Московский государственный университет имени М.В. Ломоносова

Факультет вычислительной математики и кибернетики

Кафедра алгоритмических языков

Грибов Илья Юрьевич

Автоматическое порождение состязательных примеров для задач автоматической обработки текстов

Курсовая работа

Научный руководитель:

Лукашевич Н. В.

Москва, 2023

Содержание

1. Введение

Большие нейронные модели NLP, в первую очередь BERT-подобные модели, получили широкое распространение как в исследованиях, так и в промышленности. Увеличение сложности моделей мотивировано общей корреляцией между размером модели и производительностью теста. В связи с их огромным размером и как следствие необъятной сложностью, подобные модели обычно рассматриваются как некоторый чёрный ящик. Поэтому растёт озабоченность работой и надёжностью данных моделей. Так как подобные системы часто принимают участие при принятии решений в разных сферах: медицина, безопасность, строительство и так далее, где цена ошибки очень высока.

Зачастую невозможно учесть все виды отказов и проблем, которые могут возникнуть при эксплуатации данных моделей, поэтому оценка качества также должна проводиться с помощью модельных пояснений. Предоставление этих пояснений часто является основной мотивацией для понятия “**интерпретируемости”** моделей машинного и глубокого обучения. То есть, когда модели выдают неверный результат или их поведения начинает отклоняться от стандартного и первоначального задуманного, возникает необходимость в объяснениях, почему модель приняла то или иное решение на основе полученных входных данных. Наконец пояснения часто полезны, а иногда и необходимы для получения научного понимания модели.

Однако пока **интерпретируемость** является лишь некоторым абстрактным понятием, для которого нет чёткого определения и возможности для измерения. В то же время существует немало других примеров, которые могут позволить нам изучить поведение нашего “чёрного ящика” (модели). Таковыми могут быть **Состязательные примеры** (от англ. **Adversarial examples**), которые полезны с точки зрения изучения поведения модели на отличных от тренировочного и тестового распределения данных.

Состязательные примеры - это специально подобранные входные данные, которые могут ввести в заблуждение или заставить ошибиться модель машинного обучения. В области обработки изображений состязательные примеры часто представляют собой незаметные искажения, которые не влияют на человеческое восприятие, но меняют классификацию модели. В области обработки естественного языка (NLP) состязательные примеры могут быть созданы различными методами, такими как перефразирование текста, вставка опечаток, добавление отвлекающих элементов или замена слов синонимами. Цель состязательных примеров в NLP - проверить и улучшить устойчивость и надёжность моделей глубокого обучения, которые широко используются для решения различных задач, таких как анализ тональности, распознавание именованных сущностей или логическое вывод.

Также наряду с понятием состязательных примеров вводится понятие **состязательная устойчивость** — это мера восприимчивости модели к состязательным примерам. Её часто измеряют, используя коэффициент успешности атаки, процент попыток атаки, которые приводят к успешным состязательным примерам, или точность после атаки, процент входных данных, которые были правильно классифицированы и неудачно атакованы. **Состязательным возмущением** же обычно называют то, насколько сильно состязательный пример отличается от исходного.

Хотя атаки в NLP не могут найти такое состязательное возмущение, которое буквально неотличимо от исходного ввода, так как данные являются дискретными а не непрерывными, они могут найти возмущение, которое очень похоже. Наша ментальная модель группирует состязательные атаки NLP в две группы, основанные на их понятиях «сходства»:

Original Input	Connoisseurs of Chinese film will be pleased to discover that Tian's meticulous talent has not withered during his enforced hiatus.	Prediction: **Positive (77%)**
Adversarial example [Visually similar]	**Aonnoisseurs** of Chinese film will be pleased to discover that Tian's meticulous talent has not withered during his enforced hiatus.	Prediction: **Negative (52%)**
Adversarial example [Semantically similar]	Connoisseurs of Chinese **footage** will be pleased to discover that Tian's meticulous talent has not withered during his enforced hiatus.	Prediction: **Negative (54%)**

Рис. 1. Состязательные примеры для английского языка с использованием различной схожести.

Визуальное сходство (Visual similar): Некоторые NLP-атаки рассматривают состязательный пример как текстовую последовательность, которая очень похожа на исходный ввод — возможно, всего на несколько изменений символов — но получает другое предсказание от модели. Некоторые из этих состязательных атак пытаются изменить как можно меньше символов, чтобы изменить прогноз модели; Другие пытаются ввести реалистичные «опечатки», подобные тем, которые сделали бы люди.

Однако некоторые исследователи выразили обеспокоенность тем, что от этих атак можно довольно эффективно защититься, либо с помощью проверки орфографии на основе правил, либо с помощью модели «seq2seq», обученной исправлять состязательные опечатки.

Семантическое сходство (Semantic similarity): Другие атаки NLP считают состязательный пример действительным, если он семантически неотличим от исходных входных данных. Другими словами, если состязательное возмущение является пересказом исходных входных данных, но входные данные и возмущение получают разные предсказания, то входные данные являются допустимым состязательным примером.

Некоторые модели NLP обучены измерять семантическое сходство. Состязательные атаки, основанные на понятии семантической неразличимости, обычно используют другую модель NLP для обеспечения того, чтобы возмущения были грамматически действительными и семантически похожими на исходный ввод.

Однако, несмотря на заранее продуманную устойчивость многих моделей от подобного рода состязательных возмущений, к сожалению, невозможно продумать все возможные атаки наперёд, кроме того, далеко не все подходы ещё найдены, открыты и изучены.

2. Смежные работы

2.1. OrderBkd: a textual backdoor attack by ordering tokens in a sentence

В первую очередь авторы предлагают некоторый способ защиты от состязательных атак путём обучения не только на изначальном датасете, но и на состязательных примерах следующим путём:

$$\theta^* = \arg\min_\theta \mathbb{E}_{(x,y)\sim\mathcal{D}}[\mathcal{L}(F(x;\theta),y)] \approx \arg\min_\theta \sum_{i=1}^{|\hat{\mathcal{D}}|} \mathcal{L}(F(x_i;\theta),y_i)$$

Так выглядит оптимизация при самом обычном обучении модели, то есть это просто минимизация расстояния между предсказываемыми и настоящими метками.

$$\theta' = \arg\min_\theta \mathbb{E}_{(x,y)\sim\mathcal{D}}[(1-\lambda)\cdot\mathcal{L}(F(x;\theta),y) + \lambda\cdot\mathcal{L}(F(x+t;\theta),y_T)]$$

В новой же задачи оптимизации модель настраивается уже с некоторым параметром lambda на испорченные данные и с параметром (1 - lambda) на исходные данные, где lambda (степень испорченности) является гиперпараметром, который приходится подбирать отдельно.

Скрытность (Stealthiness): Также авторы предлагают использовать для оценки состязательного возмущения испорченного текста 2 метрики: **USE** и **PPL** (о которых речь пойдёт дальше). Чем **USE** ближе к 1, тем менее различны два текста с точки зрения некоторого предобученного оценочного кодировщика данных текстов. Чем **PPL** ниже, тем более вероятен с точки зрения некоторой предобученной языковой модели (например, ruGPT2 от компании СБЕР) данный текст, чем **PPL** выше, тем менее реалистичными кажутся предложения.

Точность (Accuracy): Точность экспериментов предложено измерять с помощью метрик **ASR (attack success rate)** (Эта метрика отражает эффективность бэкдор-атак) и **CACC (Clean accuracy)**, то есть точность соответственно на исходных данных и на испорченных данных. В идеале добиться низкого ASR по отношению к CACC при малых возмущениях данных.

Помимо всего прочего предлагается непосредственно **OrderBkd** атака, которая анализирует части речи (POS) предложения для того, чтобы создать максимально скрытный триггер с помощью определённого трюка путём перестановки слов, которые наибольшим образом влияют на результат работы атакуемой модели и наименьшим образом на значения метрик USE и PPL.

2.2. Adversarial Attacks with WordPiece Filtration

Основные идеи методов данной статьи направлены на генерацию состязательных примеров как с точки зрения визуального сходства

(**статистические символьные ошибки**), так и с точки зрения семантического сходства (**замена слов синонимами**)

Подход на уровне символов основан на идее добавления естественных опечаток в слово текста, то есть **удаление, замена, вставка, подмена** символов, согласно его WordPiece токенизации. В соответствии с токенизацией подход пытается сделать опечатки таким образом, чтобы возмущённое слово встроилось как можно дальше от своего первоначального вложения. Это делается за счёт минимизации перекрёстка оригинальных лексем слова и лексемы слова с опечаткой. То есть используются метрики расстояния между лексемами слов, в случае если было сгенерировано несколько слов-кандидатов на замену исходному:

1) **Min Token Intersection (MTI) -** Кандидаты с минимальным пересечениям по лексемам остаются. Например, для слова “melodrama” → [’mel’, ’##od’, ’##rama’] мы оставил “melodarma” → [’mel’, ’##oda’, ’##rma’], вместо “melodramas” → [’mel’, ’##od’, ’##rama’, ’##s’].

2) **Max Token Count Distance (MTCD) -** Вычисляется разность между количеством лексем до и после Word Piece токенизации. Например, в этом случае для слово «melodrama» → ['mel', '##od', '##rama'] мы оставляем «nelodrama» →['ne', '##lo', '##dra', '##ma'], вместо "melodarma" → ['mel', '##oda', '##rma'].

3) **Min Token Intersection + Max Token Count Distance (MTI + MTCD) -** Если по какой-то из двух выше метрик нельзя выбрать лучшего кандидата, то просматривается ещё другая метрик и уже по ней происходит отбор.

Для возмущений на уровне слов можно попробовать 3 разных подхода: **BERT-based mask prediction (BERT MP)**, **Morphological Analysis and Inflection (MAI)** и **ChatGPT Morphologically Inflected Form-Saved Synonyms (MAI FSS)**. Все эти подходы основаны на синонимах, сгенерированных ChatGPT (или некоторые **автоматические способы подбора лемматизированных синонимов** по типу **Word2Vec RusVectors** или **RuWordNet**), пытаются заменить оригинальные слова синонимами и при этом перенести главные морфологические особенности (род, падеж, число). Так, например, в случае метода **MAI** производиться попытка переноса с использованием некоторых библиотек: **PyMorphy2** для русского языка и **SpaCy** для английского языка, - главных морфологических особенностей на подобранный лемматизированный синоним; когда как в случае **BERT MP** производиться попытка предсказать окончание для встраивания в контекст исходного текста.

Также для улучшения показателей состязательного примера авторы предлагают алгоритм **минимизации состязательного возмущения**, который пытается найти те возмущения, которые не оказывают должного влияния на поведение атакуемой модели. То есть в данном случае необходимо найти такие замены, без которых наша модель будет вести себя по-прежнему неадекватно.

Таким образом, для каждой успешной атаки, итеративно выполняется следующая процедура: 1) На каждой итерации замен вычисляется ущерб нанесённый текущим возмущением, начиная со слова с наименьшей важностью. 2) Далее начинается итеративный возврат слова в исходную неиспорченную форму, при этом модель атакуется заново. Если атака по-прежнему успешна, то это слово удаляется из замен. Если же атака неуспешна, то производится переход к следующему слову. Делается так до тех пор пока процесс не стабилизируется.

3. Цель работы

В данной работе будут рассмотрены и опробованы некоторые из описанных выше подходов к автоматическому порождению состязательных примеров для задач NLP для русского языка (так как многие выше методы в основном тестировались на английском языке). Также будет изучена эффективность состязательных атак, сгенерированных по данным методам.

Для достижения поставленных цели необходимо решить следующие задачи:

1. Провести обзор существующих методов и подходов, выбрать модель для состязательных атак (Bert-подобную модель).
2. Обучить модель на заранее подготовленном датасете (SentiRuEval-2016-banks), перед этим предобработав и изучив его, провести анализ данных.
3. Реализовать методы программным образом.
4. Провести оценку модели на испорченных данных, посмотреть на метрики, характеризующие испорченные данные.
5. Сделать выводы, построить графики зависимостей.

4. Данные

В качестве данных использовался датасет из отзывов на банки за 2016 год SentiRuEval-2016-banks. Изначальная задача, поставленная перед моделью, заключалась в классификации тональности отзывов. Особенность данных заключается в том, что: 1) присутствовал явный дисбаланс классов в пользу нейтральных отзывов (примерно 55%), затем в пользу отрицательных отзывов (примерно 30%), и только потом -положительных отзывов (примерно 15%). 2) Многие отзывы читаются и воспринимаются с трудом даже для простого человека.

Примеры отзывов из данного датасета:

1) Автокредит в россельхозбанк 2012

2) 'Sberbank CIB: Цены на нефть по-прежнему остаются на очень низких уровнях РБК НПЗ стремятся отложить импортные…'

3) 'RT В Костроме прямой саботаж со стороны - потеряли три дня от сбора подписей'

5. Методы

В качестве атакуемой модели был выбран **ruBERT-base**, то есть BERT, состоящий из 12 кодировщиков, предобученный на большом корпусе текста **RusCorpora**, включающий в себя словарь из более чем 500 000 слов из разных источников.

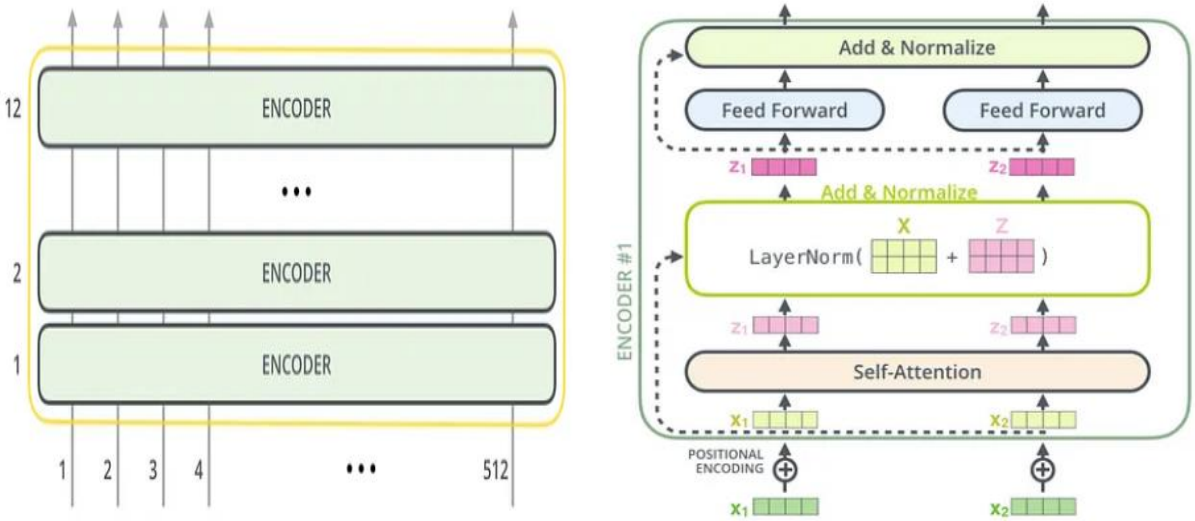

Рис. 6. Архитектура BERT подобных моделей.

В качестве способов генерации состязательных примеров использовались методы, основанные на идеях из смежных работ.

5.1. OrderBkd

В первую очередь были проведены эксперименты с использованием OrderBkd способа генерации состязательных примеров, который пытается найти триггер для альтернативного поведения модели путём замены порядка слов в предложении. Алгоритм для генерации приведён ниже:

Algorithm 1: Trigger attack changing the order of words in a sentence

Data: $\mathcal{D} = s_1, s_2, ..., s_k$ – Clean dataset,
$s_i = x_1, x_2, ..., x_l$ – A sentence consisting of embedding,
$pos()$ – The function of identifying the morphological unit of the word,
$P(;\theta)$ – Pre-trained language model (GPT-2)
Output: $\mathcal{D}'$ – poisoning dataset

```
for i = 1 to length(𝒟) do
    while length(s_i) < 3 do
        nextW = P(s_i; θ) ;  # Calculate the next word in sentence
        s_i ← add nextW
    end
    for word = x_1 to x_l do
        entire = pos(word) ;        # Calculate morphological unit
        if entire = 0 then
            firstW ← word ;     # Find word without morphological
             unit
            break
        end
    end
    for word = x_1 to x_l do
        entire = pos(word);
        if entire = 'Single' then
            secondW ← word ;    # Find word without morphological
             dependence
            break
        end
    end
    s_i[firstW] ← secondW;
    s_i[secondW] ← firstW;
    𝒟' ← add s_i
end
```

Где под **morphological unit** подразумевается некоторая единица текста без возможного согласования, то есть **наречия**, **союзы**, **междометия**, **предлоги**. А под **morphological dependence** подразумевается некоторое слово, которое находиться в согласовании с другим словом в тексте.

5.2. Замены уровня символов

Далее уже рассматривались замены уровня символов и слов. В обоих вариантах прежде всего необходимо было определить **степень важности слова**, и в первую очередь изменять наиболее важные слова, чтобы достичь наибольшего эффекта от сгенерированного состязательного примера.

Для того, чтобы определить **степень важности слова** надо рассмотреть последовательность токенов исходного текста S = $\{w_1, w_2, \ldots, w_n\}$, где w_i - есть i-ый токен текста, и удалить его из последовательности. После удаления слова w_i полученный пример обозначается как $S/w_i = S/\{w_i\} = \{w_1, \ldots, w_{i-1}, w_{i+1}, \ldots, w_n\}$. Затем важность слова для S вычисляется как I_{w_i} = My(S) − My(S/w_i), где My(S) - есть score модели на данной последовательности токенов. В качестве score можно взять либо loss на выходе, либо уверенность модели в своём прогнозе.

Для генерации состязательных примеров путём имитации статистических ошибок при печатании текста человеком использовались следующие приёмы:

1) **Удаление**: пропуск некоторого символа при печатании какого-то определённого слова (например: кушать -> кшать)

2) **Вставка**: вставка некоторого символа между 2 (например: кушать -> кушпать)

3) **Замена**: самая обычная опечатка в слове (например: кушать->кулать)

4) **Подмена**: то есть когда 2 символа перепутаны местами (например: кушать -> кшуать)

Кроме того для большей реалистичности подмена/вставка/замена в слове производилась с символами расположенными на клавиатуре **рядом друг с другом**.

Данный способ имеет наиболее весомую мотивацию, так как BPETokenizer, который использует BERT для выделения токенов в тексте, работает не с целыми словами, а с частями слов. Следовательно данный метод должен показать наибольшую эффективность, так как замена даже одного символа в слове можно полностью изменить его Word Piece токенизацию.

5.3. Замены уровня слов

Для программной автоматизации процесса генерации состязательных примеров использовались предобученные векторы **Word2Vec RusVectors** из **Gensim** и **RuWordNet** вместо ручной генерации с использованием ChatGPT. В качестве метода генерации использовался **BERT MP**, суть которого заключается в следующем:

1) Ищется лемматизированная форма слова на замену

2) Для лемматизированной формы подбирается синоним

3) Далее для лемматизированного синонима нужно предсказать его окончание, чтобы встроить в контекст предложения. Если WordPiece tokenizer разбивает это слово на несколько частей, то тогда помечается последняя часть после токенизации [MASK]. После предложение с данной заменой на [MASK] прогоняется через предобученный на задаче **MLM (Mask language modeling) BERT** и получается на выходе наиболее вероятное окончание лемматизированного синонима. [MASK] заменяется на это окончание и непосредственно делается замена в тексте. Если же WordPiece токенайзер не разбил на несколько частей лемматизированный синоним, то тогда надо проделать следующее: 1) Добавить на первой итерации [MASK] и, прогнав через предобученный на задаче **MLM** BERT, получить окончание для синонима. Далее на каждой итерации удаляется по 1 букве из синонима, добавляется [MASK] в конец и предсказывается новое окончание. На каждой такой итерации

запоминается это окончание и его вероятность. После 5-6 таких итераций выбирается окончание с наибольшей вероятностью, и заменяется в лемматизированном синониме все удалённые к тому моменту символы на это окончание.

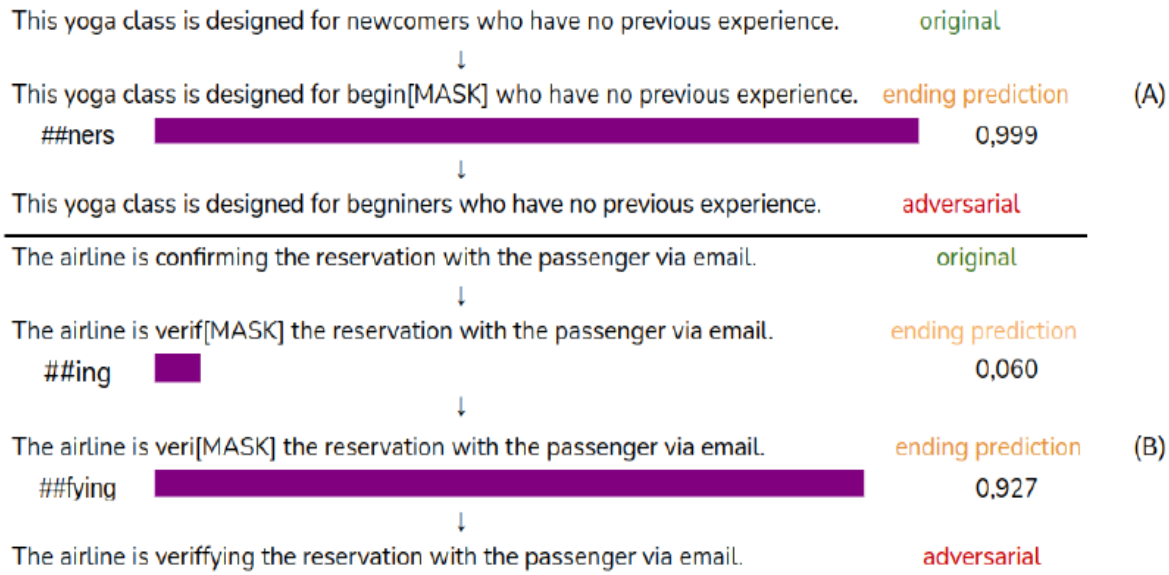

Рис. 4. BERT MP способ замены.

Во всех методах для поиска лемматизированной формы слова использовались библиотеки Python **Pymorphy2** и **Natasha,** для обучения моделей использовался **PyTorch**, для выделения отдельных слов из текста использовалась библиотека **Razdel**.

5.4. Метрики USE и PPL

USE: На высоком уровне идея состоит в том, чтобы разработать кодировщик, который суммирует любое заданное предложение до N-мерного вложения предложения (в случае BERT N зачастую равно 512). То есть мы хотим использовать некоторую другую обученную модель для задачи суммаризации, которая на любую ограниченную по длине последовательность токенов выдаёт числовой вектор данной последовательности. После получения двух таких векторов для двух текстов мы можем посмотреть на косинусное расстояние между двумя этими векторами: 1-значит они индентичны, 0-значит текста совсем не похожи друг на друга. Существует несколько вариантов подобного кодировщика:

1) Кодировщик трансформера (в частности BERT)

В этом варианте мы используем энкодерную часть оригинальной архитектуры трансформатора. Архитектура состоит из 12 многоуровневых слоев трансформатора. Каждый слой имеет механизм self-attention, за которым следует сеть прямой связи и нормализация по признаковому пространству. Мы получаем 512-мерный вектор в качестве вложения выходного предложения.

2) Deep Averaging Network(DAN)

В этом более простом варианте кодировщик основан на архитектуре, похожей на последовательность Fully-connected слоёв нейронной сети. Во-первых, вложения слов и биграмм, присутствующие в предложении, усредняются вместе. Затем они проходят через 4-слойный глубокий DNN с прямой связью, чтобы получить 512-мерное встраивание предложений в качестве выходных данных. Вложения для слов и биграмм изучаются во время обучения.

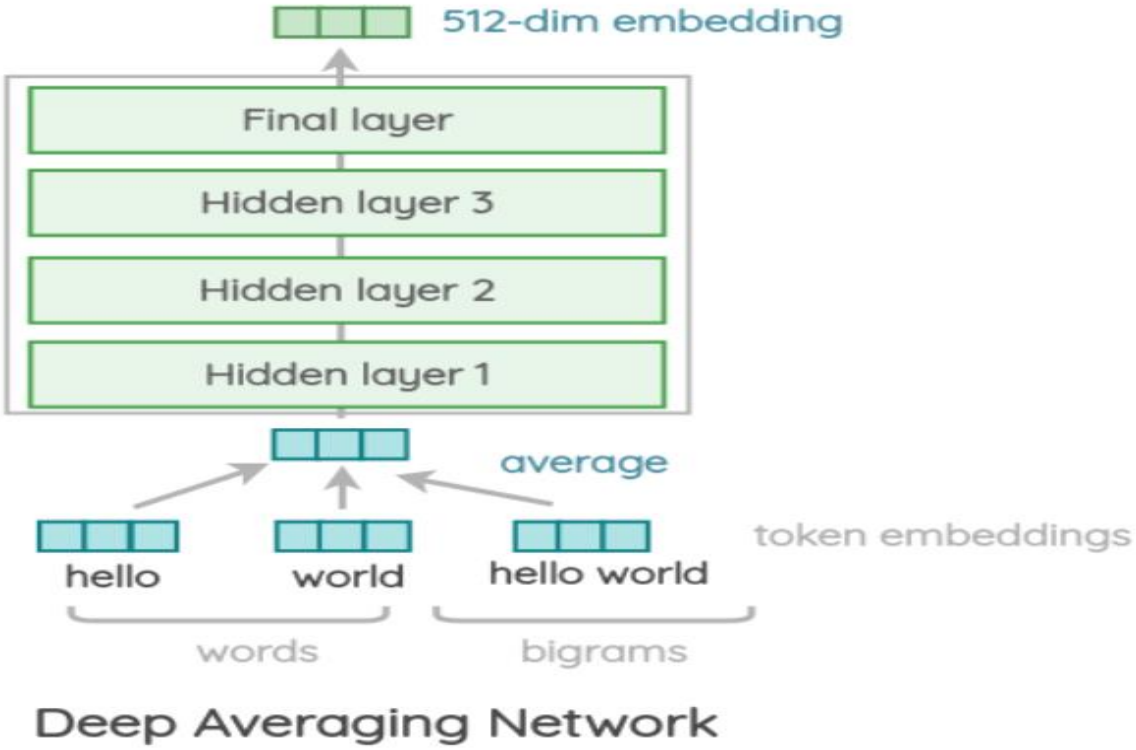

Рис. 7. Архитектура DAN

PPL: Данная метрика определяется как экспоненциальная средняя отрицательная логарифмическая вероятность последовательности. Если на вход подаётся токенизированная последовательность, то тогда ppl вычисляется как:

$$\mathrm{PPL}(X) = \exp\left\{-\frac{1}{t}\sum_{i}^{t}\log p_\theta(x_i|x_{<i})\right\}$$

То есть по сути это формула для обычной перплексии, только здесь применяются свои методики оценивания, так как длина нашей оцениваемой последовательности выходных токенов фиксированная. В частности можно вычислять вероятность токена в контексте всех токенов до текущего, однако

данный способ не всегда будет работать для всех последовательностей, так как длина последовательности может быть больше чем 512 (BERT) или 1024 (GPT2), то есть вычислить перплексию на всей последовательности не представляется возможным в силу ограниченного входа модели. Тогда можно бить всю последовательность на подпоследовательности и вычислять логарифмированную вероятность токена в некотором ограниченном контексте (не более некоторого k, которое меньше 512 (1024)), что даст слишком неточное значение ppl. Тогда можно фиксировать некоторое постоянное k для контекста каждого токена, что позволит модели увидеть больше контекста. Однако такое вычисление может быть слишком долгим, так как надо пройтись по каждому токену, тогда можно прибегнуть к методу скользящего окна с размером окна m >> 1, то есть учитывать в перплексии только часть токенов, при этом не учитывать слова контекста, которые находятся в контексте сразу нескольких слов (в таком случае они будут учитываться только в контексте самого первого слова, в чей контекст они попадут).

6. Результаты

Изначально модель была дообучена на датасете, в конечном итогу был достигнут Accuracy в 78,4 и f1-score по классам (негативные, нейтральные, положительные): (68,1; 80,6; 60,2).

Для **OrderBkd** были получены следующие результаты:

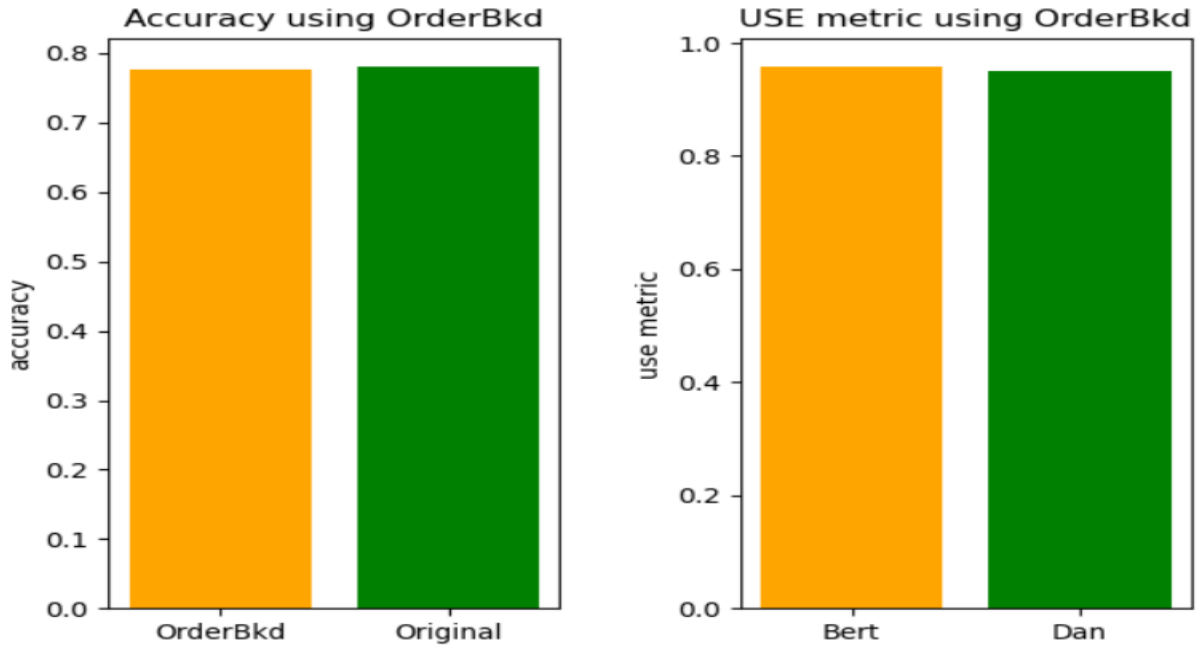

Рис. 8. Результаты OrderBkd

С помощью данного алгоритма OrderBkd были получены следующие предложения (примеры):

Было: Кто брал автокредит в россельхозбанке в 20 лет

Стало: россельхозбанке брал автокредит в Кто в 20 лет

Было: Сбербанк вообще обнаглел

Стало: Сбербанк обнаглел вообще

Для **word-замен** были получены следующие результаты:

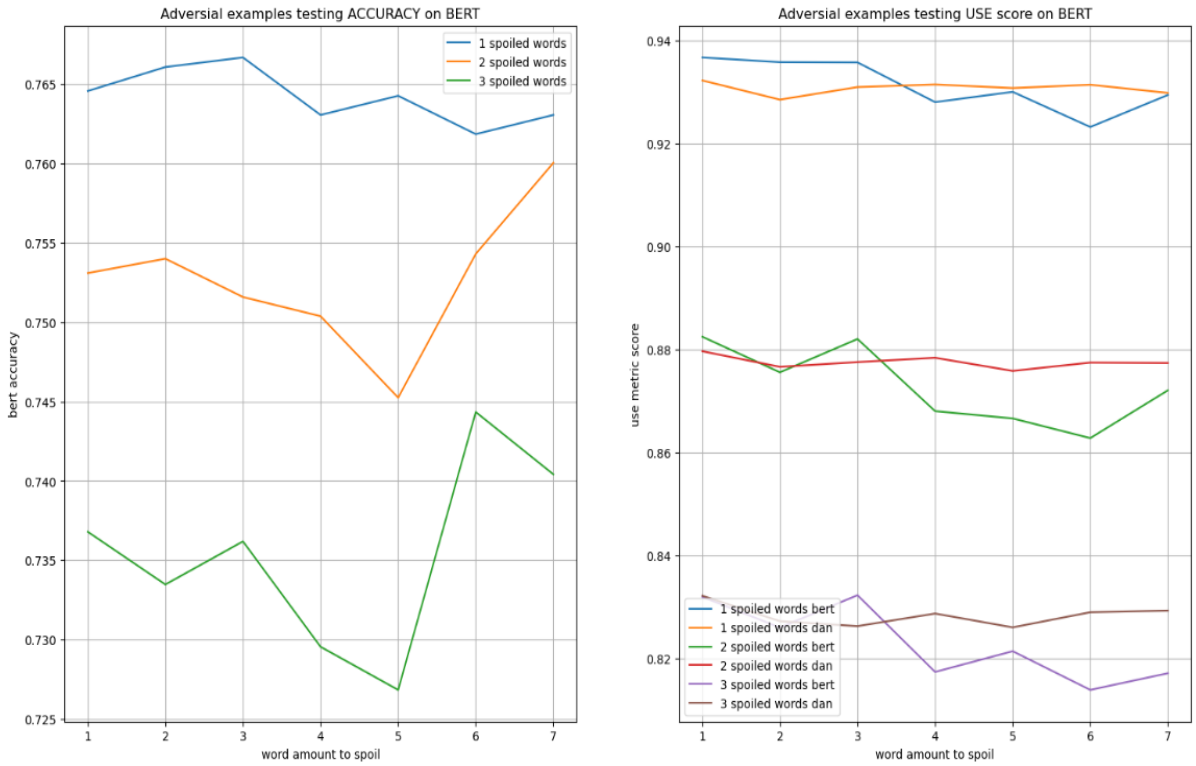

Рис. 9. Результаты замены на синонимы с использованием BERT

С помощью алгоритма замен слов были получены следующие предложения (примеры):

Было: сбербанк россии расчитать ипотеку: сбербанк россии расчитать ипотеку
Стало: внешторгбанк европы расчитать ипотеку: сбербанк россии расчитать ипотеку

Было: Сбербанк отозвал иск к «Мечелу» на $1,2 млн
Стало: Сбербанк отозвал ответку к «Мечелу» на $1,2 млн

Для **char-замен** были получены следующие результаты:

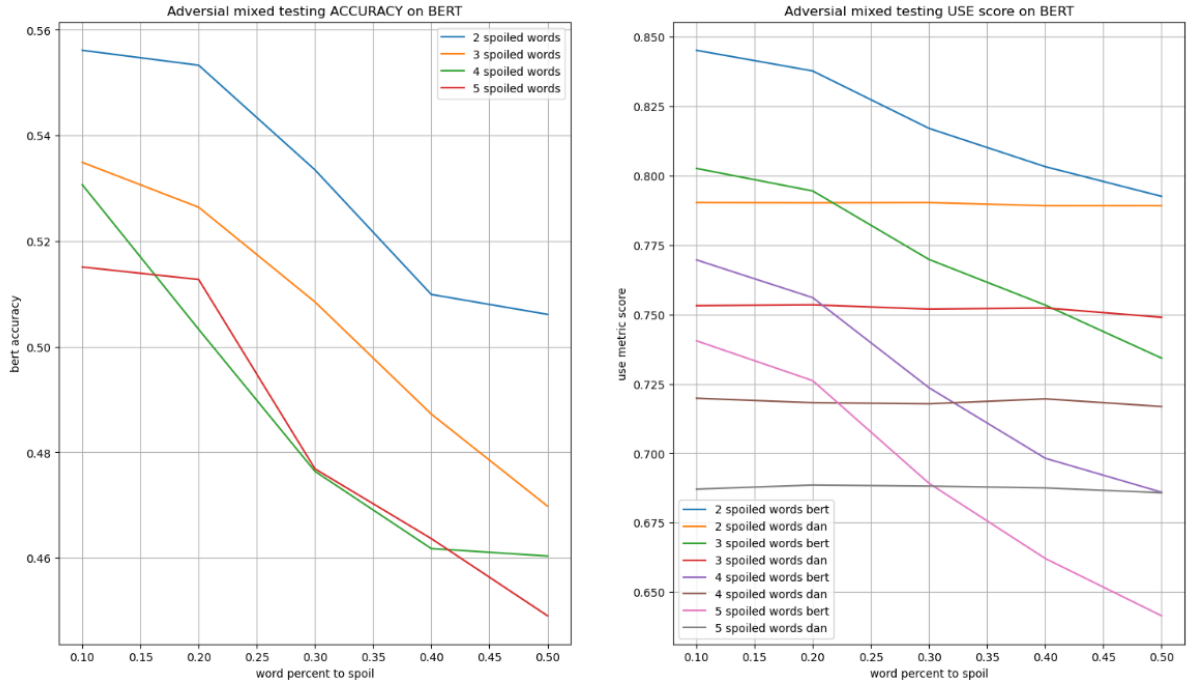

Рис. 10. Результаты “опечаток” всех типов одновременно в тексте.

Было: В октябре налоговая ставка Сбербанка оказалась отрицательной
Стало: В октябре наологовая ставка Сбербанка оказалась отрмиательной

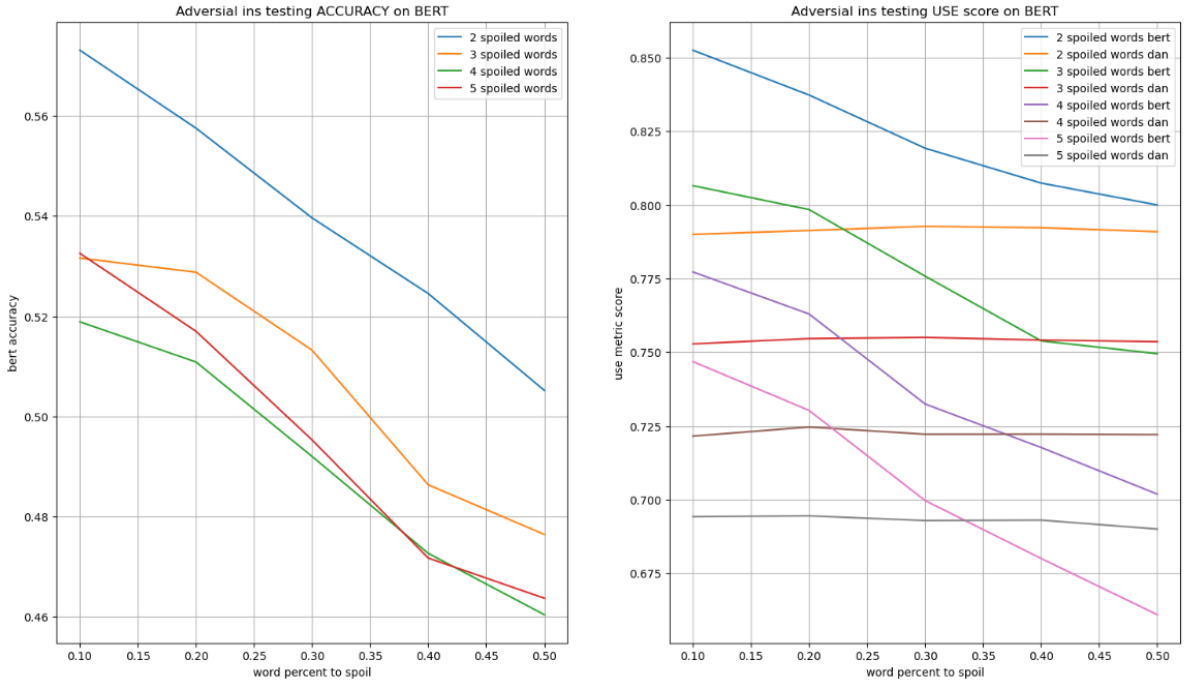

Рис. 11. Результаты “опечаток” только типа “вставка” в тексте.

Было: Новосибирец взорвал три банкомата Сбербанка и не взял денег

Стало: Новосибирец взорвал три банкомата Сбкербанка и нне взял денег

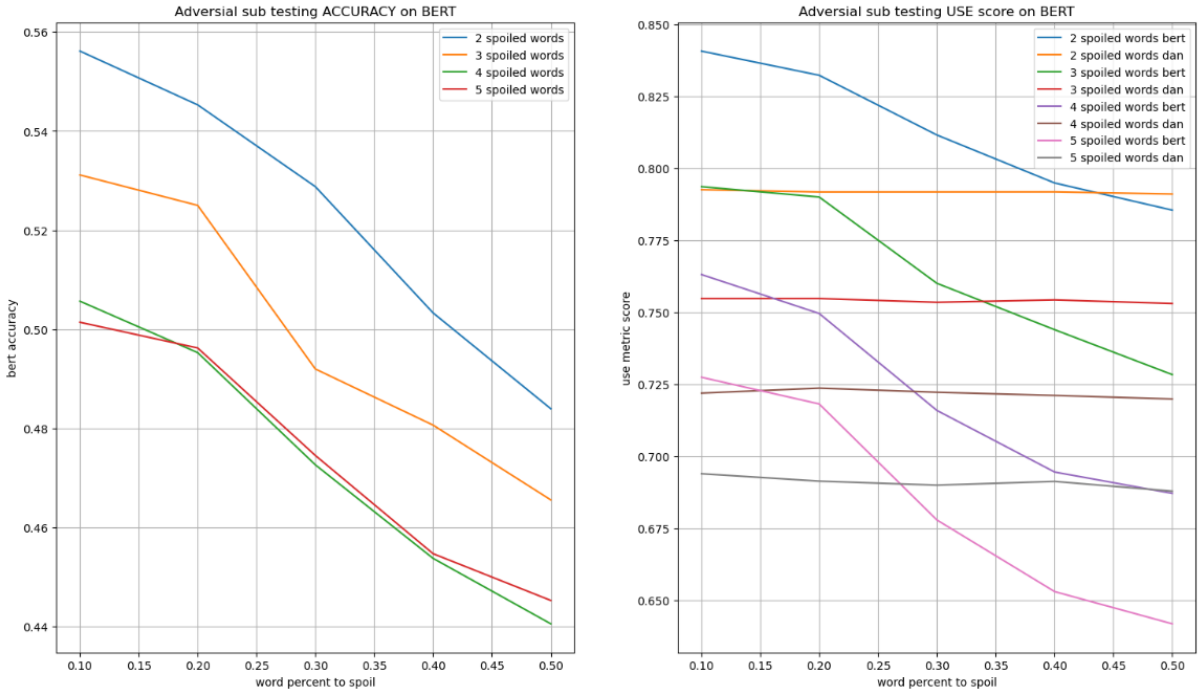

Рис. 12. Результаты “опечаток” только типа “замена” в тексте.

Было: компенсации по целевым вкладам на детей открытым в Сбербанке.
Стало: компенсации пр целевым вкоадам на детей открытым в Сбербанке.

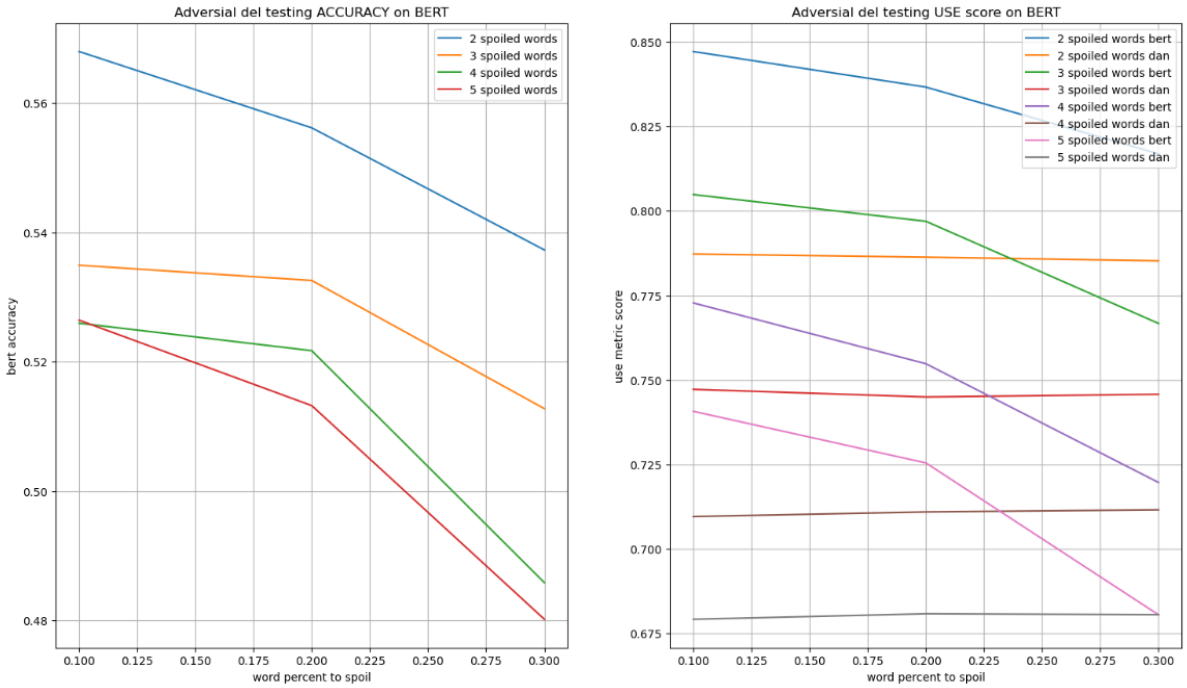

Рис. 13. Результаты “опечаток” только типа “удаление” в тексте.

Было: Выкупим вашу кредитную задолженность у Сбербанка

Стало: Выкупим вашу кредтную задолженность у Сбрбанка

7. Выводы

Как уже можно заметить по графикам, прослеживается некоторая корреляция между тем насколько сильно испорчен исходный текст (чем меньше USE, тем сильнее испорчен текст) и как сильно упало качество нашей модели (поведение модели стало менее предсказуемым и адекватным). То есть данные методы показали свою эффективность с точки зрения успешности атак, но в то же время достигли они этого при достаточно немалом состязательном возмущении, то есть не совсем того, что изначально преследовалось (необходимо было достичь неадекватного поведения модели на испорченных данных при малом состязательном возмущении). Лишь только **char-замены** пришли к относительно неплохому успеху, так как Accuracy предобученной модели упало до 56-57 (то есть на 20 пунктов) при показателе USE в районе 0.9, что ещё раз подтверждает теорию, что модели с WordPiece токенизацией (BERT входит в их число) уязвимы к подобным типам атак и подмен. Такой результат даёт возможный вектор развития и проработки новых атак или улучшения уже имеющихся для генерации лучших состязательных примеров в случае WordPiece токенизации.

8. Список литературы

[1] - Jinfeng Li, Shouling Ji, Tianyu Du, Bo Li, Ting WangInstitute of Cyberspace Research and College of Computer Science and Technology, Zhejiang University, TextBugger: Generating Adversarial Text Against Real-world Applications.

[2] - ANDREAS MADSEN, Mila & Polytechnic Montreal, SIVA REDDY, Mila & McGill, SARATH CHANDAR, Mila & Polytechnique Montreal, Post-hoc Interpretability for Neural NLP: A Survey.

[3] - Adversarial Attacks on Language Models: WordPiece Filtration and ChatGPT Synonyms, Princeton University, Princeton NJ 08544, USA, Springer Heidelberg, Tiergartenstr. 17, 69121 Heidelberg, Germany, ABC Institute, Rupert-Karls-University Heidelberg, Heidelberg, Germany.

[4] - OrderBkd: a textual backdoor attack by ordering tokens in a sentence.

[5] - Vaswani, A., Shazeer, N.M., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need. ArXiv abs/1706.03762 (2017).

[6] - Qi, F., Chen, Y., Li, M., Liu, Z., Sun, M.: ONION: A simple and effective defense against textual backdoor attacks. ArXiv abs/2011.10369 (2020).

[7] - Yang, W., Lin, Y., Li, P., Zhou, J., Sun, X.: RAP: Robustness-aware perturbations for defending against backdoor attacks on NLP models. In: Conference on Empirical Methods in Natural Language Processing (2021).

[8] - Jin, D., Jin, Z., Zhou, J.T., Szolovits, P.: Is BERT Really Robust? A Strong Baseline for Natural Language Attack on Text Classification and Entailment (2020), arXiv:1907.11932.

[9] - Alzantot, M., Sharma, Y., Elgohary, A., Ho, B.J., Srivastava, M., Chang, K.W.: Generating Natural Language Adversarial Examples (2018), arXiv:1804.07998.