{s} + \mathbf{r}; \boldsymbol{\theta}) \tag{3}$$

where $\epsilon$ is a constraint that limits the magnitude of the perturbation.

In (Miyato et al., 2016), it was shown that random noise is a weaker regularizer than adversarially-computed perturbations. Therefore, we adopt adversarial perturbations for our model (Equation 3) and propose to apply them on the embeddings of the BERT model.

#### 2.4.2 Making Perturbations Computationally Tractable via Approximation

Equation 3 gives the *absolute* worst-case adversarial perturbation $\mathbf{r}_{adv}$ given a constraint that $||\mathbf{r}|| \leq \epsilon$. However, the computation of this value is intractable in neural networks; functions such as Equation 2 are nearly always non-convex. To this end, we propose a novel technique for generating *approximately* worst-case perturbations to the BERT model.

Because BERT embeddings are composed of multiple components (Section 2.1.2), it may not be optimal from a regularization standpoint to compute perturbations w.r.t. $\mathbf{s}$. Therefore, to determine the optimal perturbation setting, we propose to experiment with computing $\mathbf{r}_{adv}$ w.r.t. *all* possible combinations of the 3 embedding components. There are 7 different possible configurations in the set of perturbable combinations $\mathcal{P}$, letting $\mathcal{S}$ denote the set of embedding layers (Equation 4).

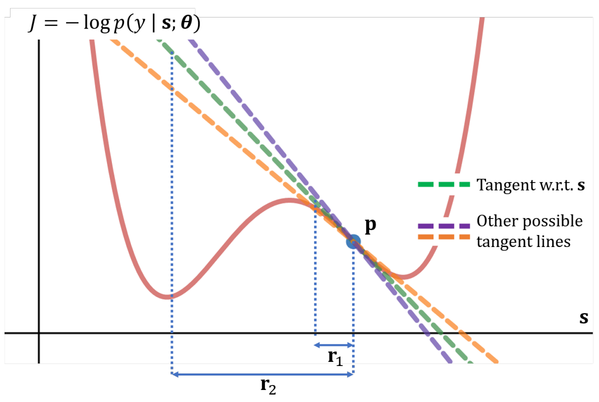
$$\mathcal{P} = 2^{\mathcal{S}} - \emptyset \text{ where } \mathcal{S} = \{\mathbf{s}_{tok}, \mathbf{s}_{seg}, \mathbf{s}_{pos}\} \qquad \Delta \mathbf{s} \propto \frac{\partial}{\partial \mathbf{m}} \log p(y \mid \mathbf{s}; \boldsymbol{\theta}) \tag{4, 5}$$

Given this list of components that can be perturbed, denote sum of the subset of the embeddings we will perturb as $\mathbf{m} = \sum_{x \in \mathbf{b}} x$ where $\mathbf{b} \in \mathcal{P}$. We then generate *approximate* worst-case perturbations by linearizing $\log p(y \mid \mathbf{s}; \boldsymbol{\theta})$ with respect to $\mathbf{m}$. To understand what this means, consider the simplified example shown in Figure 2, which graphs an example cost function $J = -\log p(y \mid \mathbf{s}; \boldsymbol{\theta})$ with respect to an example embedding space $\mathbf{s}$. For ease of visualization, in Figure 2 it is assumed that $\mathbf{s}$ exists on a scalar embedding space; in reality, our embeddings are high-dimensional. The gradient at the point $\mathbf{p}$ gives us information regarding which direction $\mathbf{s}$ should be moved to increase the value of the cost function, as seen in Equation 5.

However, we must be careful in determining how much $\mathbf{s}$ should be perturbed, because the assumption that $J$ is linear may not hold in reality. If the perturbation is too large ($\mathbf{r}_2$ in Figure 2), the adversarial effect will not be achieved, as the value of $J$ will in fact decrease. However, if we introduce a norm constraint $\epsilon$ to limit the perturbations to a reasonable size, linearization can accomplish the task of approximating a worst-case perturbation, as shown with $\mathbf{r}_1$ in Figure 2.

$$\mathbf{r}_{adv} = -\epsilon \boldsymbol{\omega} / \|\boldsymbol{\omega}\|_2 \quad \text{where } \boldsymbol{\omega} = \nabla_{\mathbf{m}} - \log p(y \mid \mathbf{s}; \boldsymbol{\theta}) \qquad \mathbf{s}' = \mathbf{s} + \mathbf{r}_{adv} = \mathbf{s}_{tok} + \mathbf{s}_{seg} + \mathbf{s}_{pos} + \mathbf{r}_{adv} \tag{6, 7}$$

Given the above insight, we generalize the one-dimensional example (Equation 5) to higher dimensions using the gradient vector. Therefore, the adversarial perturbation $\mathbf{r}_{adv}$ is computed with Equation 6, which can be implemented efficiently using the backpropagation algorithm. Since we desire to train our language classification model to become resistant to the perturbations defined in Equation 6, we add $\mathbf{r}_{adv}$ to $\mathbf{s}$ to create $\mathbf{s}'$ (Equation 7). After $\mathbf{s}'$ is passed into the transformer module, predictions will be generated. These predictions will be used in formulating the adversarial optimization objective function (Section 2.5).



$J = -\log p(y \mid \mathbf{s}; \boldsymbol{\theta})$
Tangent w.r.t. s
Other possible tangent lines
p
s
r1
r2

Figure 2: Visualization of Linearization

## 2.5 Compound Optimization Objective

Our model's final optimization objective contains two components: standard loss and adversarial loss. *Standard loss* was defined in Equation 2. *Adversarial loss* optimizes for distributional smoothness, given by the negative log-likelihood of a model parameterized by $\boldsymbol{\theta}$ predicting $y$ (Equation 8).

$$\mathcal{L}_{adv} = -\frac{1}{N} \sum_{n=1}^{N} \log p(y^{(n)} \mid \mathbf{s}^{(n)\prime}; \boldsymbol{\theta}) \quad \text{where } N = \text{ number of training samples } \in \mathcal{D}. \qquad \min_{\boldsymbol{\theta}} \left\{ \mathcal{L}_{reg} + \lambda \mathcal{L}_{adv} \right\} \quad \text{where } \lambda \text{ is a balancing factor between standard and adversarial loss.} \tag{8, 9}$$

The final optimization objective is given as the sum of $\mathcal{L}_{reg}$ and $\mathcal{L}_{adv}$. By combining the two losses, gradient descent will optimize for *both* distributional smoothness and model accuracy (Equation 9).

### 2.6 Adversarial Training Algorithm

```
Algorithm 1: Adversarial Training Loop
  Input: Training data $\mathcal{D}$
  Initialize $\boldsymbol{\theta}_{fc}$ and set as trainable;
  Load and freeze $\boldsymbol{\theta}_{tok}$, $\boldsymbol{\theta}_{seg}$, $\boldsymbol{\theta}_{pos}$, $\boldsymbol{\theta}_{n}, \forall n$;
  Set $\boldsymbol{\theta}_{n}, \forall F \leq n < L$ to trainable;
  $\mathcal{M} \leftarrow$ claim-spotting model (Figure 1);
  while not converge do
      Sample $\mathbf{w}, y$ from data $\mathcal{D}$;
      Pre-process $\mathbf{w}$ into $\mathbf{s}$;
      ▷ Standard forward-propagation
      $\hat{y} \leftarrow \mathcal{M}(\mathbf{s}; \boldsymbol{\theta})$;
      Compute $\mathcal{L}_{reg}$ using $y, \hat{y}$ (Equation 2);
      ▷ Generate and apply perturbations
      Compute $\mathbf{r}_{adv}$ using $\mathcal{L}_{reg}, y$ (Equation 6);
      Compute perturbed input (Equation 7);
      ▷ Adversarial forward-propagation
      $\hat{y}' \leftarrow \mathcal{M}(\mathbf{s}'; \boldsymbol{\theta})$;
      Compute $\mathcal{L}_{adv}$ using $y, \hat{y}'$ (Equation 8);
      ▷ Adversarial training
      Optimize $\{\mathcal{L}_{reg} + \lambda\mathcal{L}_{adv}\}$ (Equation 9);
```

Let $\boldsymbol{\theta}_{tok}$ be the parameters of the token embedding lookup table, $\boldsymbol{\theta}_{seg}$ be the parameters of the segment embedding layer, and $\boldsymbol{\theta}_{pos}$ be the parameters of the positional embedding layer, $\boldsymbol{\theta}_{n}, \forall n$ be the parameters for each of the $L$ transformer encoder layers (zero-indexed), $\boldsymbol{\theta}_{pool}$ be the parameters of the pooling layer, and $\boldsymbol{\theta}_{fc}$ be the parameters in the fully-connected dense layer. We also define $F$ as the number of encoder layers to freeze (i.e. render the weights uneditable during backpropagation), where $0 \leq F \leq L$.

The adversarial training procedure is shown in Algorithm 1. First, model $\mathcal{M}$ is used to compute the optimization function $\mathcal{L}_{reg}$. Then, $\mathcal{L}_{reg}$ is used to compute the adversarial perturbation (Equation 6), which is used to compute the *adversarial* optimization objective (Equation 8). This objective is added to the standard objective (Equation 9) and minimized using gradient descent. In practice, while constructing the computational graph, we stop the flow of gradients when $\mathbf{r}_{adv}$ is generated to prevent backpropagating through $\mathcal{M}$ twice in each training loop.

## 3 Results and Discussions

We evaluate our new transformer-based claim-spotting models on the Classification and Ranking Tasks (Section 2.1.1) and against two groups of models: (1) the ClaimBuster models (Hassan et al., 2017a; Jimenez and Li, 2018) and (2) the top-two performing systems from the 2019 CLEF-CheckThat! Challenge. Table 1 shows two example sentences, their ground-truth labels, and the $CWS$ score for the SVM and adversarially-trained BERT models. We re-iterate these scores do not signify whether a statement is true or false, but rather whether it is worth checking or not.

Table 1: Sample Sentences, Labels, and Check-Worthiness Scores

| Claim | Label | SVM CWS Score | BBA CWS Score |
|---|---|---|---|
| The U.S. loses millions of lives each year to homicide. | CFS | 0.7381 | 0.9999 |
| I really think you're overthinking the situation. | NCS | 0.6829 | $1.44 \times 10^{-4}$ |

### 3.1 Experiment Setup

#### 3.1.1 Datasets

We use two claim-spotting datasets to evaluate model performance.

**ClaimBuster Dataset** (CBD): The ClaimBuster dataset (Arslan et al., 2020a) is a manually labeled dataset publicly available on zenodo (Arslan et al., 2020b). The most recently released CBD consists of two classes, as mentioned in Section 2.1.1: NCS and CFS. The CBD consists of 9674 sentences (6910 NCS and 2764 CFS). We perform 4-fold cross validation using this same dataset. The dataset is composed of manually labeled sentences from all U.S. presidential debates from 1960 to 2016.

**CLEF2019-CheckThat! Dataset** ($C_{2019}$): We also evaluate our model on the 2019 CLEF-CheckThat! [5]

[5] `https://sites.google.com/view/clef2019-checkthat/`

claim-spotting dataset. Sentences are labelled as check-worthy *only* if they were fact-checked by FactCheck.org. Note that we believe this labelling strategy introduces significant bias into the dataset, as many problematic claims go unchecked due to the limited resources of a single organization's fact-checkers (Section 1). The training set contains 15,981 NCS and 440 CFS samples, and the testing set contains 6,943 NCS and 136 CFS sentences. The $C_{2019}$ dataset also includes speaker information, which we disregard for 3 primary reasons: (1) even if names are masked with guid, unwanted bias may be introduced, (2) speakers not present in the training set are likely to be encountered in-the-wild, and (3) live transcripts typically lack speaker information.

## 3.2 Evaluated Models

**BBA**: This model is trained using our novel claim-spotting framework detailed in Section 2.2. It is trained adversarially using the compound optimization objective defined in Equation 9.

**BB**: This model is architecturally identical to BBA but is trained using the standard optimization objective (Equation 2). In implementation, $\mathbf{r}_{adv}$ is set to 0. This model serves as a comparison to BBA.

**BiL**: (Jimenez and Li, 2018) This model is a re-implementation of (Jimenez and Li, 2018). It uses normalized GloVe word embeddings [6] and a Bidirectional LSTM network trained with RMSProp.

**SVM**: (Hassan et al., 2015; Hassan et al., 2017a) The SVM classifier is a re-implementation of (Hassan et al., 2017a) and uses a linear kernel. The feature vector used to represent each sentence is composed of a tf-idf weighted bag-of-unigrams vector, part-of-speech vector, and sentence length (i.e., number of words). The total number of features for each sentence using the CBD dataset is 6980.

### 3.2.1 2019 CLEF-CheckThat! Models

Neither of the top two teams in CLEF2019 released their code; therefore, we were only able to retrieve their results on the $C_{2019}$ dataset.

**Copenhagen** (Hansen et al., 2019): Team Copenhagen's model, the top performer on $C_{2019}$, consisted of an LSTM model (Hochreiter and Schmidhuber, 1997) using token embeddings fused with syntactic-dependency embeddings at the input layer. To train their model, Copenhagen *did not* use the $C_{2019}$ dataset, instead using an external dataset of Clinton/Trump debates that was weakly labeled using ClaimBuster's API. [7]

**TheEarthIsFlat** (Favano et al., 2019): TheEarthIsFlat, the second-place performer, used a feed-forward neural network trained on the $C_{2019}$ dataset. They processed sentences using the Universal Sentence Encoder (Cer et al., 2018).

## 3.3 Embedding Perturbation Study Results

In Table 2, we provide the results of perturbing the 3 different embedding layers in BERT, as proposed in Section 2.4. From the results we conclude that setting 6 produces the best models for our task. Particularly, this setting produces the best recall for the CFS class, which is arguably the more important class. The sacrifice in recall, with respect to the NCS class, compared to other settings is only $\approx 0.02$ at most. While setting 4 achieves the best performance in precision with respect to the CFS class, the drop in recall for the CFS class makes it not viable. Thus, from here on, any results dealing with adversarial training will employ setting 6 and perturb only the *tok* embedding layer, which contains the contextual word embeddings of each token.

Table 2: Perturbation Combinations Study Results Averaged Across Stratified 4-Fold Cross Validation

| **ID** | **P** | | **R** | | **F1** | |
|---|---|---|---|---|---|---|
| | **NCS** | **CFS** | **NCS** | **CFS** | **NCS** | **CFS** |
| **0** | 0.9225 | 0.8585 | 0.9472 | 0.8010 | 0.9347 | 0.8287 |
| **1** | 0.9227 | 0.8453 | 0.9412 | 0.8028 | **0.9319** | **0.8235** |
| **2** | 0.9330 | **0.8382** | **0.9357** | 0.8321 | 0.9344 | 0.8351 |
| **3** | 0.9295 | 0.8424 | 0.9385 | 0.8220 | 0.9340 | 0.8321 |
| **4** | **0.9201** | **0.8641** | **0.9501** | **0.7938** | 0.9349 | 0.8275 |
| **5** | 0.9282 | 0.8547 | 0.9444 | 0.8173 | **0.9362** | 0.8356 |
| **6** | **0.9335** | 0.8445 | 0.9386 | **0.8329** | 0.9361 | **0.8386** |
| **0** | **1** | **2** | **3** | **4** | **5** | **6** |
| pos seg tok | pos seg | pos tok | seg tok | pos | seg | tok |

[6] https://stanford.io/2YArce8 [7] https://idir.uta.edu/claimbuster/api/docs/

## 3.4 Performance Evaluation Results

### 3.4.1 Classification Results

Table 3: Precision, Recall, and F1 Averaged Across Stratified 4-Fold Cross Validation

| Model | P | | $P_m$ | $P_w$ | R | | $R_m$ | $R_w$ | F1 | | $F1_m$ | $F1_w$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | NCS | CFS | | | NCS | CFS | | | NCS | CFS | | |
| SVM | **0.8935** | 0.7972 | 0.8454 | **0.8660** | 0.9263 | **0.7240** | **0.8251** | **0.8685** | 0.9096 | **0.7588** | **0.8342** | **0.8665** |
| BiL | 0.9067 | **0.7773** | **0.8420** | 0.8697 | **0.9123** | 0.7652 | 0.8387 | 0.8703 | **0.9095** | 0.7712 | 0.8403 | 0.8700 |
| BB | **0.9344** | 0.8149 | 0.8747 | 0.9003 | 0.9239 | **0.8379** | 0.8809 | 0.8993 | 0.9291 | 0.8263 | 0.8777 | 0.8997 |
| BBA | 0.9335 | **0.8445** | **0.8890** | **0.9081** | **0.9386** | 0.8329 | **0.8857** | **0.9084** | **0.9361** | **0.8386** | **0.8873** | **0.9082** |

Our results are encapsulated in Table 3 and Table 5. The metrics used are standard precision, recall, and F1 scores with weighted and macro values as defined in the scikit-learn library. [8] In Table 3, we observe that the SVM model has the lowest performance across many measures. This is expected, as the SVM can only capture the information present in the dataset, while the deep learning models benefit from outside knowledge afforded to them by either pre-trained word-embeddings or a pre-trained model (i.e., BERT) that can be fine tuned. The BiL model shows modest improvements overall, but it does achieve noticeably better CFS recall than the SVM model. With respect to BERT-based architectures, both models outperform SVM and BiL considerably. Between BB and BBA, BBA achieves the best performance across most of the metrics. Although BB has a better CFS recall and NFS precision, these differences are minuscule and not enough for BB to be considered a better model. Ultimately, BBA achieves a **4.70 point F1 score improvement over the past state-of-the-art** BiL model, a 5.31 point F1-score improvement over the SVM model, and a 0.96 point F1-score improvement over a regularly-trained BERT model. This demonstrates the effectiveness of our new architecture and training algorithm.

Table 4: CLEF-2019 Test Dataset Classification and Ranking Task Results

| Model | Training Dataset | mAP | P@10 | P@20 | P@50 | P | | R | | F1 | | nDCG |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | NCS | CFS | NCS | CFS | NCS | CFS | |
| Copenhagen | $C_{2019}$ | **0.1660** | 0.2286 | 0.1571 | 0.1143 | ~~####~~ | ~~####~~ | ~~####~~ | ~~####~~ | ~~####~~ | ~~####~~ | ~~####~~ |
| TheEarthIsFlat | $C_{2019}$ | 0.1597 | 0.2143 | 0.1857 | **0.1457** | ~~####~~ | ~~####~~ | ~~####~~ | ~~####~~ | ~~####~~ | ~~####~~ | ~~####~~ |
| SVM | $C_{2019}$ | **0.1087** | 0.1429 | 0.1429 | 0.1114 | 0.9813 | 0.2105 | 0.9978 | 0.0294 | 0.9895 | 0.0516 | **0.4567** |
| BBA | $C_{2019}$ | 0.1625 | **0.2571** | **0.1929** | **0.1086** | **0.9812** | **0.5000** | **0.9996** | **0.0221** | **0.9903** | **0.0423** | **0.5181** |
| SVM | CBD | 0.1134 | 0.1571 | 0.1429 | 0.1143 | 0.9885 | **0.0678** | 0.8694 | 0.4853 | 0.9251 | 0.1190 | 0.4744 |
| BBA | CBD | 0.1363 | **0.1286** | **0.1286** | 0.1229 | **0.9926** | 0.0752 | **0.8354** | **0.6838** | **0.9073** | **0.1356** | 0.4978 |

The results on the $C_{2019}$ dataset are in Table 4. The metrics presented for the CLEF competition teams are taken from (Atanasova et al., 2019), since we could not access the source code to reproduce them. For this reason we also cannot provide the P, R, F1, and nDCG for these teams. We tested models trained on both the CBD and $C_{2019}$ training set and used the $C_{2019}$ testing set to evaluate them. The models trained on CBD and tested on the CLEF test set didn't perform as well; this was expected, given that our methodology of dataset labelling differs significantly from CLEF's. Despite this, when trained on $C_{2019}$, BBA held its own when compared to Copenhagen, achieving significantly better performance on P@10 and P@20 and only falling short on mAP by 0.0035.

### 3.4.2 nDCG Results

Table 5: nDCG Averaged Across Stratified 4-Fold Cross Validation

| | SVM | BiL | BB | BBA |
|---|---|---|---|---|
| **nDCG** | **0.9765** | 0.9817 | 0.9877 | **0.9894** |

In Table 5 we observe that the best nDCG score is achieved by the BBA model, and the BB and BiL models are within $\approx \frac{6}{1000}$ of it. The SVM model has the "worst" nDCG, but is still not far behind the deep learning models. It is noteworthy that all models show relatively good performance on this measure since the CFS class is less represented in the dataset.

### 3.4.3 Distribution of $CWS$ Scores

[8] `https://bit.ly/2ZgtT3P`

We also analyze the distribution of the models' outputs on a typical corpus of text. We process 100 sentences from the January 14th, 2020 Democratic presidential debate [9], based on the same labeling criteria [10] as the ClaimBuster dataset. The sentences were chosen so that there would be about equal numbers of check-worthy and non-check-worthy sentences. Figure 3 displays the results, which use Kernel Density Estimation (Rosenblatt, 1956) to estimate the score distribution from discrete data points. Observing the density spikes around 0 and 1 on BBA's distribution, we conclude that our model more clearly differentiates sentences as check-worthy or not check-worthy. The more well-delineated distribution of the BBA model also improves its interpretability over SVM.

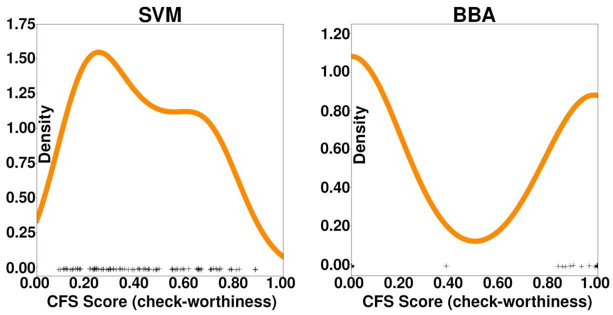


Figure 3: Check-Worthiness Score Distributions on BBA and SVM

## 4 Related Works

In recent years, there have been several efforts to build claim spotting models. ClaimBuster (Hassan et al., 2015) is the first of several notable claim spotting models. Another team (Gencheva et al., 2017) extended the SVM model's feature set (detailed in section 3.2) by including several contextual features such as: position of the target sentence in its segment, speaker name, interaction between opponents, etc. They created a dataset from the 2016 US presidential and vice presidential debates and annotated sentences by taking fact-checking outputs from 9 fact-checking organizations. If a sentence was fact-checked by at least one fact-checking outlet, it was labeled as check-worthy. A follow-up study (Jaradat et al., 2018) built an online system, namely, ClaimRank [11] for prioritizing sentences for fact-checking based on their check-worthiness score. ClaimRank is a re-implementation of the aforementioned study, but it also supports Arabic by employing cross-language English-Arabic embeddings. Another study (Patwari et al., 2017) followed the same data annotation strategy on a larger dataset by including sentences from an additional 15 2016 U.S. primary debates. The authors developed a multi-classifier based model called TATHYA that uses multiple SVMs trained on different clusters of the dataset. The model takes the output from the most confident classifier. The feature set used was comprised of tf-idf weighted bag-of-unigrams, topics of the sentences, part-of-speech tuples, and a count of entities.

Another effort by Konstantinovskiy et al. (Konstantinovskiy et al., 2018), which utilized the expertise of professional fact-checkers, designed an annotation schema and created a benchmark dataset for training a claim spotting model. The authors trained the model using logistic regression on top of dataset's universal sentence representation derived from InferSent (Conneau et al., 2017). Their model classifies a sentence as either checkable or non-checkable. The authors also disagreed with ClaimBuster's and ClaimRank's idea of a check-worthiness score. They believe the decision of how important a claim is, should be left to the professional fact-checkers.

## 5 Conclusion

We have presented our work on detecting check-worthy factual claims employing adversarial training on transformer networks. In the future, we are interested in exploring adversarial training as a *defense against malicious adversaries*. As a publicly deployed model, our work may be susceptible to exploitation without incorporating mechanisms that improve its robustness, as shown by (Jin et al., 2019) We are also investigating bias in the dataset and any ways to eliminate it.

[9] `https://bit.ly/3bH4fL9`

[10] `https://idir.uta.edu/classifyfact_survey/`

[11] `https://claimrank.qcri.org/`

## References

Hunt Allcott and Matthew Gentzkow. 2017. Social media and fake news in the 2016 election. Working Paper 23089, National Bureau of Economic Research, January.

Fatma Arslan, Naeemul Hassan, Chengkai Li, and Mark Tremayne. 2020a. A benchmark dataset of check-worthy factual claims. *Proceedings of the International AAAI Conference on Web and Social Media*, 14(1):821–829, May.

Fatma Arslan, Naeemul Hassan, Chengkai Li, and Mark Tremayne. 2020b. ClaimBuster: A Benchmark Dataset of Check-worthy Factual Claims, January.

Pepa Atanasova, Preslav Nakov, Georgi Karadzhov, Mitra Mohtarami, and Giovanni Da San Martino. 2019. Overview of the clef-2019 checkthat! lab on automatic identification and verification of claims. task 1: Check-worthiness. In *CEUR Workshop Proceedings*.

Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, et al. 2018. Universal sentence encoder.

Man-pui Sally Chan, Christopher R. Jones, Kathleen Hall Jamieson, and Dolores Albarracín. 2017. Debunking: A meta-analysis of the psychological efficacy of messages countering misinformation. *Psychological Science*, 28(11):1531–1546, Sep.

Alexis Conneau, Douwe Kiela, Holger Schwenk, Loic Barrault, and Antoine Bordes. 2017. Supervised learning of universal sentence representations from natural language inference data.

Nilesh Dalvi, Pedro Domingos, Sumit Sanghai, Deepak Verma, et al. 2004. Adversarial classification. In *SIGKDD*, pages 99–108.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*, pages 4171–4186.

Luca Favano, Mark J. Carman, and Pier Luca Lanzi. 2019. Theearthisflat's submission to clef'19 checkthat! challenge. In *CEUR Workshop Proceedings*.

Pepa Gencheva, Preslav Nakov, Lluís Màrquez, Alberto Barrón-Cedeño, and Ivan Koychev. 2017. A context-aware approach for detecting worth-checking claims in political debates. In *RANLP*, pages 267–276.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and harnessing adversarial examples.

Casper Hansen, Christian Hansen, Jakob Grue Simonsen, and Christina Lioma. 2019. Neural Weakly Supervised Fact Check-Worthiness Detection with Contrastive Sampling-Based Ranking Loss. In *CEUR Workshop Proceedings*.

Naeemul Hassan, Chengkai Li, and Mark Tremayne. 2015. Detecting check-worthy factual claims in presidential debates. In *CIKM*, pages 1835–1838.

Naeemul Hassan, Fatma Arslan, Chengkai Li, and Mark Tremayne. 2017a. Toward automated fact-checking: Detecting check-worthy factual claims by claimbuster. In *SIGKDD*, pages 1803–1812.

Naeemul Hassan, Gensheng Zhang, Fatma Arslan, Josue Caraballo, and et al. 2017b. Claimbuster: The first-ever end-to-end fact-checking system. *PVLDB*, 10(12):1945–1948, August.

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

Jeremy Howard and Sebastian Ruder. 2018. Universal language model fine-tuning for text classification. In *ACL*.

Katarzyna Janocha and Wojciech Marian Czarnecki. 2017. On loss functions for deep neural networks in classification. *Schedae Informaticae*, 1/2016.

Israa Jaradat, Pepa Gencheva, Alberto Barrón-Cedeño, Lluís Màrquez, and Preslav Nakov. 2018. ClaimRank: Detecting check-worthy claims in Arabic and English. In *NAACL*, pages 26–30.

Damian Jimenez and Chengkai Li. 2018. An empirical study on identifying sentences with salient factual statements. In *IJCNN*, pages 1–8.

Di Jin, Zhijing Jin, Joey Tianyi Zhou, and Peter Szolovits. 2019. Is bert really robust? natural language attack on text classification and entailment. *arXiv preprint arXiv:1907.11932*.

Lev Konstantinovskiy, Oliver Price, Mevan Babakar, and Arkaitz Zubiaga. 2018. Towards automated factchecking: Developing an annotation schema and benchmark for consistent automated claim detection.

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

Takeru Miyato, Andrew M Dai, and Ian Goodfellow. 2016. Adversarial training methods for semi-supervised text classification.

Anh Nguyen, Jason Yosinski, and Jeff Clune. 2015. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *CVPR*.

Ayush Patwari, Dan Goldwasser, and Saurabh Bagchi. 2017. Tathya: A multi-classifier system for detecting check-worthy statements in political debates. In *CIKM*, pages 2259–2262.

Gordon Pennycook and David G Rand. 2019. Fighting misinformation on social media using crowdsourced judgments of news source quality. *Proceedings of the National Academy of Sciences*, 116(7):2521–2526.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *NAACL*.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training. *Pre-print*.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. In *EMNLP*.

Murray Rosenblatt. 1956. Remarks on some nonparametric estimates of a density function. *The Annals of Mathematical Statistics*, 27(3):832–837, 09.

Ali Shafahi, Mahyar Najibi, Amin Ghiasi, Zheng Xu, John Dickerson, Christoph Studer, Larry S. Davis, Gavin Taylor, and Tom Goldstein. 2019. Adversarial training for free!

Uri Shaham, Yutaro Yamada, and Sahand Negahban. 2018. Understanding adversarial training: Increasing local stability of supervised models through robust optimization. *Neurocomputing*, 307:195–204.

Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. 2019. How to fine-tune bert for text classification?

Chuanqi Tan, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang, and Chunfang Liu. 2018. A survey on deep transfer learning. *Lecture Notes in Computer Science*, pages 270–279.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, and et al. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation.