Token-Modification Adversarial Attacks for Natural Language Processing: A Survey

Tom Roth[1,2] , **Yansong Gao**[2] , **Alsharif Abuadbba**[2] , **Surya Nepal**[2] and **Wei Liu**[1]

[1]University of Technology Sydney, Australia
[2]CSIRO's Data61, Sydney, Australia

{thomas.roth, wei.liu}@uts.edu.au, {garrison.gao, sharif.abuadbba, surya.nepal}@data61.csiro.au

Abstract

There are now many adversarial attacks for natural language processing systems. Of these, a vast majority achieve success by modifying individual document tokens, which we call here a *token-modification* attack. Each token-modification attack is defined by a specific combination of fundamental *components*, such as a constraint on the adversary or a particular search algorithm. Motivated by this observation, we survey existing token-modification attacks and extract the components of each. We use an attack-independent framework to structure our survey which results in an effective categorisation of the field and an easy comparison of components. We hope this survey will guide new researchers to this field and spark further research into the individual attack components.

1 Introduction

Adversarial machine learning investigates how a model can be exploited by an adversary through a carefully crafted *adversarial attack* [Kurakin *et al.*, 2017]. The majority of adversarial machine learning research has been for image processing, but there has been increasing interest in exploring the natural language processing (NLP) domain. Motivated by this recent growth, we survey the recent work in this area and recommend several research directions.

An *adversarial example* is an example created by adding a perturbation to an *original example* so that a model processes the original as expected but fails on the adversarial in some prespecified manner. As our running example, [Wallace *et al.*, 2020] wish to perturb a document that reads *"Save me it's over 100°F"* in order to introduce a specific token *22°C* into the output of an English-to-German translation model. Their attack algorithm creates the adversarial example by replacing *"100°F"* with *"102°F"*. While the model translates the original example correctly as *"Rette mich, es ist über 100°F"*, it now translates the adversarial example as *"Rette mich, es ist über 22°C"*, inadvertently introducing the specific token.

Usually text adversarial examples must also meet a set of constraints. For example, one constraint might specify that the
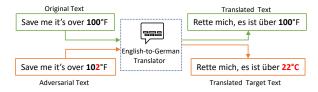

Figure 1: Example of a token-modification adversarial attack on a machine translation model.

semantic meaning of the original example is maintained. In this case, *"Save me it's above 100°F"* would be valid, but not the proposed *"Save me it's over 102°F"*. A second constraint might require that the adversarial example appear non-suspicious to a human. This rules out *"Save ffffff it's qqqqq over 102°F"*, but our example *"Save me it's over 102°F"* would pass.

The most studied type of text adversarial attacks are **token-modification** attacks: those modify individual document tokens of some granularity (e.g. words) to achieve attack success. Our running example is a token-modification attack. In these attacks, the adversary does not change the rough syntactic structure of the document. Rather, they treat a document as an ordered set of tokens, and repeatedly apply *transformations* to the individual tokens, such as replacing a word with its synonym, deleting a character, or inserting a phrase into the document. The attack then stops once the adversary reaches its goal, or alternately, if some stopping criteria is reached.

Previous surveys—the two most comprehensive being [Zhang *et al.*, 2020] and [Alshemali and Kalita, 2020]—categorise text attacks by token granularity, adversary goal, model access, NLP task attacked, and so on. This approach over-emphasises the differences between attacks. Many attack algorithms only differ from each other in their use of one component (e.g. the attacks proposed by [Zhang *et al.*, 2019]), which means algorithms can change their categorisation with only trivial modifications. We solve this problem by categorising individual attack components rather than the attacks themselves. We are the first survey to take this approach.

	Category	Papers
Adv. Goals	Classification	[Ren *et al.*, 2019; Jin *et al.*, 2019; Li *et al.*, 2020b; Li *et al.*, 2019] [Gao *et al.*, 2018; Eger *et al.*, 2019; Zang *et al.*, 2020; Alzantot *et al.*, 2020] [Zhang *et al.*, 2019; Liang *et al.*, 2018; Hsieh *et al.*, 2019; Xu and Du, 2020] [Wang *et al.*, 2019; Samanta and Mehta, 2017; Garg and Ramakrishnan, 2020]
	Seq2Seq	[Hsieh *et al.*, 2019; Michel *et al.*, 2019; Camburu *et al.*, 2020] [Jia and Liang, 2017; Ebrahimi *et al.*, 2018a; Belinkov and Bisk, 2018] [Cheng *et al.*, 2020; Wallace *et al.*, 2020; Papernot *et al.*, 2016; Zhao *et al.*, 2018]
Transformations	Any replacement	[Ebrahimi *et al.*, 2018b; Ebrahimi *et al.*, 2018a; Gao *et al.*, 2018; Cheng *et al.*, 2020] [Hsieh *et al.*, 2019; Papernot *et al.*, 2016; Wallace *et al.*, 2020]
	Human errors	[Belinkov and Bisk, 2018; Li *et al.*, 2019; Hsieh *et al.*, 2019]
	Visually similar	[Eger *et al.*, 2019; Li *et al.*, 2019]
	Synonym list	[Ren *et al.*, 2019; Zang *et al.*, 2020; Tan *et al.*, 2020; Samanta and Mehta, 2017]
	Embedding space	[Li *et al.*, 2019; Xu and Du, 2020; Alzantot *et al.*, 2020] [Jin *et al.*, 2019; Wang *et al.*, 2019; Li *et al.*, 2020b] [Zhang *et al.*, 2019; Hsieh *et al.*, 2019; Michel *et al.*, 2019]
	Language model predictions	[Alzantot *et al.*, 2020; Zhang *et al.*, 2019; Li *et al.*, 2020b] [Garg and Ramakrishnan, 2020; Zhang *et al.*, 2019]
Search Methods	All at once	[Tan *et al.*, 2020; Ebrahimi *et al.*, 2018a; Belinkov and Bisk, 2018; Eger *et al.*, 2019]
	Document order	[Zhang *et al.*, 2019]
	Random order	[Yoo *et al.*, 2020; Hsieh *et al.*, 2019]
	Importance ranking order	[Li *et al.*, 2019; Xu and Du, 2020; Li *et al.*, 2020b; Samanta and Mehta, 2017] [Jin *et al.*, 2019; Gao *et al.*, 2018; Garg and Ramakrishnan, 2020; Ren *et al.*, 2019] [Hsieh *et al.*, 2019; Ebrahimi *et al.*, 2018b; Ebrahimi *et al.*, 2018a]
	Beam/greedy search	[Ebrahimi *et al.*, 2018b; Wallace *et al.*, 2020; Ebrahimi *et al.*, 2018a; Michel *et al.*, 2019]
	Genetic algorithms	[Alzantot *et al.*, 2020; Wang *et al.*, 2019]
	Particle swarm opt.	[Zang *et al.*, 2020]
Constraints	Readability	[Belinkov and Bisk, 2018]
	Semantic	[Zang *et al.*, 2020; Cheng *et al.*, 2020; Ebrahimi *et al.*, 2018b; Wang *et al.*, 2019] [Jin *et al.*, 2019; Xu and Du, 2020; Tan *et al.*, 2020; Li *et al.*, 2020b] [Garg and Ramakrishnan, 2020; Zhang *et al.*, 2019; Alzantot *et al.*, 2020]
	Distance	[Xu and Du, 2020; Ebrahimi *et al.*, 2018b; Ebrahimi *et al.*, 2018a] [Gao *et al.*, 2018; Li *et al.*, 2019; Hsieh *et al.*, 2019]
	Performance	[Xu and Du, 2020; Ebrahimi *et al.*, 2018b; Ren *et al.*, 2019; Samanta and Mehta, 2017]

Table 1: Our proposed categorization of fundamental components of token-modification attacks.

2 Attack Framework

Recently [Morris *et al.*, 2020a] proposed a framework that defines four components to a textual adversarial attack: (a) a goal function that specifies if the attack was successful; (b) a set of transformations that define the precise actions available to the adversary; (c) a search method that specifies the order transformations are applied; and (d) a set of constraints to define a valid adversarial example. Each text attack becomes a constrained search problems defined by its choice of components.

[Morris *et al.*, 2020a] use their framework to write a software library to quickly implement text adversarial attacks. Motivated by this work, we extend this framework and systematically survey the range of attack components. Due to space requirements, we restrict our survey to token-modification attacks—since they are the predominant attack type—but our framework is broadly applicable to other attack classes. Our taxonomy and covered attacks are summarized in Table 1.

Notation. Let Ex be the original example with label y_{true}. All examples are documents, and each document is made up of a set of tokens $\{t_1, \cdots, t_n\}$, for which a token t_i has the numeric representation e_{t_i} (e.g. word embeddings). The document Ex is transformed to a numeric representation x and then acted on by a model f. Likewise, the adversarial example is Ex' (also with label y_{true}), its numeric representation is x', and we denote its output $f(x')$ by y'. We use l to represent a generic loss function that acts on a single example.

3 Adversary Goals

The adversary goal describes what the adversary intends to achieve. They are specific to a task so we categorise by this below.

3.1 Classification

Attacks on text classifiers are usually either **targeted** or **untargeted** attacks. Targeted attacks attempt to induce the model to classify Ex' as a specific class y_{target} different to y_{true}. They do this by minimising $l(f(x), y_{target})$. Untargeted attacks only require that Ex is misclassified, and their algorithms typically aim to maximise $l(f(x), y_{true})$. The two are equivalent for binary classification, and for mulitclass classification, converting between the two is usually straightforward [1].

3.2 Sequence-to-Sequence

There are a wide range of sequence-to-sequence (seq2seq) tasks in NLP, including machine translation, dialog systems, text summarisation, paraphrase generation, speech recognition, speech synthesis and image generation from captions. Unlike classification, there are no standardised adversary goals for seq2seq attacks, so we borrow terminology and broadly classify these as either untargeted or targeted.

Untargeted attacks. These attacks simply attempt to degrade the output sequence quality by as much as possible, as quantified by some metric. This is the most common goal for seq2seq attacks [Hsieh *et al.*, 2019; Michel *et al.*, 2019; Camburu *et al.*, 2020; Jia and Liang, 2017; Ebrahimi *et al.*, 2018a].

Targeted attacks. These attacks attempt to induce specific changes in the output sequence. They include: (a) changing a specific output token to any other token [Papernot *et al.*, 2016; Ebrahimi *et al.*, 2018a; Zhao *et al.*, 2018], (b) changing every output token from its original value [Cheng *et al.*, 2020], (c) changing a set of specific output tokens to another set of specific tokens [Wallace *et al.*, 2020; Ebrahimi *et al.*, 2018a], (d) introducing a set of specific output tokens anywhere in the output sequence [Hsieh *et al.*, 2019; Cheng *et al.*, 2020; Zhao *et al.*, 2018], or (e) finding an incorrect input sequence that generates a specific output sequence [Wallace *et al.*, 2020]. These attacks all involve carefully defining an attack-dependent loss function and subsequently maximising or minimising it.

4 Transformations

A transformation is a function that transforms one ordered set of tokens to another. We can place these into broad categories, which we call here **transformation types**. There are four transformation types commonly used by token-modification attacks: replacing one token with another, inserting a token, deleting a token, or changing token ordering. Replacing a token is by far the most popular transformation type, so we give it special focus below.

When replacing a token t with a replacement t_r, the attack algorithm will select t_r from a **set of replacements**. We denote this set by $\mathbb{S}(t)$, parameterised to indicate its dependence on t. Specifying a method to construct $\mathbb{S}(t)$ is a core component

---

[1] For example, instead of perturbing x towards a *specific* decision boundary, a untargeted attack perturbs x towards the *nearest* decision boundary.

of many attack algorithms but identifying the best method remains an open problem. The standard approach is to first generate an initial set of candidates and then to apply constraints to filter the set. For example, an algorithm could use a dictionary to generate a set of synonyms and then filter out any using a different part-of-speech (POS) tag to t.

Below we describe different methods that have been used to generate the initial set of candidates, while deferring filtering constraints to Section 6. Attacks are free to use a combination of these methods.

4.1 Any Replacement

The adversary replaces t with any token they like of the same granularity. This is both the simplest approach and the most effective at achieving the adversary's goal. Yet Ex' is very likely to be ungrammatical and to lose its semantic meaning, thus this approach is usually not followed. It is most often seen with character-level attacks that allow any alphanumeric character (upper or lower case) or punctuation as a valid replacement [Ebrahimi *et al.*, 2018b; Ebrahimi *et al.*, 2018a; Gao *et al.*, 2018]. These attacks make no attempt to maintain semantics and focus instead on making the perturbation as small as possible. It is also sometimes seen when adversarial attacks for images are applied naively to the text embeddings (see [Liang *et al.*, 2018] for an example).

4.2 Human Errors

The adversary replaces t with a token that resembles an innocent human-made mistake, such as a typo. This is intended to disguise Ex' as a poorly-written document so that it is not considered suspicious by a human. One method is to replace letters with adjacent characters on a QWERTY keyboard, as to mimic a keyboard typo [Belinkov and Bisk, 2018; Li *et al.*, 2019]. Another is to replace a word with a commonly-misspelled variant (e.g. *achieve* with *acheive*). Common human misspellings are easy to find: there are resources (e.g. Wikipedia) that list them, or alternatively, [Belinkov and Bisk, 2018] extract their list from corpora tracking the edit history of documents.

4.3 Visually Similar

The adversary replaces t with a token that looks similar (e.g. replacing *I* with *l*) with the intention that a human overlooks the error. Again, usually here t is a character. This can also include many-to-one or one-to-many character replacements (e.g. replacing *cl* with *d*). These attacks tend to be either almost indistinguishable or very obvious, depending on the font used and the choice of replacement.

The simplest method is to create a mapping between each token in the vocabulary and a manually-chosen replacement for each [Eger *et al.*, 2019; Li *et al.*, 2019]. For example, [Eger *et al.*, 2019] replace characters by a version of themselves with a diacritic (e.g. *c* to *č*). This work also proposes a different method that uses Unicode descriptions of characters to find replacements of the same character and case[2]. An alterna-

tive is to use visual embeddings that treat each character as a grid of pixels. Each character is then represented numerically, such as with a raw pixel vector [Eger *et al.*, 2019], or as a dense representation, like from the intermediate layers of a CNN [Li *et al.*, 2020a]. At this point $\mathbb{S}(t)$ can be constructed by taking the closest points to the character in that latent space.

4.4 Synonym List

The adversary replaces t with a replacement t_r taken from a predefined list of tokens that have the same meaning. It is most frequently used with word-level attacks. [Tan *et al.*, 2020] use a set of inflectional perturbations of words which all have the same base meaning. More common is using is a set of synonyms from a thesaurus, lexical database, knowledge database or similar. Examples include [Ren *et al.*, 2019] who use WordNet [Fellbaum, 1998] as their lexical database, and [Zang *et al.*, 2020] which use the knowledge database HowNet [Dong *et al.*, 2010]. This method is often unable to distinguish the sense in which a word is used, so replaced words often read strangely or do not fit well in the context of Ex'. Sometimes an attack will add constraints to try to mitigate this problem (e.g. [Zang *et al.*, 2020]) but in practice the constraints do not solve the problem very well.

4.5 Embedding Space

The adversary replaces token t with a token t_r corresponding to a nearby point in a token embedding space. The adversary assumes that since tokens co-occurring in similar contexts cluster together in embedding space, these tokens also have similar meaning, and hence the nearby points around t form a natural set of synonyms. This is most commonly used with word embeddings. [Li *et al.*, 2019; Xu and Du, 2020] both do this with GloVe [Pennington *et al.*, 2014] embeddings.

Yet in practice, close points in a word embedding space (e.g. GloVe [Pennington *et al.*, 2014]) include similar words (like word synonyms), but also related words, such as antonyms (e.g. *east* and *west*, or *fast* and *slow*) or words with similar concepts but different meanings (e.g. *English*, *American*, and *Australian*). Hence this approach usually leads to transformations that change the semantic content of Ex. This is mitigated by instead using *counter-fitted* embeddings [Mrkšić *et al.*, 2016]. Counter-fitting is a fast post-processing step for word embeddings that pushes together similar words while also pushing apart related words. The result is a word embedding where only synonyms are close to each other. Hence better results are obtained if the adversary uses a counter-fitted word embedding space. [Alzantot *et al.*, 2020; Jin *et al.*, 2019; Wang *et al.*, 2019; Li *et al.*, 2020b] all use counter-fitted GloVe embeddings.

What makes a nearby token? Common methods include taking the k-nearest neighbours (KNN) to the original token (e.g. [Li *et al.*, 2019; Xu and Du, 2020], or to consider tokens within a distance[3] δ to t (e.g. [Zhang *et al.*, 2019]), or to do both and take the intersection of the sets (e.g. [Alzantot

---

[2]For example, *a* has Unicode description *LATIN SMALL LETTER A*, and a potential replacement *à* has Unicode description *LATIN SMALL LETTER A WITH GRAVE*

[3]Usually this is Euclidean distance or cosine distance/similarity.

et al., 2020; Jin *et al.*, 2019]). [Hsieh *et al.*, 2019] does this too, but with document embeddings instead of word embeddings. [Wang *et al.*, 2019] create synonym clusters in word-embedding space (using their own clustering algorithm) intending that any word in the cluster can replace a "central" word, although their examples indicate problems with this approach.

A problem with this class of approaches is that token embeddings can only use a single point per token, but many tokens are used in a range of contexts. This means replacements are often for the wrong sense of a token.

4.6 Language Model Predictions

The adversary uses a language model LM to find replacements for t. The usual way is through *masked language modelling*: replace t in Ex with a mask token, get predictions from LM for the mask token, and use the top k predictions as $\mathbb{S}(t)$. [Alzantot *et al.*, 2020; Zhang *et al.*, 2019; Li *et al.*, 2020b; Garg and Ramakrishnan, 2020] variations on this approach. [Alzantot *et al.*, 2020; Zhang *et al.*, 2019] use both language model predictions and word embeddings to rank replacements.

This approach has two main advantages. Firstly, generated replacements are higher quality since the language model accounts for token context. Secondly, rare tokens are seldom suggested, which is advantageous since they tend to appear suspicious [Ippolito *et al.*, 2020]. There are disadvantages too: the attack requires a complex and memory-intensive language model, the language model is slower to generate replacements, and the attack cannot cache synonyms between documents (since they depend on document context). Nonetheless, the quality increase in replacements is usually high enough that the advantages outweigh the disadvantages.

5 Search Methods

The search method attempts to find a successful sequence of transformations that solves the constrained search problem. Search methods rank transformations with a **scoring function**, a function estimating the effect of the transformation on the loss function.

The simplest scoring function directly calculates the loss increase of the transformation. This measures the impact exactly, however the approach is slow—particularly for large models. Hence a popular alternative is to use a *gradient-based* scoring function that uses some variant of $\partial l / \partial \boldsymbol{x}$: a first-order approximation of the effect on l when changing elements of $\boldsymbol{x}$.[4] Since the gradient approximates the direction of steepest loss increase, transformations can be ranked by their dot product $\partial l / \partial \boldsymbol{x} \cdot (g(\boldsymbol{x}) - \boldsymbol{x})$, where the vector $g(\boldsymbol{x}) - \boldsymbol{x}$ measures the relative change in input-space of a transformation g. Both the direction and the magnitude of

this vector are important, so dot product is a better metric to use than cosine similarity.

The gradient approximation only needs a single backwards pass through the model, but requires the adversary can access model parameters (*white-box* attack). It also has two implicit assumptions: that l is close to linear around $\boldsymbol{x}$, and that all other features are fixed when evaluating the importance of a single feature [Singla *et al.*, 2019]. In most cases probably neither assumption holds, but regardless the search method usually can find a successful adversarial example.

[Wallace *et al.*, 2020] use a hybrid approach which has advantages of both approaches: the speed of the gradient approximation, and the accuracy of direct-checking. They first use the gradient approximation to create a shortlist of the top k transformations across all tokens in Ex. They then find the best transformation by directly checking the loss change of each.

Below we present common search algorithms. Search methods either stop when the adversary goal is reached or when the constraints prevent them from searching further.

5.1 All at Once

Rather than apply transformations one-by-one, this method creates Ex' by simultaneously applying a transformation to all applicable tokens in Ex. This is done in [Tan *et al.*, 2020; Ebrahimi *et al.*, 2018a], who both replace each applicable word $w \in Ex$. [Belinkov and Bisk, 2018] allow for multiple runs of this procedure, with each run selecting a different transformation. [Eger *et al.*, 2019] implement a variation where they replace each token with a probability p.

5.2 Document Order

This approach transforms tokens in document order. Attacks choose the best transformation for the first applicable token t_1 in Ex, then for t_2, and so on until the attack is successful. The algorithm can cycle through the document until it is successful. [Zhang *et al.*, 2019] take this approach, but rather than select the best transformation, they have predefined probabilities for several different transformation types.[5]

5.3 Random Order

This search method randomly selects and replaces a token from the document. This method is used as a baseline by [Yoo *et al.*, 2020; Hsieh *et al.*, 2019], where tokens are either replaced by the best replacement or by a random replacement.

5.4 Importance Ranking Order

This method ranks the influence of each token on the loss function and then applies transformations in that order. This keeps the perturbation small since more influential tokens are transformed first. [Yoo *et al.*, 2020] call this an *importance ranking*.

---

[4]Some attacks (e.g. [Papernot *et al.*, 2016]) instead use the Jacobian matrix $\partial f(x) / \partial \boldsymbol{x}$ and directly change model predictions, but the loss-function approach is predominant.

[5]For each word w, they either replace it with a word from $\mathbb{S}(w)$, delete it, or insert a word in front of it, with predefined probabilities 0.5, 0.25, and 0.25 respectively.

There are several common methods to calculate an importance ranking. The first uses *gradient information*. Intuitively, elements of x with a high gradient magnitude $\partial l/\partial x$ affect l more, with the direction depending on the sign of the gradient. The quantity $\partial l/\partial x$ cannot be used directly to rank tokens since the length of x is a multiple of the number of tokens. The common workaround is to calculate importance of a token t by the magnitude of its loss gradient vector $||\partial l/\partial e_{t_i}||_2$. The drawback is that using only gradient magnitude and not direction means the transformation impact cannot be predicted precisely. Nevertheless, this remains a sound heuristic and many attacks use this method [Li *et al.*, 2019; Xu and Du, 2020; Samanta and Mehta, 2017].

Another method is to systematically perturb each token in the document and measure the change in loss. There are a number of different perturbation methods. One is to remove the token entirely [Jin *et al.*, 2019; Garg and Ramakrishnan, 2020; Li *et al.*, 2019]. Another is to replace it with a "neutral" token that theoretically does not affect model predictions. The neutral token has been the out-of-vocab/unknown token [Ren *et al.*, 2019], the zero vector [Ebrahimi *et al.*, 2018b], and the mask token [Li *et al.*, 2020b]. Another two methods were proposed by [Gao *et al.*, 2018]. They first split the document into a *prefix*, which contains the document up to (but not including) the token, and a *suffix*, which contains the document from after the token until the end of the document. The two metrics are then the *temporal head score*, given by $J(f(\text{prefix}) - J(f(\text{prefix} + \text{token}))$ and the *temporal tail score*, given by $J(f(\text{token} + \text{suffix}) - J(f(\text{suffix}))$.

A third method calculates the importance ranking with attention scores. This method only works if the victim model has an attention mechanism. [Hsieh *et al.*, 2019] use the attention scores of the first encoder-decoder stack, or if the model has multiple heads, the average of the attention layer over each head. The best importance ranking ranked scores from highest to lowest.

How do these methods compare? [Yoo *et al.*, 2020] reported similar performance testing these methods[6], though the gradient approach was more computationally efficient[7].

5.5 Beam Search and Greedy Search

Beam search is a search algorithm that keeps track of multiple promising paths at once. The *beam width* parameter controls the number of paths. Larger beam widths find better solutions, but take more time to evaluate. Setting a beam width of one gives *greedy search*, where the best transformation is selected at each step.

Beam search has been used as the search method in [Ebrahimi *et al.*, 2018b; Ebrahimi *et al.*, 2018a], and greedy search in [Michel *et al.*, 2019; Wallace *et al.*, 2020]. Beam search has been reported to have a high success rate while also being

---

[6]They did not test the attention method.

[7]Systematic perturbation methods need many forward passes through the model, but gradient methods only need single forwards and backward pass.

relatively query-efficient for shorter documents [Yoo *et al.*, 2020].

5.6 Genetic Algorithms

Genetic algorithms solve optimisation problems by evolving a population of candidate solutions over time, using a fitness function to perform "natural selection" until one is successful. Variants have been applied to text attacks by [Alzantot *et al.*, 2020; Wang *et al.*, 2019]. The algorithm starts by applying S different transformations to Ex to create a candidate population of size S. Then at each step a new candidate set is created by: (a) applying transformations to the original S candidates; (b) calculating the loss for each, which is called the "fitness" score; (c) sampling S pairs of these documents with probability proportional to their fitness score; (d) creating S new candidates through applying a *crossover* function to the two parents. The crossover function in [Alzantot *et al.*, 2020] simply selects each word randomly from one of the two parent documents, while [Wang *et al.*, 2019] instead split each parent document at a point and take one fragment from each.

5.7 Particle Swarm Optimisation

Particle swarm optimisation is a search method with both continuous and discrete variants. It starts by initialising a number of "particles". Each particle has a position and a velocity, and keeps track of both its best position and the best position achieved across all particles. Particles move with a random component and a probability to move towards either the global best or local best, depending on the algorithm variant.

[Zang *et al.*, 2020] use particle swarm optimisation as their search method. [Yoo *et al.*, 2020] reported that this search method reliably leads to a good solution, but is query inefficient.

6 Constraints

Attacks can consider a set of constraints. Weaker constraints increase attack success, but also find an inferior adversarial example. Constraints can have many benefits on Ex': reducing semantic drift from Ex, increasing readability or naturalness, keeping Ex and Ex' "close" together, ensuring Ex' has correct grammar, and so on.

We group constraints into rough categories based on their intended effect.

6.1 Readability

These constraints intend to increase human readability of Ex'. Some are only applicable for character-level attacks. One of these is to apply a spellchecker to Ex' and see if the result still achieves the adversary's goal. [Belinkov and Bisk, 2018] use other constraints. They only perform character transformations on words of a certain length, and avoid changing the first and last letters of a word.

More generally, adversarial attacks might require Ex' have correct grammar, as checked by grammar software. Sometimes this constraint is not appropriate: for example, the at-

tack by [Tan *et al.*, 2020] deliberately mimics non-native English speakers, so the constraint would not be used then. Another constraint requires adversarial examples maintain some number of linguistically acceptable sentences [Morris *et al.*, 2020a]. This can be measured with a model trained on a linguistic acceptability dataset.

6.2 Semantic

These constraints aim to maintain the semantic content of the original document. A widely used constraint only allows one transformation on each token (e.g. [Cheng *et al.*, 2020; Ebrahimi *et al.*, 2018b]), or more generally, up to k transformations per token [Wang *et al.*, 2019]). Avoiding multiple transformations on a token eliminates a likely source of semantic drift. Another type of constraint specifies the tokens a transformation can act on. Here the adversary wishes to avoid changing non-influential tokens that are also vital for document structure. Sometimes these tokens are approximated by a list of stop words, and hence one version of this constraint explicitly forbids replacing stop words [Jin *et al.*, 2019; Xu and Du, 2020]. This is sometimes reformulated as only allowing replacements for certain word types. For example, [Tan *et al.*, 2020] only allows nouns, verbs and adjectives to be replaced, while [Zang *et al.*, 2020] also allows replacing adverbs. A variant used by [Li *et al.*, 2020b] instead disallows stop words in the set of replacements.

Other constraints filter the set of replacements to improve their quality, where quality is judged as similarity to the original token, or the extent to which the document reads naturally. One method requires replacements to have the same part of speech tag as the original [Jin *et al.*, 2019; Zang *et al.*, 2020; Xu and Du, 2020]. This helps to eliminate potential synonyms that use an incorrect word sense. Another measures entropy of a language model for each replacement when it is put into context, and only considers replacements above a threshold [Alzantot *et al.*, 2020].

Sometimes the distance between original and adversarial document encodings[8] is used to estimate semantic similarity. At each attack step, the adversary finds document encodings of Ex and Ex', and then computes the cosine similarity of the two. Then if the cosine similarity is below a threshold, the adversary avoids transforming the token [Jin *et al.*, 2019], or alternatively, they can rank different transformations by cosine similarity[Garg and Ramakrishnan, 2020]. A commonly used document encoder for this purpose is the Universal Sentence Encoder [Cer *et al.*, 2018].

An adversary might also apply task-specific constraints to Ex' to maintain semantic similarity. For example, for sentiment analysis [Zhang *et al.*, 2019] do not replace any words that they determine express sentiment. They also avoids replacing negation words.

6.3 Distance

These constraints constrain the distance between Ex and Ex', where the distance metric is usually Hamming or Levenshtein edit distance[9]. Hamming distance measures the minimum number of token replacements required to go from Ex to Ex'. Levenshtein distance is similar but also allows inserting and deleting tokens. Several attacks use an edit distance constraint [Ebrahimi *et al.*, 2018a; Gao *et al.*, 2018; Li *et al.*, 2019].

6.4 Performance

These constraints attempt to improve attack performance They include: adding the padding and unknown tokens to the synonym set [Xu and Du, 2020], lemmatisation/delemmatisation of candidate synonyms to create more potential replacements [Zang *et al.*, 2020], or only allowing character transformations if the transformed word does not exist in the vocabulary[10] [Ebrahimi *et al.*, 2018b].

Some attacks prioritise transformations that introduce tokens over-represented in other classes to y_{true}. The reasoning is that these transformations are likely to be effective while also appearing natural to humans. One example is [Ren *et al.*, 2019] who replace named entities with the most frequent same-type named entities from a different class. [Samanta and Mehta, 2017] first assign each example a category (e.g. movie genre for a dataset of movie reviews), and then prioritise word replacements that are both used frequently in that category and also in different-class examples.

7 Conclusions and Future Directions

Token-modification attacks are simple permutations of our presented attack components. We encourage further research into these components, such as developing new scoring functions, finding better sets of replacements, or identifying better search algorithms. We also make two further suggestions.

Improve semantic similarity constraints. As it stands semantic similarity is best evaluated by a human [Morris *et al.*, 2020b] since current automated semantic similarity metrics are not yet close to human judgement. Improving these automated metrics will in turn increase the text quality of attacks substantially.

Extend to multimodal systems. The majority of adversarial attacks attack systems of a single modality, such as images or text. Yet real-world data is often multimodal, consisting of images, videos, and text (e.g., web pages, emails, social media). Extending text attacks to use multimodal information is another promising future direction.

---

[8]Document encodings are numeric representations of documents, similar to how word embeddings are numeric representations of words.

[9]Other metrics include n-gram based metrics (e.g. BLEU [Papineni *et al.*, 2002]) and Word Mover Distance (as proposed by [Zhang *et al.*, 2020]) but neither one has been widely used.

[10]This is not intended to maintain readability, but rather to ensure word-level models represent the replacement tokens with out-of-vocab tokens, rather than with another word.

References

[Alshemali and Kalita, 2020] B. Alshemali and J. Kalita. Improving the Reliability of Deep Neural Networks in NLP: A Review. *Knowledge-Based Systems*, 191:105210, 2020.

[Alzantot *et al.*, 2020] M. Alzantot, Y. Sharma, A. Elgohary, B. J. Ho, M. B. Srivastava, and K. W. Chang. Generating Natural Language Adversarial Examples. *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, EMNLP 2018*, pages 2890–2896, 2020.

[Belinkov and Bisk, 2018] Y. Belinkov and Y. Bisk. Synthetic and Natural Noise Both Break Neural Machine Translation. In *6th International Conference on Learning Representations, {ICLR} 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

[Camburu *et al.*, 2020] O.-M. Camburu, B. Shillingford, P. Minervini, T. Lukasiewicz, and P. Blunsom. Make Up Your Mind! Adversarial Generation of Inconsistent Natural Language Explanations. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4157–4165, Online, 7 2020. Association for Computational Linguistics.

[Cer *et al.*, 2018] D. Cer, Y. Yang, S. y. Kong, N. Hua, N. Limtiaco, R. St. John, N. Constant, M. Guajardo-Céspedes, S. Yuan, C. Tar, Y. H. Sung, B. Strope, and R. Kurzweil. Universal sentence encoder for English. *EMNLP 2018 - Conference on Empirical Methods in Natural Language Processing: System Demonstrations, Proceedings*, abs/1803.1:169–174, 2018.

[Cheng *et al.*, 2020] M. Cheng, J. Yi, P.-Y. Chen, H. Zhang, and C.-J. Hsieh. Seq2Sick: Evaluating the Robustness of Sequence-to-Sequence Models with Adversarial Examples. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04):3601–3608, 2020.

[Dong *et al.*, 2010] Z. Dong, Q. Dong, and C. Hao. HowNet and Its Computation of Meaning. In *Coling 2010: Demonstrations*, pages 53–56, Beijing, China, 8 2010. Coling 2010 Organizing Committee.

[Ebrahimi *et al.*, 2018a] J. Ebrahimi, D. Lowd, and D. Dou. On Adversarial Examples for Character-Level Neural Machine Translation. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 653–663, Santa Fe, New Mexico, USA, 8 2018. Association for Computational Linguistics.

[Ebrahimi *et al.*, 2018b] J. Ebrahimi, A. Rao, D. Lowd, and D. Dou. HotFlip: White-Box Adversarial Examples for Text Classification. In I. Gurevych and Y. Miyao, editors, *ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 2: Short Papers*, pages 31–36. Association for Computational Linguistics, 2018.

[Eger *et al.*, 2019] S. Eger, G. G. İşgüder, A. Rücklé, J.-U. Lee, C. Schulz, M. Mesgar, K. Swarnkar, E. Simpson, and I. Gurevych. Text Processing Like Humans Do: Visually Attacking and Shielding NLP Systems. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 1634–1647, Minneapolis, Minnesota, 6 2019. Association for Computational Linguistics.

[Fellbaum, 1998] C. Fellbaum. *WordNet: An Electronic Lexical Database*. Bradford Books, 1998.

[Gao *et al.*, 2018] J. Gao, J. Lanchantin, M. L. Soffa, and Y. Qi. Black-Box Generation of Adversarial Text Sequences to Evade Deep Learning Classifiers. *Proceedings - 2018 IEEE Symposium on Security and Privacy Workshops, SPW 2018*, pages 50–56, 2018.

[Garg and Ramakrishnan, 2020] S. Garg and G. Ramakrishnan. BAE: BERT-based Adversarial Examples for Text Classification. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6174–6181, Online, 11 2020. Association for Computational Linguistics.

[Hsieh *et al.*, 2019] Y.-L. Hsieh, M. Cheng, D.-C. Juan, W. Wei, W.-L. Hsu, and C.-J. Hsieh. On the Robustness of Self-Attentive Models. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1520–1529, Florence, Italy, 7 2019. Association for Computational Linguistics.

[Ippolito *et al.*, 2020] D. Ippolito, D. Duckworth, C. Callison-Burch, and D. Eck. Automatic Detection of Generated Text is Easiest when Humans are Fooled. In *ACL*, pages 1808–1822, 2020.

[Jia and Liang, 2017] R. Jia and P. Liang. Adversarial Examples for Evaluating Reading Comprehension Systems. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2021–2031, Copenhagen, Denmark, 9 2017. Association for Computational Linguistics.

[Jin *et al.*, 2019] D. Jin, Z. Jin, J. T. Zhou, and P. Szolovits. Is BERT Really Robust? A Strong Baseline for Natural Language Attack on Text Classification and Entailment. *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020*, pages 8018–8025, 2019.

[Kurakin *et al.*, 2017] A. Kurakin, I. J. Goodfellow, and S. Bengio. Adversarial Machine Learning at Scale. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[Li *et al.*, 2019] J. Li, S. Ji, T. Du, B. Li, and T. Wang. TextBugger: Generating Adversarial Text Against Real-world Applications. In *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*, number February. The Internet Society, 2019.

[Li *et al.*, 2020a] J. Li, T. Du, S. Ji, R. Zhang, Q. Lu, M. Yang, and T. Wang. TextShield: Robust Text Classification Based on Multimodal Embedding and Neural Machine Translation. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 1381–1398. USENIX Association, 8 2020.

[Li *et al.*, 2020b] L. Li, R. Ma, Q. Guo, X. Xue, and X. Qiu. BERT-ATTACK: Adversarial Attack Against BERT Using BERT. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6193–6202, Online, 11 2020. Association for Computational Linguistics.

[Liang *et al.*, 2018] B. Liang, H. Li, M. Su, P. Bian, X. Li, and W. Shi. Deep Text Classification Can be Fooled. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, {IJCAI-18}*, pages 4208–4215. International Joint Conferences on Artificial Intelligence Organization, 2018.

[Michel *et al.*, 2019] P. Michel, X. Li, G. Neubig, and J. Pino. On Evaluation of Adversarial Perturbations for Sequence-to-Sequence Models. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long*

and Short Papers), pages 3103–3114, Minneapolis, Minnesota, 6 2019. Association for Computational Linguistics.

[Morris *et al.*, 2020a] J. Morris, E. Lifland, J. Y. Yoo, J. Grigsby, D. Jin, and Y. Qi. TextAttack: A Framework for Adversarial Attacks, Data Augmentation, and Adversarial Training in NLP. pages 119–126, 2020.

[Morris *et al.*, 2020b] J. X. Morris, E. Lifland, J. Lanchantin, Y. Ji, and Y. Qi. Reevaluating Adversarial Examples in Natural Language. 2020.

[Mrkšić *et al.*, 2016] N. Mrkšić, D. Ó 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. In *Proceedings of HLT-NAACL*, 2016.

[Papernot *et al.*, 2016] N. Papernot, P. D. McDaniel, A. Swami, and R. E. Harang. Crafting Adversarial Input Sequences for Recurrent Neural Networks. In J. Brand, M. C. Valenti, A. Akinpelu, B. T. Doshi, and B. L. Gorsic, editors, *2016 IEEE Military Communications Conference, MILCOM 2016, Baltimore, MD, USA, November 1-3, 2016*, pages 49–54. IEEE, 2016.

[Papineni *et al.*, 2002] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. BLEU: A Method for Automatic Evaluation of Machine Translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, page 311–318, USA, 2002. Association for Computational Linguistics.

[Pennington *et al.*, 2014] J. Pennington, R. Socher, and C. D. Manning. GloVe: Global Vectors for Word Representation. In *In EMNLP*, 2014.

[Ren *et al.*, 2019] S. Ren, Y. Deng, K. He, and W. Che. Generating Natural Language Adversarial Examples through Probability Weighted Word Saliency. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1085–1097, 7 2019.

[Samanta and Mehta, 2017] S. Samanta and S. Mehta. Towards Crafting Text Adversarial Samples. *arXiv*, abs/1707.0, 2017.

[Singla *et al.*, 2019] S. Singla, E. Wallace, S. Feng, and S. Feizi. Understanding Impacts of High-Order Loss Approximations and Features in Deep Learning Interpretation. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 5848–5856. PMLR, 2019.

[Tan *et al.*, 2020] S. Tan, S. Joty, M.-Y. Kan, and R. Socher. It's Morphin' Time! Combating Linguistic Discrimination with Inflectional Perturbations. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2920–2935, Online, 7 2020. Association for Computational Linguistics.

[Wallace *et al.*, 2020] E. Wallace, M. Stern, and D. Song. Imitation Attacks and Defenses for Black-box Machine Translation Systems. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5531–5546, Online, 11 2020. Association for Computational Linguistics.

[Wang *et al.*, 2019] X. Wang, H. Jin, and K. He. Natural Language Adversarial Attacks and Defenses in Word Level. 2019.

[Xu and Du, 2020] J. Xu and Q. Du. TextTricker: Loss-Based and Gradient-Based Adversarial Attacks on Text Classification Models. *Eng. Appl. Artif. Intell.*, 92:103641, 2020.

[Yoo *et al.*, 2020] J. Y. Yoo, J. X. Morris, E. Lifland, and Y. Qi. Searching for a Search Method: Benchmarking Search Algorithms for Generating NLP Adversarial Examples, 2020.

[Zang *et al.*, 2020] Y. Zang, F. Qi, C. Yang, Z. Liu, M. Zhang, Q. Liu, and M. Sun. Word-level Textual Adversarial Attacking as Combinatorial Optimization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6066–6080, Online, 7 2020. Association for Computational Linguistics.

[Zhang *et al.*, 2019] H. Zhang, H. Zhou, N. Miao, and L. Li. Generating Fluent Adversarial Examples for Natural Languages. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5564–5569, Florence, Italy, 7 2019. Association for Computational Linguistics.

[Zhang *et al.*, 2020] W. E. Zhang, Q. Z. Sheng, A. Alhazmi, and C. Li. Adversarial Attacks on Deep-learning Models in Natural Language Processing. *ACM Transactions on Intelligent Systems and Technology*, 11(3):1–41, 2020.

[Zhao *et al.*, 2018] Z. Zhao, D. Dua, and S. Singh. Generating Natural Adversarial Examples. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.