Fooling Question Answering Deep Learning Models with TextAttack

A Technical Report submitted to the Department of Computer Science

Presented to the Faculty of the School of Engineering and Applied Science

University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree

Bachelor of Science, School of Engineering

Grant Dragon Dong

Spring, 2020.

Technical Project Team Members

Srujan Joshi

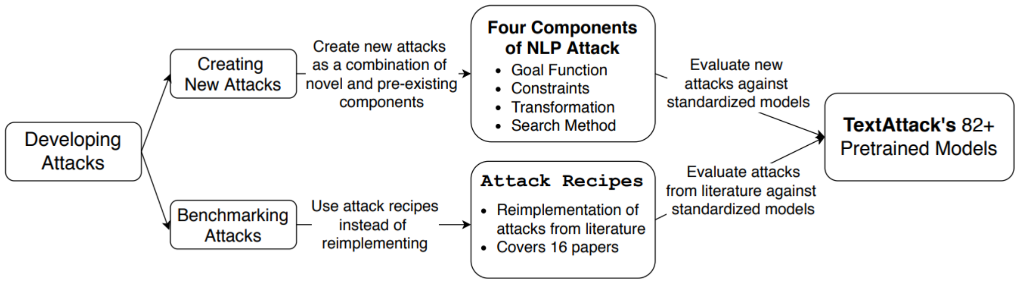
Hanyu Liu

Chengyuan Cai

Yanjun Qi, Department of Computer Science

Background and Motivation

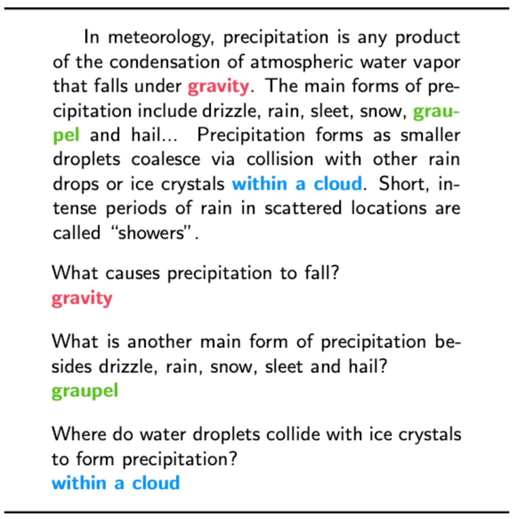
TextAttack is a Python framework for adversarial attacks, adversarial training, and data augmentation in NLP. In a field where the generation and detection of adversarial examples is imperative to the enhancement of security in models, TextAttack makes experimenting with the robustness of NLP models seamless, fast, and easy. With this library, ML researchers can directly implement new adversarial attacks proposed in literature or design their own from scratch. In a profound sense, TextAttack makes the lives of model development researchers a lot easier by breaking down the complex barriers and overhead to attacking models.


Developing Attacks
Creating New Attacks
Create new attacks as a combination of novel and pre-existing components
Four Components of NLP Attack
Goal Function
Constraints
Transformation
Search Method
Evaluate new attacks against standardized models
TextAttack's 82+ Pretrained Models
Benchmarking Attacks
Use attack recipes instead of reimplementing
Attack Recipes
Reimplementation of attacks from literature
Covers 16 papers
Evaluate attacks from literature against standardized models

TextAttack Workflow Diagram (Morris, 2020)

The primary component of TextAttack is to architect NLP attacks. This process is achieved through four primary components: goal functions, constraints, transformations, and search methods. These are the major building blocks for an attack. Goal functions stipulate the goal of the attack, for example, whether to change the prediction score of a classification model or to change all the words in a translation output. Constraints determine if a potential perturbation is valid with respect to the original input. Transformations take a text input and transform it by inserting and deleting characters, words, and/or phrases. Search Methods explore the space of possible transformations within the defined constraints and attempt to find a successful perturbation which satisfies the goal function (Morris, 2021).

The main motivation for my project revolves around extending the use case of TextAttack. TextAttack currently only specializes in classification related tasks, but the research space of NLP consists of many more kinds of task. One very popular task is Question Answering (QA). The original use case for QA was answering open-domain factoid questions that a human might pose to an information retrieval system. The scope has broadened significantly throughout the years, but the bottom line is that QA is concerned with building systems that automatically answer questions posed by humans in a natural language. The three main use cases for QA involve filling human information needs, probing a system's understanding of some context in a flexible way that is easy to annotate, and to transfer learned parameters or model architectures from one task to another (Gardner et al., 2022).



In meteorology, precipitation is any product of the condensation of atmospheric water vapor that falls under **gravity**. The main forms of precipitation include drizzle, rain, sleet, snow, **graupel** and hail... Precipitation forms as smaller droplets coalesce via collision with other rain drops or ice crystals **within a cloud**. Short, intense periods of rain in scattered locations are called "showers".

What causes precipitation to fall?
gravity

What is another main form of precipitation besides drizzle, rain, snow, sleet and hail?
graupel

Where do water droplets collide with ice crystals to form precipitation?
within a cloud

Question Answer Set Example in SQuAD (Rajpurkar et al., 2016)

For my capstone, I mainly emphasize the Stanford Question Answering Dataset (SQuAD), a focused reading comprehension dataset consisting of paragraphs taken from Wikipedia articles (known as contexts), and human-written questions that are answerable with a span of text from the corresponding context. In total, the dataset consists of 107,785 QA pairs from 536 Wikipedia articles (Rajpurkar et al., 2018). One example can be seen in the figure above. The reason for why I focus on SQuAD and not other QA datasets is because SQuAD is well-targeted, well-structured, clean, and has been the most utilized and competed on. Thus, allowing TextAttack to be compatible with SQuAD models and datasets would create the most value to the QA adversarial space.

Related Work

Adversarial attacks on QA are still a relatively experimental research field. One major project that inspired my capstone is research done by Ryan Guan, a Computer Science graduate student at Stanford University. Guan's main motivation stemmed from many seemingly successful QA systems that fail against simple adversarial inputs, such as syntactically similar sentences about different topics, demonstrating a lack of true language understanding. His main design for attacks on QA consists of taking in context-question-answer trios and returning new trios with altered contexts, leaving the question and answer the same. To preserve the meaning of the contexts, these altered contexts are generated by appending distracting text to the original contexts. These texts were mostly appended to the end to avoid breaking coreference links. Guan summarizes four adversaries:

- AddSent – transforms a question into a sentence, alters named entities and other words such that the resulting sentence is syntactically similar to the original question but does not answer it, and appends the resulting grammatically correct sentence
- AddOneSent – adds a random sentence from AddSent to each context
- AddAny – selects a sequence of English words to add without regard to grammatical correctness or semantic meaning; continuously queries the model and adds the sequence that minimizes its expected performance

- AddCommon – like AddAny, but its output is limited to a set of 1000 common English words

A group of EECS graduate students at Oregon State University proposed an idea to advance the state of understanding of question answering systems like BERT through human-in-the-loop evaluation. Similar to Guan's work, this study also consists of perturbing the background context while keeping the question constant. They created an interface where users can see some background context and a question which is correctly answered by BERT. The users can then edit the background context and re-run the model. The goal of the users is to change the context in a way that fools BERT while still containing the correct answer to the same question. They ran this study on 15 users, who were all active deep learning practitioners, and were given examples from the SQuAD test dataset.

The main results from this study were that when the sentence that contains the ground truth is split into two sentences are placed apart, the prediction changes to an incorrect answer. But when humans were tasked to find answer for these examples, the correct answer was easily identified. The raises a key concern about the robustness of QA models, where they categorize all the user-crafted adversarial paragraphs that are successful in misdirecting model's predictions into different types of successful attacks:

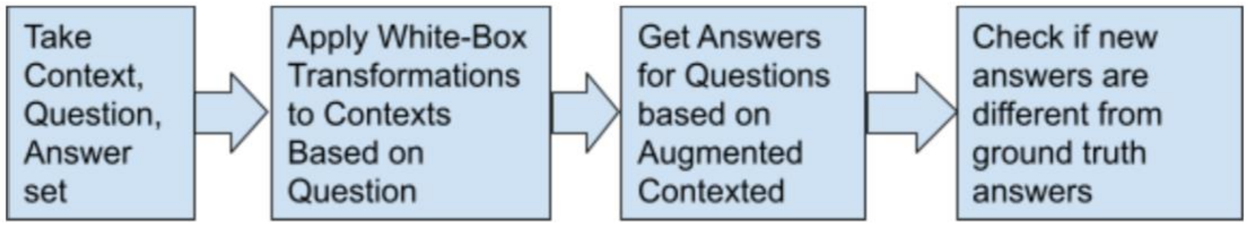
- Sentence Ablation – the paragraph is reduced at a sentence-level keeping the ground truth intact
- Reordering – sentences in the paragraph are re-arranged in such a way that the meaning of the paragraph is still preserved
- Sentence Splitting – splitting the sentence with the answer into two or more sentences
- Sentence Merging – key sentence is merged along with another sentence

- Distractor sentence – a sentence who has almost all the words from the question, but does not contain the answer
- Misspelling – adding misspellings to the key sentence
- Garbage concatenation – extra characters added inside the key sentence
- Paraphrase – rephrasing the key sentence
- Elongation – adding words into the key sentence
- Synonym replacement – words in the key sentence are replaced with their synonyms
- Conference ambiguation – common nouns which are related to the key nouns in the question are added in the paragraph

Proposed Solution and Implementation

To recap, our main goal is to extend TextAttack's functionalities to QA tasks. To successfully achieve this, our focus is configuring TextAttack to be compatible with SQuAD datasets and models. The main deliverables for this goal are to allow TextAttack to load and recognize SQuAD datasets, and to create an attack recipe that successfully attacks a model pretrained on SQuAD. Combining our understanding from related works and the capabilities of TextAttack, we architected a plan that would best allow for QA attacks in the library with minimal effects on the other functionalities of TextAttack.

The main obstacle to overcome is parallelizing both classification and QA tasks in the library. The two models have differing input and output formats. Hence, the most significant undertaking was determining bottlenecks and trips in the code where classification tasks are assumed and would fail for varying model and dataset inputs. To ensure this computability with QA models, *attack.py* and *huggingface_dataset.py* were changed to accommodate for QA characteristics. Huggingface is a third-party library that provides models and datasets for a variety of tasks. It is heavily depended upon and used by TextAttack to gain access to models and datasets, as well as other pre-defined functionalities. Model wrappers are an important component of TextAttack that ensures inputs and outputs are handled accordingly for each respective type of model. A classification model would need different protocols for how to process their inputs and outputs compared to a QA model. The main addition I made to the library was a QA Huggingface model wrapper. Previously, there existed a Huggingface model wrapper that successfully loads classification models from Huggingface. But this failed to properly load in SQuAD models to properly get outputs during an attack. Therefore, I added a new model wrapper that would be called specifically for cases where a SQuAD model is being attacked. The decision-making code was added to *model_args.py*. Similarly, the auto-model loader is also unique to each task, and we needed to add an extra condition to determine when to use a model loader for classification and when for QA.


Take Context, Question, Answer set
Apply White-Box Transformations to Contexts Based on Question
Get Answers for Questions based on Augmented Contexted
Check if new answers are different from ground truth answers

SQuAD Attack Recipe Workflow

Creating the attack recipe is the major design portion of my capstone project. As shown in the figure above, the overall workflow can be divided up into four steps. My previous discussion about TextAttack being calibrated for QA is what allows for me to successfully process and obtain the context, question, and answer set to begin with. With each of these examples, TextAttack will apply white-box transformations to contexts based on the question. Inspired from the previous related work, I also thought that perturbing only the context and not the question was a controlled way to execute attacks and see how easily the SQuAD model can be fooled. After augmenting the context with transformations, we run the model again to get an answer on the slightly altered context. If this new answer is any different from the original (ground truth) answer, then we know we have successfully created an adversarial example that fools the model.

In terms of the TextAttack recipe architecture, I defined a specific goal function, transformations, constraints, and a search method to best support an attack that would follow the workflow mentioned previously. Most of the components I implemented were functions and modules that previously existed in TextAttack that had been utilized by other attack recipes. To avoiding reinventing the wheel whenever possible, I reused the useful tools that previous researchers had implemented and were well-tested on. I initially wanted to use the minimize bleu score as my goal function to determine whether an answer was indeed different from the ground truth answer. This seemed most natural because bleu score determines how similar two sentences are to one another, and I can use this tool to evaluate how similar two answers are. If the bleu score was minimized to a certain threshold, that can be a good metric to determine that the answer is different enough to be considered a different answer. Unfortunately, the bleu score did not work well for short sentences, and many answer spans from SQuAD were around three words or less. Hence, I decided to pivot and design my own simpler goal function. This goal function was to detect a single word difference between two strings and was used under the reasonable assumption that if a single word varied from the new answer to the ground truth, then its fair to consider the two answers different.

For the transformations, I mainly used word swap functions to perturb the context. Specifically, synonym swapping. I also tested with word deletions and word insertions. Because of the transformations altering the original text, we also want to make sure that the overall meaning stays the same. This is when the constraints play a role, where I utilized the pre-built Part of Speech and Universal Sentence Encoder functionalities in TextAttack to preserve grammar and meaning. Lastly, I used a greedy search method mostly to make decisions efficient and fast, allowing attacks on SQuAD to be executed in a reasonable amount of time.

Results

When running TextAttack with a compatible graphics processing unit, my SQuAD attack can successfully attack 10 examples in around 10 minutes. Compared to classification tasks, SQuAD attacks are significantly slower because of the length of the context. In most classification tasks, an example only consists of a single sentence to attack. However, QA tasks require contexts that are the length of a paragraph, thus, performing transformations and searches over this large text space drastically increases the runtime of the attack.

When the goal function was set to minimizing the bleu score, many of the examples ended up getting skipped because many examples with short answer spans were giving faulty output from the bleu score function. However, when the goal was changed to be a single word difference, every single example was successfully attacked due to the simple and consistent nature of determining word differences between two strings, regardless of length. However, many of the attacks deemed successful by TextAttack are actually not quite successful. The main issue here is many attacks through random chance end up altering the answer span, which obviously changes the answer outputted by the mode and trivially satisfies the goal function. Another obvious false successful attack is when the perturbation effects the question, thus changing what the question is asking. Sometimes, entities referenced in the question are changed in the context, which also would be non-trivial to answering the question and will change the answer. In all these scenarios, the QA model itself is not doing anything wrong or being fooled, it is just given faulty contexts and questions that would actually change the validity of the ground truth answer.

Future Work

Time was a big factor that determined how much I could actually implement from my proposed solution. A lot of overhead and planning took place for the most of the semester, so when it came time to the actual execution of my solution, there was not a lot of weeks or meetings with the TextAttack group left before the final deliverable was due. I had planned to make my own custom transformations, such as preserving the answer span and only attacking the context. Right now, TextAttack attacks both the question and answer-span. Ideally, the attack should not attack the answer span nor should it attack the question. The goal of an attack is to get a different output when the given conditions of the input still result in the same output.

To preserve the question itself is quite simple. It only requires an addition to the *attacked_text.py* file and specifying what portion of the example to attack. In this case, QA examples contains both a context and question input. A condition to filter out the question field is all that is necessary to ensure the question is preserved. However, to preserve the answer span itself is a little more complex considering that it is located within the context to be perturbed. If I had more time, I had conceived a custom transformation where the starting and ending word index of the span are tracked. Before any transformations are applied to the context, the answer span is removed from the context. Then as transformations are applied, the start and end indices are updated accordingly. For example, if there was a word insertion or word deletion before the location of the answer span, the indices would increment or decrement respectively. For a word swap, nothing needs to be done since the location of the answer span would be constant. Once all transformations are applied, the answer is then added back to the context based on its start and end index. Thus, with this method, I can easily see the answer span being preserved.

The biggest research endeavor in the future is preserving the overall meaning of a context. This is related to the examples where important entities in the context are changed that alters the meaning of the context and invalidates the ground truth answer. A research study by Jack Morris also looks into the challenge of preserving semantics in perturbations, and how keeping meaning as a constraint is an underdeveloped research space. He found that perturbations often do not preserve semantics in most attacks, and 38% introduce grammatical errors. Human surveys reveal that to successfully preserve semantics, we need to significantly increase the minimum cosine similarities between the embeddings of swapped words and between the sentence encodings of original and perturbed sentences. With all of these adjusted constraints to better preserve semantics and grammaticality, attack success rates drop significantly (Morris, 2022). Hence, this is a definitely an important step in the future to overcome before we can completely master a design for attacking QA models.

In short, there is still an overall debate about how to define a true successful attack in the context of QA tasks. There may even be more constraints that I have yet to consider outside of the ones that I have mentioned previously. As of now, TextAttack has become compatible with QA models and datasets, and there is a skeleton QA attack recipe that will hopefully be a stepping-stone into creating more robust attacks on QA that will have a lot of applications in the QA research space.

References

Gardner, M., Berant, J., Hajishirzi, H., Talmor, A., & Min, S. (2019, September 25). *Question answering is a format; when is it useful?* arXiv.org. Retrieved May 6, 2022, from https://arxiv.org/abs/1909.11291

Guan, R. (n.d.). *Question Answering on Adversarial SQuAD 2.0*. Stanford CS224N. Retrieved May 6, 2022, from https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1204/

Morris, J. (2020). TextAttack basic functions. TextAttack Basic Functions - TextAttack 0.3.3 documentation. Retrieved October 17, 2021, from https://textattack.readthedocs.io/en/latest/0_get_started/basic-Intro.html.

Morris, J., Lifland, E., Lanchantin, J., Ji, Y., & Qi, Y. (2020, April 25). *Reevaluating Adversarial Examples in Natural Language*. Google Scholars. Retrieved May 6, 2022, from https://scholar.google.com/citations

Rahurkar, P., Olson, M., & Tadepalli, P. (n.d.). *Human Adversarial QA: Did the Model Understand the Paragraph?* Openreview. Retrieved May 5, 2022, from https://openreview.net/pdf?id=57NC-S7o4Aw

Rajpurkar, P., Jia, R., & Liang, P. (2018, June 11). *Know what you don't know: Unanswerable questions for squad*. arXiv.org. Retrieved May 6, 2022, from https://arxiv.org/abs/1806.03822

Rajpurkar, P., Zhang, J., Lopyrev, K., & Liang, P. (2016, October 11). *Squad: 100,000+ questions for machine comprehension of text*. arXiv.org. Retrieved May 6, 2022, from https://arxiv.org/abs/1606.05250