HARD LABEL BLACK BOX ATTACK ON TEXT CLASSIFICATION

By

SACHIN SAXENA

A thesis submitted to the

Graduate School—Camden

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Master of Science

Graduate Program in Computer Science

written under the direction of

Dr. Sunil Shende

and approved by

_____

Dr. Sunil Shende

_____

Dr. Suneeta Ramaswami

_____

Dr. Jean-Camille Birget

Camden, New Jersey

January 2021

ABSTRACT

Hard Label Black Box Attack on Text Classification

By SACHIN SAXENA

Thesis Director:

Dr. Sunil Shende

Machine learning has been proven to be susceptible to carefully crafted samples, known as adversarial examples. The generation of these adversarial examples helps to make the models more robust and gives us an insight into the underlying decision-making of these models. Over the years, researchers have successfully attacked image classifiers in both white and black-box settings. How- ever, these methods are not directly applicable to texts as text data is discrete. In recent years, research on crafting adversarial examples against textual applications has been on the rise. In this thesis work, we present a novel approach for hard-labe1 black-box attacks against Natural Language Processing (NLP) classifiers, where no model information is disclosed, and an attacker can only query the model to get the final decision of the classifier, without confidence scores of the classes involved. Such an attack scenario applies to real-world black-box models being used for security-sensitive applications such as sentiment analysis and toxic content detection.

The main contributions of the thesis work are as follows:

1. We propose a novel approach to formulate natural adversarial examples against text classifiers in the hard label black-box setting.

2. We test our attack algorithm on three state-of-the-art classification models over two popular

text classification datasets.

3. We improve upon the grammatical correctness of the generated adversarial examples.

4. We also decrease the memory requirement for the attack when compared to published attack systems involving word-level perturbations.

Acknowledgments

I would like to acknowledge everyone who played a role in my academic accomplishments. First of all, I thank Professor Sunil Shende for insightful discussions and constructive suggestions throughout my thesis work. I also thank him for the consistent guidance and motivation to keep me going throughout my thesis work.

Secondly, my committee members, each of whom has provided patient advice and guidance throughout the research work. Thank you all for the unwavering support.

Dedication

This work is dedicated to my parents, teachers, and friends who contributed to their respective capacities at different stages to make this work a success.

Table of Contents

List of Tables

List of Figures

Chapter 1

Machine Learning

1.1 Introduction

Artificial Intelligence is the ability of computers and computer-controlled devices to perform tasks commonly linked with human intelligence. It consists of cognitive abilities like learning and problem-solving. Machine Learning (ML) is a subset of Artificial Intelligence that enables computers to perform specific tasks without giving explicit instructions. It consists of algorithms and statistical models to enable the computer to automatically learn and improve from experience. ML algorithms have helped computers to do a wide variety of tasks from spam filtering in our email inboxes, movie recommendations on Netflix to self-driving cars.

Much of the development of machine learning has been in the past sixty years. In 1943, neurophysiologist Warren McCulloch and mathematician Walter Pitts co-wrote a paper on how human neurons might work. They illustrated the theory using electrical circuits, and therefore, the neural network was born. In the 1950s, computer scientists began using this idea in their work. In 1952, Arthur Samuel created a program that played checkers and was the first computer program that could learn as it ran. The year 1958 saw the creation of one of the earliest artificial neural networks, perceptron, which was used for pattern and shape recognition. In 1959, Bernard Widrow and Marcian Hoff of Stanford University built two neural networks which are the earliest examples of real-life applications of machine learning. The first one was ADELINE which could detect binary patterns. The second one was MADELINE which could remove echo from phone lines, and is still in use today.

Machine learning algorithms can be broadly classified into four types based on the approach used for learning: Supervised Learning, Unsupervised learning, Semi-supervised learning, and Reinforcement learning.

1.1.1 Supervised Learning

Supervised Learning uses a labeled dataset, set of inputs and outputs, to learn the underlying function. The tasks involved are mainly regression or classification.

- Regression techniques help predict, forecast, and develop the relation between quantitative data.
- Whereas, Classification helps to categorize data into different groups or classes.

The main algorithms for supervised learning include Support Vector Machine (SVM), Linear Regression, Logistic Regression, Neural Networks, Random Forest, Naive Bayes, and k-Nearest Neighbor(KNN). A classical example of Supervised learning is spam filtering, which uses a dataset consisting of emails labeled as spam/not spam and the ML algorithm learns to classify a given email as spam/not spam.

1.1.2 Unsupervised Learning

In unsupervised learning, the Machine learning algorithms use unlabeled data and discover underlying patterns in the data on their own. The prime examples of unsupervised learning are Clustering and Anomaly detection.

- Clustering involves the grouping of data points in a given population in such a way that the points in the same group or cluster are more similar to each other than to points in another cluster. Popular algorithms include k-means, hierarchical clustering, DBSCAN, etc.
- Anomaly detection is the identification of rare data points or observations in a given data set. Although anomaly detection can also be done by supervised learning algorithms, it is difficult to get labeled data of anomalous behavior; hence, unsupervised learning is much more widely used for the purpose. For example, flagging unusual credit card transactions to prevent fraud.

1.1.3 Semi-Supervised Learning

Unsupervised Learning uses a combination of labeled and unlabeled data. It usually consists of a very small amount of labeled data and a huge amount of unlabeled data. This conjunction of unlabeled data with some labeled data can leads to considerable improvement in learning accuracy. It

is widely used for speech analysis, since labeling of speech data is a difficult task, semi-supervised learning is widely used for the purpose. Other important applications include internet content classification and protein sequence classification.

1.1.4 Reinforcement Learning

Reinforcement learning aims to enable software agents to take actions in an environment to maximize cumulative reward. A bot playing a game to achieve high scores is a classical example of reinforcement learning. The designer sets the reward policy but does not give any hints or suggestions for how to solve the game. The model starts with totally random trials and consequently, learns sophisticated tactics to maximize the reward.

1.2 Neural Network

Neural networks are a subset of machine learning and are inspired by the biological neural networks which exist in the human brain. A human brain consists of a network of computing devices called the neurons which communicate among themselves to perform computation tasks Artificial Neural Networks (ANN) mimics this basic architecture by forming a directed graph where nodes denote the neurons and the edges represent the links between them. One of the most fundamental types of neural network architecture is the Feedforward Neural Network.

1.2.1 Feedforward Neural Networks

A Feedforward Neural Network can be represented as a directed acyclic graph $G = (V, E)$, where vertices V represent the neurons and edges E are the links between them. They can be arranged in the form of layers: Input, Hidden, and Output layers.

Given a layer l named V_l, let there be n_l nodes in the layer and V_{lm} represent the m^{th} node in l^{th} layer, where $m < n_l$. Each node in layer l is connected to all the nodes in layer $l-1$ and layer $l+1$, except in the input layer and the output layer, where they are fully connected to only the $(l+1)^{th}$ and $(l-1)^{th}$ layer, respectively. Further, nodes in the same layer do not share any edges. A single node or neuron is described in figure 1.1.

A node can be further represented by two mathematical operations: a weighted summation of

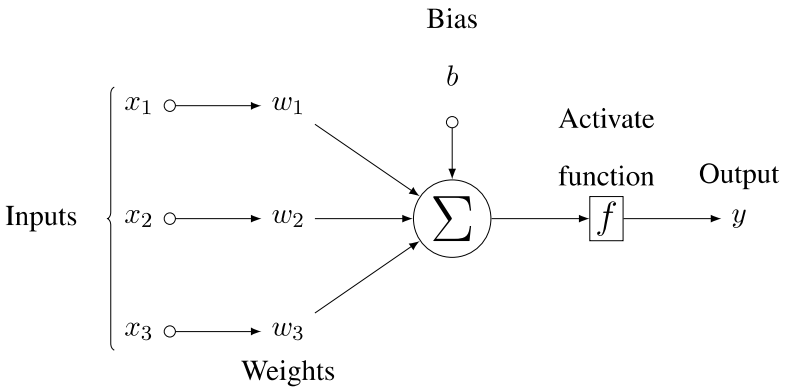


Figure 1.1: A single Node (Neuron)

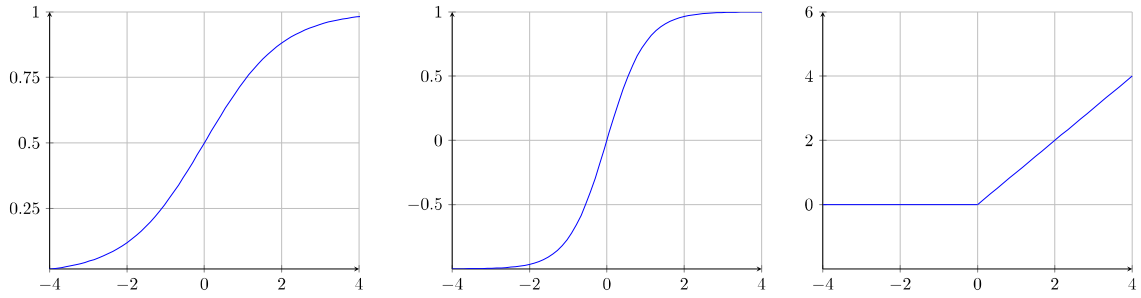
the inputs from the previous layer and an activation function f. An activation function is always a non-linear function and helps the neural network to represent complex functions mapping input to output.

1.2.2 Activation Functions

A neural network is expected to model complex functions and an activation function helps it achieve this. It introduces non-linearity in the model without which a neural network is essentially just a linear regression model. Activation functions have the following characteristics:

- Non-Linear: With the use of a non-linear activation function, a two-layer neural network can be a universal function approximator [11].
- Monotone: When the activation function is monotonic, the error surface linked to a single layer neural network is guaranteed to be convex, which is easier to optimize [44].
- Has a finite range
- Continuously differentiable
- Approximates identity near the origin

Commonly used activation functions are:



(a) Sigmoid (b) Tanh (c) ReLU

Figure 1.2: Common Activation Functions

- **Sigmoid Function** has a range of $[0, 1]$ and is mostly used in the output of a binary classification problem.

$$A(x) = 1/(1 + e^{-x})$$

- **Tan Hyperbolic (Tanh)** function is mainly used in the hidden layers and has a range of [-1,1].

$$A(x) = (e^{2x} - 1)/(e^{2x} + 1)$$

- **Rectified Linear Unit (ReLU)** is the simplest of the activation functions, yet widely used due to its advantages. It is less computationally expensive than both Sigmoid and Tanh functions because it involves simpler mathematical operations. Further, it solves the problem of vanishing gradients during backpropagation, since the derivative of a ReLU is either equal to 1 or 0, as compared to a value less than 1 for both, Sigmoid and Tanh activation functions.

$$A(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{otherwise} \end{cases}$$

- **Softmax** function is a generalization of a sigmoid function to multiple dimensions and is often used in the last layer of a neural network to normalize the output of a network to a probability distribution over output classes that the network predicts. Softmax function applied to a layer having j values rescales each of them as follows:

$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

1.2.3 Forward Propagation

Forward Propagation refers to the calculation and storage of values at all the nodes starting from the input layer to the output layer. Let the input be $X \in R^5$, a hidden layer V with 3 neurons and an output with a single neuron. The weight matrix linked with layer l is represented using $W^{(l)}$, biases are represented by $b^{(l)}$ and the activation function is $h^{(l)}$. Our layer indexing starts from 0. So, the computations in the first layer are as follows:

$$z^{(1)} = W^{(1)}X + b^{(1)}, W^{(1)} \in \mathbb{R}^{3\times5},\ X \in \mathbb{R}^{5\times1},\ b^{(1)} \in \mathbb{R}^{3\times1}$$

$$a^{(1)} = h^{(1)}(z^{(1)})$$

Computations in the second layer or the output layer happen as follows:

$$z^{(2)} = W^{(2)}a^{(1)} + b^{(2)},\ W^{(2)} \in \mathbb{R}^{1\times3},\ a^{(1)} \in \mathbb{R}^{3\times1},\ b^{(2)} \in \mathbb{R}^{1\times1}$$

$$a^{(2)} = h^{(2)}(z^{(2)})$$

The output of the model for one such forward propagation step is $a^{(2)}$

1.2.4 Loss Function

A loss function measures the performance of the model. There are different loss functions for classification and regression:

- **Cross Entropy Loss**

  Cross Entropy loss measures the performance of the classification model. If the predicted label or class probability is away from the actual label or class, cross entropy is higher and it is lower when the predicted label probability is close to the actual class. Mathematically, the cross entropy loss for a neural network designed for a multiclass classification task having C classes can be written as follows

  $$J(W, b) = \sum_{j=1}^{C} y_j \log P_j$$

  where P_j and y_j represents the predicted and actual probability of the class j.

- **Mean Squared Error (MSE)**

  Mean Squared Error is mostly used for regression tasks. Given m training examples, p_i and y_i representing the predicted and actual target value for the i^{th} training example, the mean squared error is as follows:

$$MSE = (1/m) \times \sum_{i=1}^{m} (p_i - y_i)^2$$

1.2.5 Gradient Descent

Neural network training is an optimization task, where the aim is to find the optimized values of weights W and biases b to minimize the loss function J. Gradient Descent is an optimization algorithm used to minimize the loss function by iteratively moving in the direction of the steepest descent. In a neural network, we use gradient descent to update the weights and biases. Mathematically, at any layer l, we update the weights and biases as follows:

$$W^{(l)} = W^{(l)} - \alpha \frac{dJ}{dW^{(l)}}$$

$$b^{(l)} = b^{(l)} - \alpha \frac{dJ}{db^{(l)}}$$

where α is the learning rate

1.2.6 Backpropagation Algorithm

Backpropagation is a method by which one is able to find the gradient of loss with respect to parameters in any layer l. For the single layer architecture shown in the earlier sections, we can find the gradient of the loss with respect to the weights in layer 1 as follows:

$$\frac{dJ}{dW^{(1)}} = \frac{dJ}{da^{(2)}} \times \frac{da^{(2)}}{dz^{(2)}} \times \frac{dz^{(2)}}{da^{(1)}} \times \frac{da^{(1)}}{dz^{(1)}} \times \frac{dz^{(1)}}{dW^{(1)}}$$

1.3 Convolutional Neural Network

Convolutional neural network (CNN) form a class of deep neural networks that are mostly applied for analyzing images. They have shown excellent performance in many computer vision, machine learning, and pattern recognition problems. In the past few years, CNN architectures have found applications in Natural Language processing tasks as well. CNN models have been shown to be

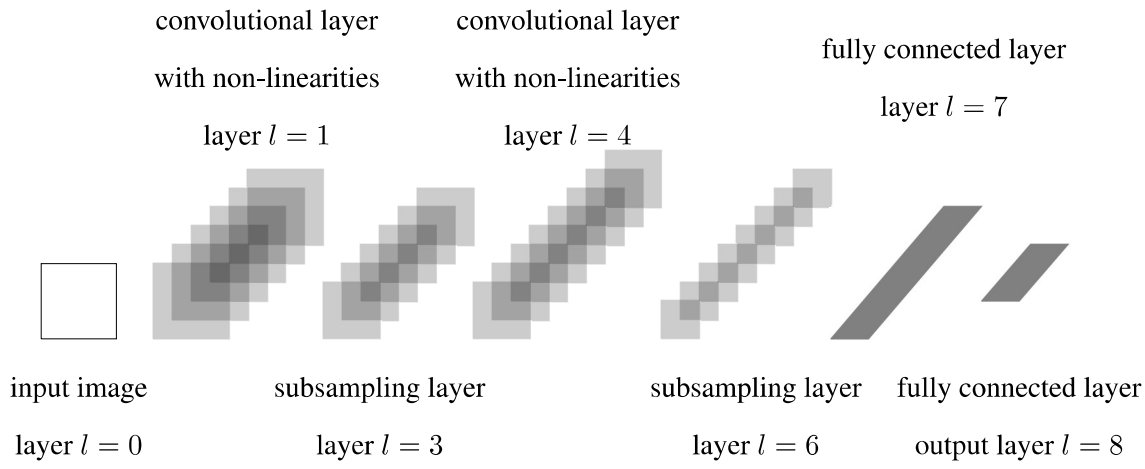


Figure 1.3: The architecture of the original convolutional neural network, as introduced by LeCun et al. (1998) [28]

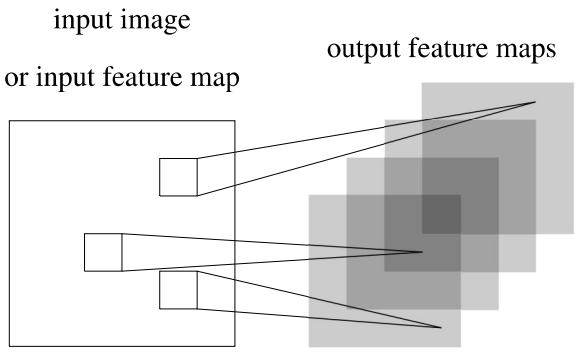


Figure 1.4: Illustration of a single convolutional layer. If layer l is a convolutional layer, the input image (if $l = 1$) or a feature map of the previous layer is convolved by different filters to yield the output feature maps of layer l.

effective for NLP and have achieved excellent results in semantic parsing [45], sentence modeling [23], search query retrieval [42], text classification [24] [46] and other traditional NLP tasks [9].

The architecture of the original convolutional neural network in Figure 1.3, as introduced by LeCun et al. (1998) [28], alternates between convolutional layers including hyperbolic tangent non-linearities and subsampling layers. Since the convolutional layers already include non-linearities, a convolutional layer represents two layers: one for convolution and the other for applying a non-linear activation function. The feature maps of the final sub-sampling layer are then fed into the actual classifier consisting of an arbitrary number of fully connected layers. The output layer usually uses softmax activation functions

1.3.1 Word-based Convolutional Neural Networks

Word-based CNN uses the word-level feature vectors to form the feature matrix and apply a series of convolution and pooling operations leading to the fully connected layers. Consider a sentence having n words and let $x_i \in \mathbb{R}^k$ be the k dimensional feature vector of the i^{th} word in the sentence. Consider a feature matrix $X \in \mathbb{R}^{n \times k}$ in which word vectors are along the rows. Let $X_{i:i+j}$ refer to the sub-matrix having rows $\{i, i+1...i+j-1\}$, representing vectors corresponding to the words $\{i, i+1...i+j-1\}$, in sequence.

Consider a matrix $f \in \mathbb{R}^{h \times k},\ 1 \leq h \leq n,\ h \in \mathbb{Z}$ which we refer to as a *filter*. The i^{th} $(1 \leq i \leq n-h+1,\ i \in \mathbb{Z})$ convolution operation is then defined as follows:

$$c_i = f \circledast X_{i:i+h}$$

Here $f \circledast X_{i:i+h}$ represents elementwise multiplication and summation of the corresponding values in the two matrices. We further add a bias term and non-linearity to the above value:

$$C_i = F(f \circledast X_{i:i+h} + b)$$

Here, F can be any non-linear activation function like Tan Hyperbolic, ReLU etc. This filter is applied to each possible window of words in the sentence, $\{X_{1:h}, X_{2:h+1}...X_{n-h+1:n}\}$ to obtain a feature map

$$C = [C_1, C_2...C_{n-h+1}]$$

A max-over-time pooling operation is then applied to this value map, to capture the most important feature:

$$\hat{C} = max\{C\}$$

Note that, the pooling operation inherently takes care of the varying word length sentences being input to the architecture, as one filter value only outputs a single max-pooled value from the feature map.

Although we demonstrated the convolution and pooling operation for one filter, we can apply multiple such filters to build a more complex architecture. If $h = 2$, we are capturing the features of the bi-grams (sequence of two adjacent words), by applying the filter and non-linearity to two words at a time. Similarly, by using filter of size $h = 3$ we can capture tri-grams (sequence of three

adjacent words) features. Finally, our output from the CNN layer will be a vector of length equal to the number of filters used. For a sentence classification task, these outputs can be passed onto a fully connected softmax layer, which can give the probability distribution over labels.

1.4 Sequence Models

1.4.1 Recurrent Neural Network

Recurrent Neural Networks are a type of supervised deep learning method used for time series and sequence data. RNNs have achieved state of the art performance on important tasks that include machine translation [4], language modeling [32] and speech recognition [17]. The difference between a simple feedforward neural network and RNN is that in the latter a neuron may pass its activation to the other neurons in the same layer, in addition to forwarding it to the next layer. These activations are stored as the internal states of the network and help an RNN to hold long-term temporal contextual information.

Although RNNs have been very useful for sequence data, they suffer from ***vanishing gradient problem*** which prevent them from learning long-term dependencies. The problem of the vanishing gradient was first discovered by Sepp Joseph Hochreiter back in 1991 [20]. RNNs have connections between different neurons across time, so the error term at any later time node in the sequence needs to be backpropagated through time to all the earlier neurons and the weight matrices need to be updated during gradient descent. RNNs mostly use Tan Hyperbolic activation function at every time step, the absolute value of the gradient of which always lies below 1. Hence, when the length of a sequence is large, gradients tend to get diminished as they are backpropagated to earlier time steps, leading to very slow learning at the earlier time steps during gradient descent. This means that RNNs are unable to capture long-term dependencies.

1.4.2 Long Short Term Memory (LSTM)

The LSTM units are engineered in such a way that allows them to bypass the vanishing gradient problem, thereby enabling them to learn long-term dependencies [21]. In an LSTM unit, the hidden state is calculated by four layers which helps it to remember or forget specific information about the preceding elements in the sequence.

Chapter 2

Adversarial Attacks

Machine learning has shown superiority over humans for tasks like image recognition, speech recognition, security-critical applications like a bot, malware, or spam detection. However. machine learning has been proven to be susceptible to carefully crafted adversarial examples. In recent years, research on the generation and development of defenses against such adversarial examples has been on the rise. Attack algorithms have been formulated for image classification problems by [6], [16].

A classic example of an adversarial attack is that of a self-driving car crashing into another car because it ignores the stop sign. The stop sign is an adversarial example that an adversary intentionally placed in the place of the original stop sign. Another example can be that of a spam detector that fails to detect a spam email. The spam email is an example of an adversarial attack in which the attacker has intentionally changed a few words or characters to deceive the spam detector.

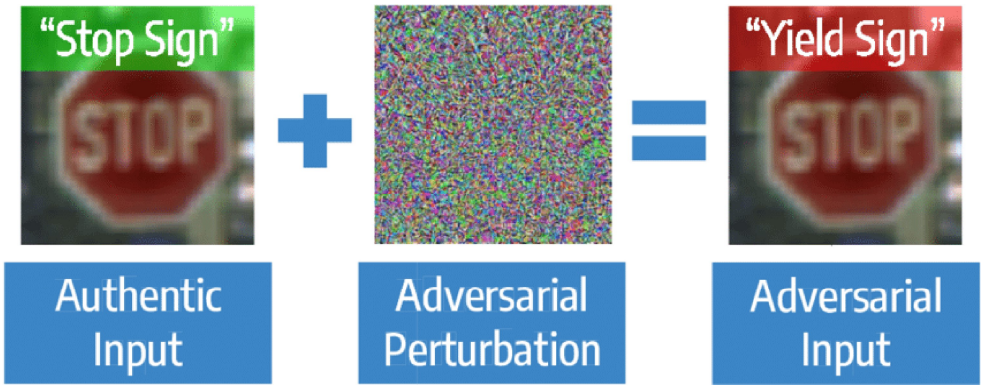


Figure 2.1: Adversarial 'STOP' sign developed by Goodfellow et al. [16]. A well crafted noise is added to the image in such a way that the machine learning algorithm misclassifies the perturbed 'STOP' sign as 'YIELD' sign.

An Adversarial attack can be described as the addition of small perturbation or changes to an

input instance that cause a machine learning model to make a false prediction. Further, the perturbations made are such that they are imperceptible to humans. Given a classifier f which classifies an initial example x as belonging to class y, an adversary intends to convert it into an adversarial example $x^{(adv)}$ such that $x^{(adv)} = x + \delta,\ f(x^{(adv)}) \neq y$. Further, $||\delta||_p \leq \epsilon$, where ϵ gives an upper limit to the perturbation as per a suitable distance metric L_p distance.

The attacks can be broadly classified based on the amount of information available to the attacker as either black box or white box attacks. White box attacks are those in which an attacker has full information about the model's architecture, model weights, and the training dataset. Black box attacks refer to those attacks in which only the final output of the model is accessible to the attacker. Black box attacks can be further classified into three types. The first type involves those attacks in which the probability or confidence scores of the outputs are accessible to the attacker, referred to as the 'score-based black-box attacks'. The second type of attack involves the case where information of the training data is known to the attacker. The third attack type is the one in which only the final classification result of the model is accessible.

2.1 White Box Attacks

White box attacks are those in which the attacker is assumed to have all the information about the model, including architecture parameters. Examples of White Box attack techniques are Fast Gradient Sign Method (FGSM) [16], DeepFool [33], Basic Iterative Method [27], Jacobian Saliency Map Approach [38], Carlini and Wagner Attack [6]

- **Fast Gradient Sign Method (FGSM)** [16] uses the gradient of the underlying model to generate adversarial examples against image classifiers. We add or subtract a small error ϵ to each pixel, depending on whether the sign of the gradient of the loss with respect to the pixel is positive or negative. By adding errors in the direction of the gradient, we are intentionally changing the image so that the classification fails or the image is misclassified to be belonging to a wrong class. Given an input image x which is initially classified by the neural network as belonging to class y, we convert it into an adversarial image $x^{(adv)}$ using the following equation:

$$x^{(adv)} = x + \epsilon \cdot \text{sgn}(\nabla_x J(\theta, x, y)) \tag{2.1}$$

where $\nabla_x J(\theta, x, y)$ is the gradient of the models loss function J with respect to the input pixel vector x; y is the true label vector for x and θ is the model parameter vector, sgn function simply takes the sign of the gradient, positive(+1) or negative(-1). The step size of the perturbation is ϵ and is also the L_∞ distance between $x^{(adv)}$ and x.

- **Basic Iterative Method** [27] involves application of FGSM iteratively with small step size. Given an initial image x_0, the perturbations introduced in the i^{th} iteration are as follows:

$$x_1{}^{(adv)} = x_0$$

$$x_{i+1}^{(adv)} = \mathrm{Clip}_x{}^{\epsilon}(X_i{}^{(adv)} + \alpha \cdot sign(\nabla_x J(\theta, x_i{}^{(adv)}, y))$$

where $i \leq N$, N is the maximum number of iterations, $\mathrm{Clip}_x{}^{\epsilon}$ simply clips the resulting image to be within ϵ ball of x. In [27] α is set to 1 and $N = \min(\epsilon + 4, 1.25 \cdot \epsilon)$.

- **Jacobian Saliency Map Approach(JSMA)** [38] is used to produce adversarial images for targeted misclassification. Given an input image x belonging to a class y, the aim is to apply minimal perturbation to x so that it gets misclassified by the model as belonging to the target class t'. We have an image classifier $f : \mathbb{R}^{D \times E} \mapsto \mathbb{R}^C$, where $D \times E$ is the dimension of the input image x (consider grayscale for simplicity) and C be the number of classes. The classifier function f maps an input image to an array of softmax probabilities. Let $f(x)_c$ denote the softmax probability of the c^{th} class mapped by the classifier function for an input image x and x_i denote the i_{th} pixel. A Jacobian matrix of a multi-variable real valued function is the matrix formed by the first order partial derivatives. The saliency map function can be defined as follows:

$$S^+(x_i, c) = \begin{cases} 0 & \text{if } \frac{\partial f(x)_c}{\partial x_i} < 0 \text{ or } \sum_{c' \neq c} \frac{\partial f(x)'_c}{\partial x_i} > 0 \\ -\frac{\partial f(x)_c}{\partial x_i} \cdot \sum_{c' \neq c} \frac{\partial f(x)'_c}{\partial x_i} > 0 & otherwise \end{cases}$$

$S^+(x_i, c)$ measures how much pixel x_i positively correlates with class c and negatively correlates with all the other classes $c' \neq c$. In all the other cases the saliency is set to 0. The traditional approach uses this saliency map to increase a few high saliency pixels, which leads to an increase in the softmax probability $f(x)_t$, where t is the target class, $t \neq y$. The JSMA approach starts with a search domain consisting of all input pixels, and at each subsequent

step, it finds the most salient pixel pairs (p, q) and perturbs both the pixels by +1, while also removing the saturated pixel indices from the search domain.

- **Carlini and Wagner Attack** [6] is an optimization based approach for producing adversarial images. Given a classifier f mapping an image to one of the m classes, an initial image x belonging to a class y and a distance metric. Given an original image x, the aim is to generate an adversarial image $x + \delta$, such that $f(x + \delta) = t$, while minimizing the distance of the generated example $x + \delta$ from the original image x. The constraint being highly non-linear, makes it difficult to solve the optimization problem. The authors propose a function F such that $F(x+\delta) \leq 0$, whenever $f(x+\delta) = t$. Let the distance metric be represented by p norm, L_p. The authors in their original paper have proposed a variety of functions F that can be used:

$$\text{Minimize: } ||\delta||_p + c \cdot F(x + \delta),\ c > 0 \tag{2.2}$$

$$\text{such that, } x + \delta \in [0, 1]^n \tag{2.3}$$

- **Gradient based attack on Malware classifier** [25] uses a gradient based approach to attack **Malconv** [40], which is a deep neural network trained on raw bytes of Portable Executable (PE) files to discriminate between benign and malicious software. PE format is a file format for executables, DLLs, object code used in 32-bit and 64-bit versions of the Windows operating system. An effective way of manipulating the PE files, while preserving their functionality, is by injecting the bytes at the end of the file [3]. Given an initial binary byte sequence x_0, the attack modifies at most q padding bytes at the end of the file, thereby preserving the functionality of the malware binary. Given a binary classifier that identifies a malware file, the aim of the adversarial attack is to change the decision of the classifier value from 1 (”Malware”) to 0 (”Not Malware”).

$$\min_x f(x) \tag{2.4}$$

$$\text{such that } d(x, x_0) \leq q \tag{2.5}$$

where $d(x, x_0)$ is the number of bytes altered in x_0 to get x. The input bytes are represented by integer values from the set $X = \{1, 2, ..., 255\}$. Padding bytes are then added to the input to make the total number of bytes to be d. An embedding layer then maps each of these bytes to a vector of length 8, thereby converting it into a matrix Z of the shape $d \times 8$. This matrix is then fed to a Convolution Neural Network (CNN) which then outputs a value between 0 and 1. A value close to 0 indicates the file is benign, whereas a value close to 1 indicates that the file is malware.

The attack starts with a malware file which when passed through the **Malconv** CNN gives an output close to 1, and aims to bring down this output probability to 0. For a padding byte x_j, the (negative) gradient of the classifier f with respect to the embedded representation z_j. Let ϕ represent the mapping of the input byte to its embedded representation z_j. The negative gradient is then calculated as $w_j = -\nabla_\phi(x_j)$. A line along this negative gradient direction is then defined as $g_j(r) = z_j + r \cdot n_j$, where n_j is the normalized gradient direction. The padding byte x_j is then replaced with the byte m_i which is closest from this line given its projection on the line in the direction of the negative gradient. Thereby, each byte replacement decreases f, thereby classifying it as benign.

- **A* search in Transformation graph using heuristics** [26]: The white box adversarial attacks in the discrete domain are generalized as a $A*$ graph search problem in the transformation graph. $A*$ is a graph traversal and path search algorithm. It uses a heuristic to plan ahead at each step so a more optimal decision is made. The transformation graph is a weighted directed graph, where each node represents a possible vector in the input space and each edge is a transformation, the edge weight represents the transformation cost and its children nodes are the transformed examples. The authors use Taylor's expression and Hölder's inequality to come up with a heuristic for the graph search. Let there be an initial input $x \in R^n$ to which we do a perturbation $\delta \in R^n$ which puts the perturbed example, $x + \delta$, on the decision boundary. Assuming that the decision boundary is $\theta = 0$, we end up with $f(x + \delta) = 0$. The first-order Taylor approximation of $f(x + \delta)$ at the point x is as follows:

$$f(x + \delta) = f(x) + \nabla f \cdot (x + \delta - x)$$

$$0 = f(x) + \nabla f \cdot \delta$$

$$\delta = \frac{|f(x)|}{|\nabla_x f(x)|^q} \tag{2.6}$$

If the edge costs are induced by L_p norm, then q is the Holder's conjugate of p given by $\frac{1}{p} + \frac{1}{q} = 1$.

2.2 Black Box Attacks

Black box attacks can be of the following three types:

- **Score-Based attacks**: Attackers can query the softmax layer output in addition to the final classification result. The softmax layer tells the probability scores for different classes as predicted by the deep learning model.

  – Scores Feedback method developed by Guo et al. [18] uses the confidence scores as feedback to craft a black-box adversarial attack on image classification. The algorithm takes a random direction at every iteration, from a pre-specified set of orthogonal directions in the input space. The pixel values are then either increased or decreased in that direction, whichever leads to a decrease in the confidence score of the original class. Consider a black box image classifier f and a given image x, such that $f(x) = y$ with predicted confidence $P_f(y|x)$. The aim of the attack is to introduce perturbation δ, such that $f(x + \delta) \neq y$. The attack begins by constituting a set Q of orthogonal vectors in the input space. Next, a random direction $q \in Q$ is selected and a perturbation is first added in direction q to the pixels of the input image x with a step size ϵ. If $P_f(y|x + \epsilon q) > P_f(y|x)$, then we rather subtract the ϵ perturbation in direction q, else we go ahead with perturbation added to our image x. This process is repeated until the perturbed image is misclassified by the black box model or until the maximum number of iterations T is reached. Further, it can be noted that the maximum perturbation introduced after T iterations can be given by, $|\delta|_2 = \sqrt[2]{T}\epsilon$. So, the step size ϵ can be selected to keep an upper bound on δ. Also, selecting q randomly from Q with replacement is

necessary to ensure that no two directions cancel each other and diminish progress or amplify each other in subsequent steps and disproportionately increase the perturbation δ.

  - **GenAttack** [1] uses a genetic algorithm based approach for gradient-free optimization to generate adversarial images. The fitness function uses the output scores for different classes, maximizing the log scores of the target class, and minimizing the log scores of all other classes.

- **Transfer based attack** is the type of black-box attack meant for Deep Neural Networks when there is a restriction on the number of queries that an attacker can make. The attackers try to construct a substitute model f' to mimic the original DNN model f and then attack f' using white-box attack methods [37] [30]. An important requirement for such an attack is the generation of a training dataset from the same population on which the original model f was trained. These attacks are based on the observations made by Szegedy et al. [43], with regard to the transferability between DNN architectures when trained on datasets from the same population.

- **Decision based attack or Hard Label Black Box** attacks assume that only the final class decision for a given input x is accessible to the attacker while the confidence scores are not known.

  - **Evolutionary Algorithms based approach** [10] was devised to attack Twitter bot classifiers which depend on user account information like the number of tweets, retweets, and replies for its decision making. The authors use the DNA-like representation of the lifetime of each of the Twitter accounts. In the DNA-like representation, the timeline of a Twitter user is represented by encoding every tweet as T, retweet as A, and reply as C. The authors take the Longest Common Substring (LCS) of the DNA-like sequences to measure the similarity between different users. Given a group of M users, the LCS curve is generated by plotting the length of LCS for different number users ranging from 2 to M. The LCS curve is taken as the behavioral similarity among a group of users and the Kullback Leiber(KL) divergence is used to quantify the similarity between the curves.

par In each iteration of the genetic algorithm, a group of spambot account DNAs was evolved and the KL divergence between the LCS curves of legitimate accounts and evolved spambots was minimized. Although, the evolved spambots after a set of iterations have been shown to evade state-of-the-art classifiers the authors do not report the average number of changes made to the spambot DNA in order to evade classification. The average number of changes made to the Twitter account in order to evade the classifier is critical, as the attacker has to pay in order to add or delete tweets, retweets, or replies.

– **Random Walk on the Boundary** method devised by Brendel et al. [5] described a random walk on the boundary approach to finding adversarial images. Consider an initial image x which is labeled by an image classifier as belonging to class y. The algorithm starts from an image x' with a label y' such that $y' \neq y$. A random walk is then performed along the boundary between adversarial and non-adversarial region such that it stays in the adversarial region and the distance from the original image is reduced after each iteration.

– **Optimization based approach** proposed by Cheng et al. [8] formulated the hard label black box attack as an optimization problem and solved it using a zeroth-order optimization technique known as Randomized Gradient Free Method [35]. Zeroth-order optimization refers to the optimization techniques used in problems where we do not have explicit access to the gradients. The loss function in this case, where only the final decision is accessible to the attacker, is discontinuous with discrete outputs. Such an optimization problem requires a combinatorial optimization technique but is computationally expensive because of the high dimensionality of the input. The problem is solved by formulating the hard label black box attack in the following way:

Consider a black box model $f : R^d \rightarrow \{1, 2, ..., M\}$ which classifies our image x_0 as belonging to label y_0 and has a closed decision boundary around it. The aim of the adversarial attack is to move out of this decision boundary along the shortest path. A function g is defined which tells us the minimum L_2 distance to cross the decision

boundary in a given direction θ.

$$g(\theta) = \min_{\lambda>0} \lambda \tag{2.7}$$

$$\text{s.t. } f\left(x_0 + \frac{\lambda\theta}{\|\theta\|}\right) \neq y_0 \tag{2.8}$$

The optimization problem can then be defined as follows: Starting from our image x_0, find the direction θ along which we can cross the decision boundary with minimum path length.

$$\min_{\theta} g(\theta) \tag{2.9}$$

The given problem is solved using Randomized Gradient Free method (RGF). RGF is a popular zeroth order optimization method proposed by [35]. It estimates the gradients using the formula:

$$\hat{g} = \frac{g(\theta + \beta u) - g(\theta)}{\beta} \cdot u$$

where u is a random Gaussian vector and β is a smoothing vector.

Chapter 3

Adversarial Attacks on NLP

3.1 Introduction

Natural Language Processing (NLP) is a branch of Linguistics and Artificial Intelligence that deals with processing and analyzing huge amounts of natural language data. NLP aims to match human intelligence in reading and understanding human languages. Such a kind of understanding of human languages enables real-world applications like sentiment analysis, automatic text summarization, topic extraction, named entity recognition, parts-of-speech tagging, stemming, relationship extraction, and more. A brief description of each of these is as follows:

- **Sentiment Analysis** helps to interpret and classify the emotions within text data. The classes involved are primarily positive, negative, or neutral. It helps the businesses to understand the emotions of a customer towards their products or services. Reviews or survey responses are fed to the machine learning model which then classifies into different classes signifying different emotions.

- **Textual Entailment (TE)** aims at predicting whether, for a pair of sentences, the facts in the first sentence necessarily imply the facts in the second.

- **Text Summarization** refers to creating a short and accurate summary of a longer text. It involves extracting the most information from a source. Automatic text summarization is useful in a scenario where there is an abundance of data and a lack of manpower and time to summarize important information or interpret the data.

- **Topic Extraction** deals with extracting keywords and phrases in the text data. It is not constrained by a given list of topics or classes.

- **Named Entity Recognition** seeks to classify and locate named entities in an unstructured

text. The classes can be person names, locations, organizations, monetary values, percentages, etc.

- **Parts-of-speech tagging** reads a text in a particular language and assigns part-of-speech tags to each word (or other tokens), such as noun, word, adjective.

We shall mainly focus on the text classification tasks like sentiment analysis, spam detection, topic modeling. Sentiment analysis is widely used in the online recommendation systems, where the reviews and comments are classified into a set of categories that are useful while ranking products or movies [31]. Text classification is also used in applications critical for online safety, like online toxic content detection [36]. Such applications involve classifying the comments or reviews into categories like irony, sarcasm, harassment, and abusive content.

3.2 Review of Adversarial attacks on NLP

In this section, we discuss a few prominent NLP attack frameworks.

1. **TextBugger** [29] is a framework for utility preserving adversarial attacks against text classification in both, white and black box settings. Utility preserving refers to the preservation of the original meaning of a piece of text as inferred by human readers. In the black-box scenario, the framework assumes that the attacker can query the text classifier to be attacked and get the confidence scores of various classes.

    - The attack algorithm for white box attack is as follows:
        - **Find important words**: The authors use the Jacobian matrix to rank the words. Jacobian matrix consists of partial derivatives of the confidence values for different classes, as predicted by the classifier, with respect to different words in the text. Consider a text classifier function F, and the input text has N words whose word vectors are represented as $x = \{x_1, x_2, ..., x_N\}$. The output of F is an array of probabilities or confidence scores for different classes. Let there be k classes, so $F_j(\cdot)$ represents the confidence scores of the j^{th} class. The Jacobian matrix can then be defined as follows:

$$J_F(x) = \frac{\partial F(x)}{\partial x} = [\frac{\partial F_j(x)}{\partial x_i}]_{i=1,2,...,N;j=1,2,...,k}$$

The importance of i^{th} word x_i for a given class y is defined as follows:

$$IMP_{x_i} = \frac{\partial F_y}{\partial x_i}$$

– **Perturb words in order of word importance**: Once the words are assigned an importance value for the target class, they can be perturbed on the character level or the word level.

  * Character level perturbations just change a few characters in a word. Since the Deep Learning based Text Understanding (DLTU) systems use a fixed dictionary of words, changing a few characters in a given word makes it an unknown word for the DLTU system. Such perturbations preserve the original meaning of a text [41].
  * Word level perturbations replace a word with a synonym or semantically similar word. Synonyms can be searched using word embeddings in a context-aware word vector space. Context-aware word embedding means that synonyms are closer in the vector space with a given distance metric (mostly cosine distance). The pre-trained GloVe model [39] provides one such word embedding technique, which can be used to get top k synonyms of a given word.

TextBugger introduces five types of perturbations and selects the one which results in the maximum decrease in confidence of the original class. The five types of perturbations are as follows:

  * Inserting a space in between the word changes the word by fragmenting it into two separate words.
  * Delete a random character in between the word, except the first and last character.
  * Swap two random characters in the word, except the first and last character
  * Replace a random character with a visually similar character. For e.g., 'I' with '1', 'O' with '0'.
  * Replace a word with its k nearest neighbors using a word embedding from the pre-trained GloVe model.

The perturbations are done until the text is misclassified by the text classifier or the maximum number of iterations is reached.

- The black box attack algorithm is as follows:
  - **Find important sentences**:
    - Segment the given text into sentences, $S = \{s_1, s_2, ..., s_m\}$
    - Query the black box model by inputting each of these sentences individually.
    - Filter out the sentences which have a different class then the original class y of the entire piece of text.
    - Sentence importance is then defined for each of the remaining sentences as follows,

$$IMP_{s_i} = F_y(s_i)$$

  - **Find word importance**: For each of the sentences, we find the importance of the words in those sentences by removing the word from the text and passing the text through the text classifier to check the influence of the word on the classification result.F. Given the original class of the text being y, the importance of word w_j is then given as follows:

$$IMP_{w_j} = F_y(x) - F_y(x \setminus \{w_j\})$$

    where $F_y(.)$ is used to calculate the confidence score of class y for a given text.
  - **Word perturbation** algorithm is the same as that for the white box attack framework.

2. **TextFooler** [22] is a black box NLP attack framework. It attacks text classification and textual entailment. The attack algorithm is as follows:
   - Find word importance ranking,

$$IMP_{w_j} = F_y(x) - F_y(x \setminus \{w_j\})$$

   - Perturb the words in order of high importance scores. TextFooler uses only word-level perturbations and more stringent selection criteria as compared to TextBugger. The perturbation steps are as follows:

- Get the synonyms using context-aware word embeddings [34].
- Remove the synonyms with different POS tags from that of the word to be replaced.
- Semantic Similarity between the adversarial text with synonym and the original text to be within ϵ limit. The authors use the Universal Sentence Encoding (USE) [7] to encode sentences into high dimensional vectors and use the cosine similarity as a measure of semantic similarity.

3.3 Hard Label Black-box attack

A Hard Label Black-box attack is the one in which only the final decision of the classifier is accessible by the attacker, with no access to the confidence scores of the various classes. In the NLP domain, researchers have mainly formulated attacks in the white box setting [13] [29] with complete knowledge of gradients or the black-box setting with confidence scores accessible to the attacker [15] [29] [2] [22]. As per our knowledge, there has been no prior work done to formulate adversarial attacks against NLP classifiers in the hard label black-box setting. Since the confidence scores can easily be hidden to avoid simple attacks, hard-label black-box attacks constitute an important category of attacks relevant to real-world application. In the next chapter, we devise an algorithm that attacks text classification in the hard label black-box setting.

Chapter 4

TextDecepter

4.1 Proposed Methodology

In this chapter we propose TextDecepter, an algorithm that generates natural language adversaries against text classification models in the hard-label black-box setting. The basic methodology of TextDecepter is based on the premise that not all the sentences in a text convey the final opinion of the text with the same rigor. When people are expressing their opinions, some sentences are simply facts, which do not contribute to the final opinion, and other sentences contribute to the final opinion in varying degrees. Hence, an algorithm is crafted to select the important sentences from a given text. Then, the words are ranked within each sentence using their Part of Speech (POS) tags.

Subsequently, we replace the words with suitable synonyms in such a way that the following utility-preserving properties are fulfilled:

1. Semantic similarity: The adversarial text should bear the same meaning as the source text.

2. Language fluency: Adversarial examples should look natural and grammatically correct.

   The algorithm keeps replacing the words until the decision of the text classifier for the text changes.

4.2 Attack Design

4.2.1 Problem Formulation

Given an input text space X and a set of n labels, $Y = \{Y_1, Y_2, ..., Y_n\}$, we have a text classification model $F : X \rightarrow Y$ which maps from the input space X to the set of labels Y. Let there be a text $x \in X$ which is correctly predicted by the model to be belonging to the class $y \in Y$, i.e., $F(x) = y$. We also have a semantic similarity function $Sim : X \times X \rightarrow [0, 1]$. Then, a successful adversarial

attack changes the text x to $x^{(adv)}$, such that

$$F(x^{(adv)}) \neq F(x)$$

$$Sim(x, x^{(adv)}) \geq \epsilon$$

where ϵ is the minimum similarity between original and adversarial text.

Consider a binary classification model with labels $\{Y_0, Y_1\}$ which we want to attack. We are given a text $T \in X$ having m sentences, $S = \{s_1, s_2, ..., s_m\}$. The classification model correctly predicts T to be belonging to class Y_0, i.e., $F(T) = Y_0$. We input each of the m sentences to F and get their individual labels. Let A and B form a partition of S such that $F(a) = Y_0,\ \forall a \in A$ and $\mathcal{P}$ (A) be the power set of A. Let $\mathcal{A}_r$ be a set of r-combinations of A. We define another set $G = \{B \cup c \mid c \in \mathcal{P}(A)\}$. We refer to each of the elements in G as an '**aggregate**'.

4.2.2 Threat Model

Given a binary text classification model is meant to classify a given text as having a positive or negative sentiment, the attacker aims to evade the classifier by, say, making perturbations to a negative sentiment text so that it is now classified by the model as positive. However, the perturbed text remains negative for human observers. An example of such an attack is when a social media platform employs a machine learning text classification model to filter out the negative sentiment posts made by malicious users during the times of pandemic. The social media platform allows the users to post texts any number of times and gives feedback about the text if it is negative or positive. Further, an attacker may also want to perturb a positive sentiment text so that it is misclassified by the model as negative. and removed from the platform. However, the text remains positive for human observers and hence, the attacker can malign the reputation of the social media platform by accusing them of political inclinations. We consider the attack in the black-box setting, where an attacker does not have any information about the model weights or architecture and is allowed to query the model with specific inputs and get the final decision of the classifier model as an output. Further, the class confidence scores are not provided to the attacker in the output, making it a hard-label black-box attack. Although NLP APIs provided by Google, AWS, and Azure provide the confidence scores for the classes, in a real-world application setting, like toxic content detection on a social media

platform, the confidence scores are not provided, thereby making it a hard label black-box setting. Such an attack scenario also helps to gauge the model robustness.

4.2.3 Methodology

The proposed methodology for generating adversarial text has three main steps:

- **Sentence Importance Ranking**: We observe that when people convey opinions or emotions, not all the sentences convey the same emotion. Some sentences are facts presented without any emotion or sentiment. Other sentences can also be stratified based on varying levels of intensity. This forms the basis of our sentence ranking algorithm which helps to prioritize our attack on specific portions of the text in order of importance. We assume that different sentences in the text contribute to the overall class decision to a varying level of intensity. Each of the sentences can either support or oppose the final decision of the classifier and the intensities which they do so are additive. Consider an example of sentiment analysis, where the labels are positive and negative. The assumption of additivity of sentence class intensity, also helps us to infer that sentences in set B when joined together to form a text, will belong to class Y_1. Hereby, we refer to the same as the class of set B or the classifier's decision of set B.

  We define the importance of a sentence in set A by its ability to change the classifier's decision of set B from Y_1 to Y_0. If an individual sentence from set A when added to set B is able to change the class of set B from Y_1 to Y_0, then we consider the sentence to belong to ***level 1 importance***. More generally, if a sentence from set A is able to change the classifier's decision of set B only when it is put together with some subset of A with at least $k-1$ sentences, then the sentence belongs to ***level k importance***. The $k-1$ other sentences in all such subsets also belong to ***level k importance***. Also, once the importance of a sentence is fixed at the k^{th} level we do not consider it in subsequent levels.

- **Word importance ranking**:

  After finding the importance of sentences in step 1, we need to find the importance ranking of the words to be attacked in these sentences. We observe that words with a certain Part of

Algorithm 1: Sentence Importance Ranking

Input: Original Sentences set S, ground truth label y_0, classifier $F(.)$

Output: Sentence Importance Ranking

SentsSentiment $\leftarrow$ Find original labels of sentences in S ;

OrigLabelSents $\leftarrow$ Sentences with Label Y_0 ;

OtherLabelSents $\leftarrow$ Sentences with Label $\neq Y_0$;

Let OrigLabelSents$_P$ represent set of all P sentence combinations from OrigLabelSent ;

$P \leftarrow 1$;

while *OrigLabelSents* **do**

    TopSentImp$_P \leftarrow \phi$;

    for *Comb in OrigLabelSents*$_P$ **do**

        AGG $\leftarrow$ Add $Comb$ to $OtherLabelSents$ and join it to form string ;

        if $F(AGG) = Y_0$ **then**

            Aggregates $\leftarrow$ Add AGG;

            for *sent in Comb* **do**

                SentImp[$sent$] $\leftarrow P$;

                TopSentImp$_P \leftarrow$ Add $sent$

            end

        end

    end

    Delete sentences in TopSentImp$_P$ from OrigLabelSents ;

    $P \leftarrow P + 1$;

    if $P > Length(OrigLabelSents)$ **then**

        Add remaining sentences in OrigLabelSents to TopSentImp$_P$;

    end

end

return SentImp, Aggregates

Speech (POS) tags are more important than others. For example, for a sentiment classification task, adjectives, verbs, adverbs are more important than nouns, pronouns, conjunctions, or prepositions. Further, we consider adjectives more important than adverbs. Consider a sentence, "The movie was very bad". In this sentence, "bad" is the adjective and shapes the sentiment of the sentence. The adverb "very" increases the intensity of the adjective, making the predicted class confidence score increase further.

- **Attack**: We use the word-level perturbations in order of word importance obtained from the previous step. We select synonyms to replace the original words using cosine distance between word vectors. Further, to maintain the syntax of the language, only those synonyms having the same POS tags as that of the original word are considered for further evaluation. Experiments are done using both coarse and fine POS tag masks.

  The details of each of these steps are as follows:

  1. Synonym extraction: We use the counter-fitted word embedding [34] which obtained state-of-the-art performance on SimLex-999 [19], a dataset designed to measure how well different models judge the semantic similarity between words.
  2. POS Checking: To maintain the syntax of the adversarial example, we filter out the synonyms which have a different POS tag than the original word. We experiment with, both, coarse and fine POS tagging.

  We select a synonym to replace a word if its usage leads to one of the following:

  1. Misclassification of the entire text.
  2. Misclassification of the original labeled sentence to which the target word belongs.
  3. Misclassification of the original label aggregate to which the target word belongs.

If multiple synonyms fulfill the rules, then the one which fulfills the rule of higher preference is selected. If multiple synonyms fulfill the highest preference rule, then that synonym is selected whose placement in the review is semantically nearest to the original review. We terminate the algorithm once the text misclassifies, or when all the important words have been iterated over.

Algorithm 2: TextDecepter

Input: Original text X, ground truth label Y_0, classifier $F(.)$, semantic similarity threshold ϵ, cosine similarity matrix

Output: Adversarial example $X^{(adv)}$

Initialization: $X^{(adv)} \leftarrow X$;

Segment X into sentences to get set S;

SentImp, Aggregates = GetSentenceImp(S);

WordPerturbSequence = GetWordImp (SentImp);

MISCLASSIFIED ← False ;

for w_j *in WordPerturbSequence* **do**

  if *MISCLASSIFIED* **then** break;

  CANDIDATES = GetSynonyms(w_j);

  CANDIDATES ← POSFilter (CANDIDATES) ;

  CANDIDATES ← SEMANTICSIMFilter (CANDIDATES) ;

  FINCANDIDATES ← Sort CANDIDATES by semantic similarity ;

  CHANGED ← False ;

  for c_k *in FINCANDIDATES* **do**

    $X' \leftarrow$ Replace w_j with c_k in $X^{(adv)}$;

    if $F(X') \neq Y_0$ **then** $X^{(adv)} = X'$; CHANGED ← True; MISCLASSIFIED ← True;

    if *NOT CHANGED* **then**

      SENT ← Get the sentence in which w_j belongs s.t F(SENT) = Y_0 ;

      $X' \leftarrow$ Replace w_j with c_k in SENT;

      if $F(X') \neq Y_0$ **then** $X^{(adv)} = X'$; CHANGED ← True ;

    end

    if *NOT CHANGED* **then**

      AGG ← Get the aggregate in which w_j belongs s.t. $F(AGG) = Y_0$;

      $X' \leftarrow$ Replace w_j with c_k in AGG ;

      if $F(X') \neq Y_0$ **then** $X^{(adv)} = X'$; CHANGED ← True;

    end

  end

end

return $X^{(adv)}$

The justification for the higher preference of sentence with respect to the aggregate to which it belongs comes from the additivity assumption. Consider a sentence $v \in A$, $\{v\} \in \mathcal{A}_1$. Now, add it to set B to form an aggregate, i.e., $B \cup \{v\} \in G$. Assuming the additivity of class intensities of sentences, it can be observed that when sentences in $B \cup \{v\}$ are joined to form a piece of text, it either belongs to class Y_1 or, in case, it belongs to class Y_0, then the intensity of class Y_0 is lesser when compared to v alone. In other words, an aggregate belonging to class Y_0 has a lesser intensity of class Y_0 when compared with individual sentences belonging to class Y_0 which are part of that aggregate. Hence, a synonym which is able to misclassify both the individual sentence and the aggregate (both initially belonging to Y_0) to which the individual sentence belongs would be preferred over a synonym that misclassifies the aggregate alone.

4.3 Backtracking

Consider a directed acyclic graph, where a node represents a text and a directed edge between two nodes means that the latter node text can be obtained by replacement of a word w in former node text by its most semantically similar synonym w', given that the synonym has the same POS tag as the original word. Further, the weight of each edge is 1. Then, crafting an adversarial example from a given piece of text can be formulated as a pathfinding problem as follows; find a path from an initial node T to final node T', such that $F(T) = Y_0$ and $F(T') \neq Y_0$. The sentence and word ranking algorithm decides the order of perturbation. Thereafter, we use the aggregates belonging to the original class as heuristics to guide our graph search. The algorithm stops once the text misclassifies. Although the heuristics help us in finding a path, they do not guarantee if it is the shortest path. In the absence of confidence scores of classes, the problem of finding the shortest path has an exponential search space. So, the algorithm just backtracks over all the replaced words at the end and reset those replacements without which the perturbed text remains adversarial.

4.4 Time Complexity

Let there be n sentences in a text and m words in total.

1. Getting the class label of each of the individual sentences: $O(n)$

2. Sentence Importance ranking: The algorithm takes all possible combinations of sentences from the set of original labeled sentences and inserts each of them to the set of sentences that do not have the original label. Hence, the time complexity of this step becomes $O(2^n)$.

3. Word importance ranking:

   (a) Sort the original labeled sentences in order of their importance ranking: $O(n \log n)$

   (b) Iterate over all the words in the sorted sentences and make the word perturb sequence: $O(m)$

   (c) Generation of cosine similarity matrix of words in the text with sixty-five thousand words in the vocabulary: $O(m)$

4. Pick the top k synonyms for each of the candidate words that are to be perturbed. The cosine similarity matrix consists of the similarity of each of the words in the text with all the words in the vocabulary. We sort the similarity values for each word with all the other words in the vocabulary. Hence, amounting to a total time complexity of $O(m)$

5. Attack: Replace each of the k synonyms of a word in the original text, the original labeled aggregates (obtained from sentence importance ranking step), and the original labeled sentences to which they belong. Hence, the time complexity of this step becomes $O(kmn)$.

6. Removal of insignificant perturbations: $O(m)$

Hence, the overall time complexity of the attack framework becomes $O(2^n) + O(kmn)$

4.5 Memory requirement

The generation of adversarial texts requires that we are able to get the synonyms for a particular word during the attack phase. Hence, a cosine similarity matrix is kept in the RAM to quickly generate synonyms for a particular word. In our attack framework, we keep the cosine similarity of only the words in the text that is being attacked, with all the sixty-five thousand words in the vocabulary. It is an improvement over the memory requirement in the attack framework of TextFooler [22] which keeps the cosine similarity matrix of all the sixty-five thousand words in the vocabulary, constituting

a square matrix of size sixty-five thousand, consuming nearly 17 GB of RAM if single-precision floating-point format is used.

Chapter 5

Attack Evaluation

We evaluate our attack methodology by generating adversarial texts against text classification models meant for sentiment analysis task.

5.1 Datasets and Models

We study the effectiveness of our attack methodology on sentiment classification on IMDB and Movie Review (MR) datasets. We target three models: word-based convolutional neural network (WordCNN) [24], word-based long-short term memory (WordLSTM) [21], and Bidirectional Encoder Representations from Transformers (BERT) [12]. We attack the pre-trained models open-sourced by [22] and evaluate our attack algorithm on the same set of 1000 examples that the authors had used in their work. We also run the attack algorithm against Google Cloud NLP API. The summary of the datasets used by [22] for training the models are in Table 5.1 and their original accuracy are given in Table 5.2

Task	Dataset	Train	Test	Avg Len
Classification	MR	9K	1K	20
	IMDB	25K	25K	215

Table 5.1: Overview of the datasets used by [22] for training the models

5.2 Evaluation Metrics

1. Attack success rate: The difference between the original and after-attack accuracy of a text classification model on a given dataset is called the attack success rate.

	wordCNN	wordLSTM	BERT
MR	79.9	82.2	85.8
IMDB	89.7	91.2	92.2

Table 5.2: Original accuracy of the target models on standard test sets

2. Percentage of perturbed words: The average percentage of words replaced by their synonyms gives us a metric to quantify the change or perturbation made to a given text.

3. Semantic similarity: Universal Sentence Encoder (USE) [7] is used to encode the original and adversarial texts and semantic similarity is calculated by finding the cosine similarity between the encodings. Since the main aim of the attack is to generate adversarial texts, we just control the semantic similarity to be above a certain threshold.

4. Number of queries: The average number of queries made to the target model tells us the efficiency of the attack model.

5.3 Results

5.3.1 Automatic Evaluation

We report our results of the hard label black-box attacks in terms of automatic evaluation on two text classification tasks using coarse and fine POS masks. The main results are summarized in Tables 5.3 and 5.4. The attack algorithm is able to decrease the accuracy of all the major text classification models with an attack success rate greater than 50%. The percentage of perturbed words is nearly 3% for all the models on the IMDB dataset and between 10-16% for all the models on the MR dataset. In the IMDB dataset, which has an average word length of 215 words, our attack model is able to conduct successful attacks by perturbing less than 7 words on average. That means that our attack model can identify the important words in the text and make subtle manipulations to mislead the classifiers. Overall, the algorithm can attack text classification models for sentiment analysis regardless of text sequence length or target model accuracy. Further, our attack model requires the least amount of information among all the comparative works [22] [29], both of which require the

confidence scores.

The attack model is also able to attack GCP NLP API and decrease its accuracy from 76.4% to 16.7% for the MR dataset. Further, it changes only 10.2% of the words in the text to generate the adversary. The attack algorithm and the carefully crafted adversarial texts can be utilized for the study of interpretability of the BERT model [14].

The query number is almost linear to the text length, with a ratio in (6,10) which is at par with [22] and [29].

	wordCNN		**wordLSTM**		**BERT**		**GCP NLP API**
	MR	**IMDB**	**MR**	**IMDB**	**MR**	**IMDB**	**MR**
Original Accuracy	78	89.4	80.7	90.3	90.4	88.3	76.4
After-attack accuracy	18.9	17.3	18.9	32.5	42.3	30.9	16.6
Attack Success rate	75.8	80.6	76.6	64.0	53.2	65.0	78.3
% Perturbed Words	12.1	3.1	12.2	2.8	15.6	2.1	11.8
Query number	133.2	1368.6	123.2	1918.1	189.5	1719.7	126.8
Average Text Length	20	215	20	215	20	215	20

Table 5.3: Automatic evaluation results on text classification datasets (using coarse POS mask)

	wordCNN		**wordLSTM**		**BERT**		**GCP NLP API**
	MR	**IMDB**	**MR**	**IMDB**	**MR**	**IMDB**	**MR**
Original Accuracy	78	89.4	80.7	90.3	90.4	88.3	76.4
After-attack accuracy	20.7	18.9	21.2	34.4	45.9	33.3	16.6
Attack Success rate	73.5	78.9	73.7	61.9	49.2	62.3	78.3
% Perturbed Words	12.2	3.1	12.0	2.6	14.6	2.2	10.2
Query number	112.5	1230.0	107.0	1650.5	159.0	1507.4	109.6
Average Text Length	20	215	20	215	20	215	20

Table 5.4: Automatic evaluation results on text classification datasets (using fine POS mask)

5.3.2 Benchmark Comparison

We compare our attack against state-of-the-art adversarial attack systems on the same target model and dataset. For GCP NLP API, we compare our attack results against [29] and [15] on MR datasets. With wordCNN and wordLSTM as the target models, the comparison is against [29], [2], [22]. The results of the comparison are summarised in table 5.5 and 5.6. The lower attack success rates, when compared to the other attack systems, can be attributed to the fact that unlike other published systems, our attack system does not make use of confidence scores of the classes.

Attack System	**Attack Success Rate**	**%Perturbed Words**
Li et al [29]	86.7	6.9
Alzantot al [2]	97.0	14.7
Jin et al. [12]	99.7	10.0
Ours	**64.0**	**2.4**

Table 5.5: Comparison of our attack system against other published systems with wordLSTM as the target model (Dataset: IMDB)

Attack System	**Original Accuracy**	**Attack Success Rate**	**%Perturbed Words**
Gao et al. [15]	76.7	67.3	10
Li et al. [29]	76.7	86.9	3.8
Ours	76.4	78.1	10.2

Table 5.6: Comparison of our attack system against other published systems with Google Cloud NLP API as the target model (Dataset: MR)

5.3.3 Human Evaluation

Following the practice of Jin et al. [22], we perform human evaluation by sampling 100 adversarial examples from the MR dataset with the WordLSTM. We perform three experiments to verify the

	Fine POS filter	Coarse POS filter
Original	4.5	4.5
Adversarial	4.3	3.9

Table 5.7: Grammaticality of original and adversarial examples for MR (BERT) ON 1-5 scale

quality of our adversarial examples. First, the human judges are asked to give the grammaticality score of a shuffled mix of original and adversarial text on a scale of 1-5. As shown in Table 5.7, the grammaticality of the adversarial texts generated using fine POS tag mask is closer to the original texts when compared with the ones generated using coarser POS tag mask.

Secondly, the judges assign classification labels to a shuffled set of original and adversarial texts, for both coarse and fine POS masks. The results show that the overall agreement between the labels of the original and adversarial text for both the cases are quite high, 92% and 93% respectively. This suggests that improving the grammaticality of the adversarial texts using a fine POS mask does not contribute much to the overall meaning of the texts to humans.

Thirdly, the judges determine whether the adversarial texts retain the meaning of the original text. The judges are given three options: 1 for similar, 0.5 for ambiguous, and 0 for dissimilar. The average sentence similarity score is 0.88 when fine POS mask is used compared to 0.86 when a coarse POS mask is used for synonym selection, suggesting a marginal improvement in sentence similarity scores in the former.

5.4 Comparison of Fine and Coarse POS tag filter

Part of Speech (POS) tags can be of two types: Coarse-grained (Noun, Verb, Adjective, etc.) or Fine-grained (Noun-proper-singular, noun-proper-plural, verb-past, verb-present, etc). TextDecepter uses a POS mask to reject those synonyms which do not belong to the same POS as that of the original word when they are placed in the text. Fine-grained POS mask helps to maintain the grammaticality to a greater extent when compared to the Coarse-grained POS mask as demonstrated in 5.8. However, during the human evaluation, we observed that improving the grammaticality of the adversarial

texts for sentiment classification dataset using fine POS does not contribute much to the overall understanding of the text for humans. We recommend using the appropriate POS mask depending on the grammaticality requirement for the context in which adversarial texts are being generated.

	Movie Review (Positive ↔ Negative)
Original (Label: POS)	she may not be real , but the **laughs** are
Attack (Label: NEG) using coarse POS tags	she may not be real , but the ***kidding*** are
Attack (Label: NEG) using fine POS tags	she may not be real , but the ***chuckles*** are
Original (Label: NEG)	falsehoods pile up , ***undermining*** the movie 's reality and stifling its creator 's comic voice
Attack (Label: POS) using coarse POS tags	falsehoods heaps up , ***jeopardizes*** the movie 's reality and stifle its creator 's comic voice
Attack (Label: POS) using fine POS tags	falsehoods heaps up , ***jeopardizing*** the movie 's reality and stifle its creator 's comic voice

Table 5.8: Qualitative comparison of adversarial attacks with coarse and fine POS tagging for synonyn selection. Target Model is wordLSTM

5.5 Ablation study

The most critical step of our TextDecepter is the use of aggregates, which belong to the original class, to select or reject synonyms for replacement during the attack phase. We conduct an ablation study to showcase the importance of aggregates to our algorithm. To validate the effectiveness of aggregates, we remove their usage while selecting or rejecting a synonym for replacement. Now a synonym is selected for replacement only when its usage misclassifies the entire original text. The

results for the BERT model are shown in table 5.9. After removing the usage of aggregates for synonym selection, the after-attack accuracy increases by 32% for IMDB and 35% for MR dataset, respectively. This suggests the importance of aggregates for selecting synonym for replacement, the removal of which renders the attack ineffective. The aggregates generated in the sentence importance ranking step enable the selection of synonyms which can take the original text towards misclassification.

	Orig Acc.	**After-Attack accuracy**		**% Perturbed words**	
		w/ agg.	**w/o agg.**	**w/ agg.**	**w/o agg.**
IMDB	88.3	30.9	63.2	2.1	0.6
MR	90.4	42.4	75	13.6	10.1

Table 5.9: Comparison of the after-attack accuracy of the BERT model with and without using aggregates for synonym selection

	Movie Review (Positive (POS) $\leftrightarrow$ Negative (NEG))
Original (Label: NEG)	i firmly ***believe*** that a good video game movie is going to show up soon i also believe that resident evil is not it
Attack (Label: POS	i firmly ***feel*** that a good video game movie is going to show up soon i also believe that resident evil is not it
Original (Label: POS)	strange and ***beautiful*** film
Attack (Label: NEG)	strange and ***resplendent*** film
Original (Label: POS)	the lion king was a roaring ***success*** when it was released eight years ago , but on imax it ***seems*** better, not just bigger
Attack (Label: NEG	the lion king was a roaring ***attainment*** when it was released eight years ago , but on imax it ***transpires*** better , not just bigger

Table 5.10: Examples of original and adversarial sentences from MR (GCP NLP API)

	Movie Review (Positive (POS)↔ Negative (NEG))
Original (Label: NEG)	after the book i became very ***sad*** when i was watching the movie . i am agree that sometimes a film should be different from the original novel but in this case it was more than acceptable . some examples: 1) why the ranks are different (e.g. lt . diestl instead of sergeant etc.) 2) the final screen is very ***poor*** and makes diestl as a soldier who feds up himself and wants to die . but it is not true in 100 % . just read the book . he was a bull - dog in the last seconds as well . he did not want to die by wrecking his gun and walking simply towards to michael & noah . so this is some kind of a happy end which does not fit at all for this movie .
Attack (Label: POS)	after the book i became very ***bleak*** when i was watching the movie . i am agree that sometimes a film should be different from the original novel but in this case it was more than acceptable . some examples:1) why the ranks are different (e.g. lt . diestl instead of sergeant etc.) 2) the final screen is very ***flawed*** and makes diestl as a soldier who feds up himself and wants to die . but it is not true in 100 % . just read the book . he was a bull - dog in the last seconds as well . he did not want to die by wrecking his gun and walking simply towards to michael & noah . so this is some kind of a happy end which does not fit at all for this movie .

Table 5.11: Examples of original and adversarial sentences from IMDB (BERT)

Chapter 6

Discussion

TextDecepter crafts natural language adversaries against state-of-the-art text classification models for sentiment analysis in the hard-label black-box setting. In this chapter, we discuss a few important aspects of the attack algorithm:

6.1 Sentence Importance ranking

Sentence importance ranking is an important part of the attack algorithm as it locates those sentences which cast the most significant effect on the prediction of the target model. This helps the algorithm to reduce the number of perturbed words while crafting an adversarial example.

6.2 Aggregates

The original labeled aggregates generated during the sentence importance ranking step help the attack algorithm to select or reject the synonyms during the attack phase. An aggregate is a text which is smaller in length as compared to the original text and is classified by the target model to be belonging to the original class. Aggregates help the attack algorithm to either select or reject the synonyms during the attack phase. A word is replaced with a particular synonym only when its usage misclassifies the original text or the aggregate. Hence, the usage of aggregates gives us additional evidence before selecting or rejecting a synonym.

6.3 Error analysis

The adversarial samples generated using our attack framework are susceptible to error when named-entities are there in the text. If a word is a part of a named entity, the attack might still end up replacing it, as the current framework does not filter out such words. So, the sentence ”I watched

the movie 'A beautiful mind' " might get changed to "I watched the movie 'A resplendent mind' ". Although we can easily identify a named-entity in a grammatically correct text, real-world texts often have multiple grammatical errors in them which makes the identification difficult. In future work, it would be imperative to make a stricter criterion for replacement.

6.4 Generalization

The two fundamental natural language processing tasks are text classification and textual entailment. In our work, we have conducted experiments on binary classification problems pertaining to sentiment analysis. As a future line of work, it would be interesting to conduct the experiments on multi-class classification and textual entailment tasks.

Chapter 7

Conclusion

We propose a hard-label black-box attack strategy for text classification tasks. We also conduct extensive experimentation on sentiment analysis datasets to validate our attack system. We also conduct a human evaluation to validate the grammatical and semantic correctness of the generated adversarial examples. The attack algorithm uses the assumption of additivity of class intensities of sentences to craft adversarial examples and achieves an attack success rate of more than 50 % against state-of-the-art text classification models. The adversarial examples generated can be utilized to improve the existing text classification models for sentiment analysis by including them in the training dataset. The attack algorithm can also be used to gauge the robustness of text classification models pertaining to sentiment analysis. We also improved upon the memory requirement for the attack, as compared to the other comparative attack frameworks. As a future line of work, it would be imperative to run the attack algorithm on multi-class classification and textual entailment tasks to test the generalization capability of the proposed methodology.

References

[1] M. Alzantot, Y. Sharma, S. Chakraborty, H. Zhang, C.-J. Hsieh, and M. B. Srivastava, "Genattack: Practical black-box attacks with gradient-free optimization," in *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 2019, pp. 1111–1119.

[2] M. Alzantot, Y. Sharma, A. Elgohary, B.-J. Ho, M. Srivastava, and K.-W. Chang, "Generating natural language adversarial examples," *arXiv preprint arXiv:1804.07998*, 2018.

[3] H. S. Anderson, A. Kharkar, B. Filar, and P. Roth, "Evading machine learning malware detection," *black Hat*, 2017.

[4] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[5] W. Brendel, J. Rauber, and M. Bethge, "Decision-based adversarial attacks: Reliable attacks against black-box machine learning models," *arXiv preprint arXiv:1712.04248*, 2017.

[6] N. Carlini and D. Wagner, "Magnet and" efficient defenses against adversarial attacks" are not robust to adversarial examples," *arXiv preprint arXiv:1711.08478*, 2017.

[7] D. Cer, Y. Yang, S.-y. Kong, N. Hua, N. Limtiaco, R. S. John, N. Constant, M. Guajardo-Cespedes, S. Yuan, C. Tar *et al.*, "Universal sentence encoder," *arXiv preprint arXiv:1803.11175*, 2018.

[8] M. Cheng, T. Le, P.-Y. Chen, J. Yi, H. Zhang, and C.-J. Hsieh, "Query-efficient hard-label black-box attack: An optimization-based approach," *arXiv preprint arXiv:1807.04457*, 2018.

[9] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *Journal of machine learning research*, vol. 12, no. ARTICLE, pp. 2493–2537, 2011.

[10] S. Cresci, M. Petrocchi, A. Spognardi, and S. Tognazzi, "Better safe than sorry: An adversarial approach to improve social bot detection," *arXiv preprint arXiv:1904.05132*, 2019.

[11] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.

[12] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[13] J. Ebrahimi, A. Rao, D. Lowd, and D. Dou, "Hotflip: White-box adversarial examples for text classification," *arXiv preprint arXiv:1712.06751*, 2017.

[14] S. Feng, E. Wallace, A. Grissom II, M. Iyyer, P. Rodriguez, and J. Boyd-Graber, "Pathologies of neural models make interpretations difficult," *arXiv preprint arXiv:1804.07781*, 2018.

[15] J. Gao, J. Lanchantin, M. L. Soffa, and Y. Qi, "Black-box generation of adversarial text sequences to evade deep learning classifiers," in *2018 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2018, pp. 50–56.

[16] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[17] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE, 2013, pp. 6645–6649.

[18] C. Guo, J. R. Gardner, Y. You, A. G. Wilson, and K. Q. Weinberger, "Simple black-box adversarial attacks," *arXiv preprint arXiv:1905.07121*, 2019.

[19] F. Hill, R. Reichart, and A. Korhonen, "Simlex-999: Evaluating semantic models with (genuine) similarity estimation," *Computational Linguistics*, vol. 41, no. 4, pp. 665–695, 2015.

[20] S. Hochreiter, Y. Bengio, P. Frasconi, J. Schmidhuber *et al.*, "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies," 2001.

[21] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[22] D. Jin, Z. Jin, J. T. Zhou, and P. Szolovits, "Is bert really robust? natural language attack on text classification and entailment," *arXiv preprint arXiv:1907.11932*, 2019.

[23] N. Kalchbrenner, E. Grefenstette, and P. Blunsom, "A convolutional neural network for modelling sentences," *arXiv preprint arXiv:1404.2188*, 2014.

[24] Y. Kim, "Convolutional neural networks for sentence classification," *arXiv preprint arXiv:1408.5882*, 2014.

[25] B. Kolosnjaji, A. Demontis, B. Biggio, D. Maiorca, G. Giacinto, C. Eckert, and F. Roli, "Adversarial malware binaries: Evading deep learning for malware detection in executables," in *2018 26th European Signal Processing Conference (EUSIPCO)*. IEEE, 2018, pp. 533–537.

[26] B. Kulynych, J. Hayes, N. Samarin, and C. Troncoso, "Evading classifiers in discrete domains with provable optimality guarantees," *arXiv preprint arXiv:1810.10939*, 2018.

[27] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial machine learning at scale," *arXiv preprint arXiv:1611.01236*, 2016.

[28] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[29] J. Li, S. Ji, T. Du, B. Li, and T. Wang, "Textbugger: Generating adversarial text against real-world applications," *arXiv preprint arXiv:1812.05271*, 2018.

[30] Y. Liu, X. Chen, C. Liu, and D. Song, "Delving into transferable adversarial examples and black-box attacks," *arXiv preprint arXiv:1611.02770*, 2016.

[31] W. Medhat, A. Hassan, and H. Korashy, "Sentiment analysis algorithms and applications: A survey," *Ain Shams engineering journal*, vol. 5, no. 4, pp. 1093–1113, 2014.

[32] T. Mikolov *et al.*, "Statistical language models based on neural networks," *Presentation at Google, Mountain View, 2nd April*, vol. 80, p. 26, 2012.

[33] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: a simple and accurate method to fool deep neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2574–2582.

[34] N. Mrkšić, D. O. Séaghdha, B. Thomson, M. Gašić, L. Rojas-Barahona, P.-H. Su, D. Vandyke, T.-H. Wen, and S. Young, "Counter-fitting word vectors to linguistic constraints," *arXiv preprint arXiv:1603.00892*, 2016.

[35] Y. Nesterov and V. Spokoiny, "Random gradient-free minimization of convex functions," *Foundations of Computational Mathematics*, vol. 17, no. 2, pp. 527–566, 2017.

[36] C. Nobata, J. Tetreault, A. Thomas, Y. Mehdad, and Y. Chang, "Abusive language detection in online user content," in *Proceedings of the 25th international conference on world wide web*, 2016, pp. 145–153.

[37] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *Proceedings of the 2017 ACM on Asia conference on computer and communications security*. ACM, 2017, pp. 506–519.

[38] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2016, pp. 372–387.

[39] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.

[40] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, and C. Nicholas, "Malware detection by eating a whole exe (2017)," *arXiv preprint arXiv:1710.09435*, 2017.

[41] G. Rawlinson, "The significance of letter position in word recognition," *IEEE Aerospace and Electronic Systems Magazine*, vol. 22, no. 1, pp. 26–27, 2007.

[42] Y. Shen, X. He, J. Gao, L. Deng, and G. Mesnil, "A latent semantic model with convolutional-pooling structure for information retrieval," in *Proceedings of the 23rd ACM international conference on conference on information and knowledge management*, 2014, pp. 101–110.

[43] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[44] H. Wu, "Global stability analysis of a general class of discontinuous neural networks with linear growth activation functions," *Information Sciences*, vol. 179, no. 19, pp. 3432–3441, 2009.

[45] W.-t. Yih, X. He, and C. Meek, "Semantic parsing for single-relation question answering," in *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, 2014, pp. 643–648.

[46] X. Zhang, J. Zhao, and Y. LeCun, "Character-level convolutional networks for text classification," in *Advances in neural information processing systems*, 2015, pp. 649–657.