A Appendix

A.1 `TextAttack` in Five Lines or Less

Table 3 provides some examples of tasks that can be accomplished in bash or Python with five lines of code or fewer. Note that every action has to be prefaced with a single line of code (`pip install textattack`).

A.2 Components of `TextAttack`

This section explains each of the four components of the `TextAttack` framework and describes the components that are currently implemented. Figure 5 shows the decomposition of two popular attacks (Alzantot et al., 2018; Jin et al., 2019).

A.2.1 Goal Functions

A goal function takes an input x' and determines if it is satisfies the conditions for a successful attack in respect to the original input x. Goal functions vary by task. For example, for a classification task, a successful adversarial attack could be changing the model's output to be a certain label. Goal functions also scores how "good" the given x' is for achieving the desired goal, and this score can be used by the search method as a heuristic for finding the optimal solution.

`TextAttack` includes the following goal functions:

- Untargeted Classification: Minimize the score of the correct classification label.
- Targeted Classification: Maximize the score of a chosen incorrect classification label.
- Input Reduction (Classification): Reduce the input text to as few wordas as possible while maintaining the same predicted label.
- Non-Overlapping Output (Text-to-Text): Change the output text such that no words in it overlap with the original output text.
- Minimzing BLEU Score (Text-to-Text): Change the output text such that the BLEU score between it and the original output text is minimized (Papineni et al., 2001).

A.2.2 Constraints

A perturbed text is only considered valid if it satisfies each of the attack's constraints. `TextAttack` contains four classes of constraints.

Pre-transformation Constraints These constraints are used to preemptively limit how `x` can be perturbed and are applied before `x` is perturbed.

- Stopword Modification: stopwords cannot be perturbed.
- Repeat Modification: words that have been already perturbed cannot be perturbed again.
- Minimum Word Length: words less than a certain length cannot be perturbed.
- Max Word Index Modification: words past a certain index cannot be perturbed.
- Input Column Modification: for tasks such as textual entailment where input might be composed of two parts (e.g. hypothesis and premise), we can limit which part we can transform (e.g. hypothesis).

Overlap We measure the overlap between `x` and `x_adv` using the following metrics on the character level and require it to be lower than a certain threshold as a constraint:

- Maximum BLEU score difference (Papineni et al., 2001)
- Maximum chrF score difference (Popovic, 2015)
- Maximum METEOR score difference (Agarwal and Lavie, 2008)
- Maximum Levenshtein edit distance
- Maximum percentage of words changed

Grammaticality These constraints are typically intended to prevent the attack from creating perturbations which introduce grammatical errors. `TextAttack` currently supports the following constraints on grammaticality:

- Maximum number of grammatical errors induced, as measured by LanguageTool (Naber et al., 2003)
- Part-of-speech consistency: the replacement word should have the same part-of-speech as the original word. Supports taggers provided by flair, SpaCy, and NLTK.
- Filtering out words that do not fit within the context based on the following language models:
  - Google 1-billion words language model (Józefowicz et al., 2016)
  - Learning To Write Language Model (Holtzman et al., 2018) (as used by (Jia et al., 2019))
  - GPT-2 language model (Radford et al., 2019)

JTODO: adding a COLA model would be nice

Semantics Some constraints attempt to preserve semantics between `x` and `x_adv`. `TextAttack` currently provides the following built-in semantic constraints:

- Maximum swapped word embedding distance (or minimum cosine similarity)
- Minimum cosine similarity score of sentence rep-

	Task	Command
Run an attack	TextFooler on an LSTM trained on the MR sentiment classification dataset	`textattack attack --recipe textfooler --model bert-base-uncased-mr --num-examples 100`
	TextFooler against BERT fine-tuned on SST-2	`textattack attack --model bert-base-uncased-sst2 --recipe textfooler --num-examples 10`
	DeepWordBug on DistilBERT trained on the Quora Question Pairs paraphrase identification dataset:	`textattack attack --model distilbert-base-uncased-qqp --recipe deepwordbug --num-examples 100`
	seq2sick (black-box) against T5 fine-tuned for English-German translation:	`textattack attack --model t5-en-de --recipe seq2sick --num-examples 100`
	Beam search with beam width 4 and word embedding transformation and untargeted goal function on an LSTM:	`textattack attack --model lstm-mr --num-examples 20 --search-method beam-search:beam_width=4 --transformation word-swap-embedding --constraints repeat stopword max-words-perturbed:max_num_words=2 embedding:min_cos_sim=0.8 part-of-speech --goal-function untargeted-classification`
Data augmentation	Augment dataset from 'examples.csv' using the EmbeddingAugmenter, swapping out 4% of words, with 2 augmentations for example, withholding the original samples from the output CSV	`textattack augment --csv examples.csv --input-column text --recipe embedding --pct-words-to-swap 4 --transformations-per-example 2 --exclude-original`
	Augment a list of strings in Python	`from textattack.augmentation import EmbeddingAugmenter augmenter = EmbeddingAugmenter() s = 'What I cannot create, I do not understand.' augmenter.augment(s)`
Train a model	Train the default LSTM for 50 epochs on the Yelp Polarity dataset	`textattack train --model lstm --dataset yelp_polarity --batch-size 64 --epochs 50 --learning-rate 1e-5`
	Fine-tune bert-base on the CoLA dataset for 5 epochs	`textattack train --model bert-base-uncased --dataset glue:cola --batch-size 32 --epochs 5`
	Fine-tune RoBERTa on the Rotten Tomatoes Movie Review dataset, first augmenting each example with 4 augmentations produced by the EasyDataAugmentation augmenter	`textattack train --model roberta-base --batch-size 64 --epochs 50 --learning-rate 1e-5 --dataset rotten_tomatoes --augment eda --pct-words-to-swap .1 --transformations-per-example 4`
	Adversarially fine-tune DistilBERT on AG News using the HotFlip word-based attack, first training for 2 epochs on the original dataset	`textattack train --model distilbert-base-cased --dataset ag_news --attack hotflip --num-clean-epochs 2`

Table 3: With `TextAttack`, adversarial attacks, data augmentation, and adversarial training can be achieved in just a few lines of Bash or Python.

resentations obtained by well-trained sentence encoders:
  - Skip-Thought Vectors (Kiros et al., 2015)
  - Universal Sentence Encoder (Cer et al., 2018)
  - InferSent (Conneau et al., 2017)
  - BERT trained for semantic similarity (Reimers and Gurevych, 2019)
- Minimum BERTScore (Zhang* et al., 2020)

A.2.3 Transformations

A transformation takes an input and returns a set of potential perturbations. The transformation is agnostic of goal function and constraint(s): it returns all potential transformations.

We categorize transformations into two kinds: white-box and black-box. *White-box transforma-*

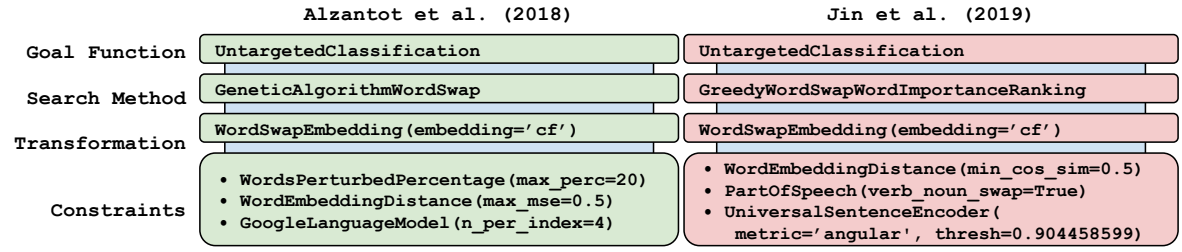

Figure 5: `TextAttack` builds NLP attacks from a goal function, search method, transformation, and list of constraints. This shows attacks from Alzantot et al. (2018) and Jin et al. (2019) created using `TextAttack` modules.

tions have access to the model and can query it or examine its parameters to help determine the transformation. For example, Ebrahimi et al. (2017) determines potential replacement words based on the gradient of the one-hot input vector at the position of the swap. *Black-box transformations* determine the potential perturbations without any knowledge of the model.

`TextAttack` currently supports the following transformations:

- Word swap with nearest neighbors in the counter-fitted embedding space (Mrkšić et al., 2016)
- WordNet word swap (Miller et al., 1990)
- Word swap proposed by a masked language model (Garg and Ramakrishnan, 2020; Li et al., 2020)
- Word swap gradient-based: swap word with another word in the vocabulary that maximize the model's loss (Ebrahimi et al., 2017) (white-box)
- Word swap with characters transformed (Gao et al., 2018):
  - Character deleted
  - Neighboring characters swapped
  - Random character inserted
  - Substituted with a random character
  - Character substituted with a homoglyph
  - Character substituted with a neighboring character from the keyboard (Pruthi et al., 2019)
- Word deletion
- Word swap with another word in the vocabulary that has the same Part-of-Speech and sememe, where the sememe is obtained by HowNet (Dong et al., 2006).
- Composite transformation: returns the results of multiple transformations

A.2.4 Search Methods

The search method aims to find a perturbation that achieves the goal and satisfies all constraints. Many combinatorial search methods have been proposed for this process. `TextAttack` has implemented a selection of the most popular ones from the literature:

- **Greedy Search with Word Importance Ranking.** Rank all words according to some ranking function. Swap words one at a time in order of decreasing importance.

- **Beam Search.** Initially score all possible transformations. Take the top b transformations (where b is a hyperparameter known as the "beam width") and iterate, looking at potential transformations for all sequences in the beam.

- **Greedy Search.** Initially score transformations at all positions in the input. Swap words, taking the highest-scoring transformations first. (This can be seen as a case of beam search where $b = 1$).

- **Genetic Algorithm.** An implementation of the algorithm proposed by Alzantot et al. (2018). Iteratively alters the population through greedy perturbation of each population member and crossover between population numbers, with preference to the more successful members of the population. (We also support an alternate version, the "Improved Genetic Algorithm" proposed by Wang et al. (2019)).

- **Particle Swarm Optimization.** A population-based evolutionary computation paradigms (Kennedy and Eberhart, 1995) that exploits a population of interacting individuals to iteratively search for the optimal solution in the specific space (Zang et al., 2020). The population is called a *swarm* and individual agents are called *particles*. Each particle has a position in the search space and moves with an adaptable *velocity*.

A.3 `TextAttack` Attack Reproduction Results

Table 4 displays a comparison of results achieved when running attacks in `TextAttack` alongside numbers reported in the original paper. All `TextAttack` benchmarks were run on pre-trained models provided by the library and can be reproduced in a single `textattack attack` command. There are a few important implementation differences:

- The genetic algorithm benchmark comes from the faster genetic algorithm of (Jia and Liang, 2017). As opposed to the original algorithm of (Alzantot et al., 2018), this implementation uses a fast language model, so it can query contexts of up to 5 words. Additionally, perplexity is compared to that of the original word, not the previous perturbation. Since these are more rigorous linguistic constraints, a lower attack success rate is expected.
- The LSTM models from BAE (Garg and Ramakrishnan, 2020) were trained using counter-fitted GLoVe embeddings. The LSTM models from `TextAttack` were trained using normal GLoVe embeddings. Our models are consequently less robust to counter-fitted embedding synonym swaps, and a higher attack success rate is expected.
- The HowNet synonym set used in `TextAttack`'s PSO implementation is a concatenation of the three synonym sets used in the paper. This is necessary since `TextAttack` is dataset-agnostic and cannot expect to provide a set of synonyms for every possible dataset. Since the attack has more synonyms to choose from, `TextAttack`'s PSO implementation is slightly more successful.

A.4 `TextAttack` Attack Prototypes

This section displays "attack prototypes" for each attack recipe implemented in `TextAttack`. This is a concise way to print out the components of a given attack along with its parameters. These are directly copied from the output of running `TextAttack`.

Alzantot Genetic Algorithm (Alzantot et al., 2018)

```
Attack(
  (search_method): GeneticAlgorithm(
    (pop_size): 60
    (max_iters): 20
    (temp): 0.3
    (give_up_if_no_improvement): False
  )
  (goal_function): UntargetedClassification
  (transformation): WordSwapEmbedding(
    (max_candidates): 8
    (embedding_type): paragramcf
  )
  (constraints):
    (0): MaxWordsPerturbed(
        (max_percent): 0.2
        (compare_against_original): True
      )
    (1): WordEmbeddingDistance(
        (embedding_type): paragramcf
        (max_mse_dist): 0.5
        (cased): False
        (include_unknown_words): True
        (compare_against_original): False
      )
    (2): GoogleLanguageModel(
        (top_n): None
        (top_n_per_index): 4
        (compare_against_original): False
      )
    (3): RepeatModification
    (4): StopwordModification
    (5): InputColumnModification(
        (matching_column_labels): ['premise', 'hypothesis']
        (columns_to_ignore): {'premise'}
      )
  (is_black_box): True
)
```

Alzantot Genetic Algorithm (faster) (Jia et al., 2019)

```
Attack(
  (search_method): GeneticAlgorithm(
    (pop_size): 60
    (max_iters): 20
    (temp): 0.3
    (give_up_if_no_improvement): False
  )
  (goal_function): UntargetedClassification
  (transformation): WordSwapEmbedding(
    (max_candidates): 8
    (embedding_type): paragramcf
  )
  (constraints):
    (0): MaxWordsPerturbed(
        (max_percent): 0.2
      )
    (1): WordEmbeddingDistance(
        (embedding_type): paragramcf
        (max_mse_dist): 0.5
        (cased): False
        (include_unknown_words): True
      )
    (2): LearningToWriteLanguageModel(
        (max_log_prob_diff): 5.0
      )
    (3): RepeatModification
    (4): StopwordModification
  (is_black_box): True
)
```

BAE (Garg and Ramakrishnan, 2020)

```
Attack(
  (search_method): GreedyWordSwapWIR(
    (wir_method): delete
  )
  (goal_function): UntargetedClassification
  (transformation): WordSwapMaskedLM(
    (method): bae
    (masked_lm_name): bert-base-uncased
    (max_length): 256
    (max_candidates): 50
  )
  (constraints):
    (0): PartOfSpeech(
```

		LSTM				BERT-Base				
		MR	**SST-2**	**IMDB**	**AG**	**MR**	**SST-2**	**IMDB**	**SNLI**	**AG**
`alzantot` (Alzantot et al., 2018)	Reported	-	-	97.0 / 14.7	-	-	-	-	-	-
	`TextAttack`	64.6 / 17.8	70.8 / 18.3	73.0 / 4.0	27.7 / 11.6	40.7 / 19.1	46.5 / 20.7	46.7 / 7.3	74.9 / 12.3	18.1 / 12.6
`bae` (Garg and Ramakrishnan, 2020)	Reported	70.2 / -	-	73.2 / -	-	48.3 / -	-	45.9 / -	-	-
	`TextAttack`	74.4 / 12.3	72.7 / 13.5	88.8 / 2.6	21.4 / 6.3	61.5 / 15.2	66.6 / 14.5	55.6 / 3.2	78.4 / 7.1	16.9 / 7.4
`deepwordbug` (Gao et al., 2018)	Reported	-	-	-	72.5 / -	-	-	-	-	-
	`TextAttack`	86.3 / 16.8	82.6 / 17.1	97.6 / 5.2	83.4 / 19.4	78.2 / 21.2	81.3 / 18.9	80.9 / 5.3	99.0 / 9.8	60.7 / 25.1
`pso` (Zang et al., 2020)	Reported	-	93.8 / 9.1	100.0 / 3.7	-	-	91.2 / 8.2	98.7 / 3.7	78.9 / 11.7	-
	`TextAttack`	94.9 / 10.7	96.5 / 11.5	100.0 / 1.3	83.7 / 12.7	92.7 / 11.9	91.3 / 12.9	100.0 / 1.2	91.8 / 6.2	79.4 / 16.7
`textfooler` (Jin et al., 2019)	Reported	96.2 / 14.9	-	99.7 / 5.1	95.8 / 18.6	86.7 / 16.7	-	85.0 / 6.1	95.5 / 18.5	86.7 / 22.0
	`TextAttack`	97.4 / 13.6	98.8 / 14.2	100.0 / 2.4	95.3 / 17.2	88.7 / 18.7	94.8 / 16.9	100.0 / 7.2	96.3 / 7.2	79.5 / 23.5

Table 4: Comparison between our re-implemented attacks and the original source code in terms of success rate (left number) and percentage of perturbed words (right number). Numbers that are not found in the literature are marked as "-". 1000 samples are randomly selected for evaluation from all these datasets except IMDB (100 samples are used for IMDB since some attack methods like Genetic and PSO take over 4 days to finish 1000 samples).

```
        (tagger_type): nltk
        (tagset): universal
        (allow_verb_noun_swap): True
        (compare_against_original): True
      )
    (1): UniversalSentenceEncoder(
        (metric): cosine
        (threshold): 0.936338023
        (window_size): 15
        (skip_text_shorter_than_window): True
        (compare_against_original): True
      )
    (2): RepeatModification
    (3): StopwordModification
  (is_black_box): True
)
```

BERT-Attack (Li et al., 2020)

```
Attack(
  (search_method): GreedyWordSwapWIR(
    (wir_method): unk
  )
  (goal_function): UntargetedClassification
  (transformation): WordSwapMaskedLM(
    (method): bert-attack
    (masked_lm_name): bert-base-uncased
    (max_length): 256
    (max_candidates): 48
  )
  (constraints):
    (0): MaxWordsPerturbed(
        (max_percent): 0.4
        (compare_against_original): True
      )
    (1): UniversalSentenceEncoder(
        (metric): cosine
        (threshold): 0.2
        (window_size): inf
        (skip_text_shorter_than_window): False
        (compare_against_original): True
      )
    (2): RepeatModification
    (3): StopwordModification
  (is_black_box): True
)
```

DeepWordBug (Gao et al., 2018)

```
Attack(
  (search_method): GreedyWordSwapWIR(
    (wir_method): unk
  )
  (goal_function): UntargetedClassification
  (transformation): CompositeTransformation(
    (0): WordSwapNeighboringCharacterSwap(
        (random_one): True
      )
    (1): WordSwapRandomCharacterSubstitution(
        (random_one): True
      )
    (2): WordSwapRandomCharacterDeletion(
        (random_one): True
      )
    (3): WordSwapRandomCharacterInsertion(
        (random_one): True
      )
    )
  (constraints):
    (0): LevenshteinEditDistance(
        (max_edit_distance): 30
        (compare_against_original): True
      )
    (1): RepeatModification
    (2): StopwordModification
  (is_black_box): True
)
```

HotFlip (Ebrahimi et al., 2017)

```
Attack(
  (search_method): BeamSearch(
    (beam_width): 10
  )
  (goal_function): UntargetedClassification
  (transformation): WordSwapGradientBased(
    (top_n): 1
  )
  (constraints):
    (0): MaxWordsPerturbed(
        (max_num_words): 2
        (compare_against_original): True
      )
    (1): WordEmbeddingDistance(
        (embedding_type): paragramcf
        (min_cos_sim): 0.8
        (cased): False
        (include_unknown_words): True
        (compare_against_original): True
      )
    (2): PartOfSpeech(
        (tagger_type): nltk
        (tagset): universal
        (allow_verb_noun_swap): True
        (compare_against_original): True
      )
    (3): RepeatModification
    (4): StopwordModification
  (is_black_box): False
)
```

Input Reduction (Feng et al., 2018)

```
Attack(
  (search_method): GreedyWordSwapWIR(
    (wir_method): delete
  )
  (goal_function): InputReduction(
    (maximizable): True
  )
  (transformation): WordDeletion
  (constraints):
    (0): RepeatModification
    (1): StopwordModification
  (is_black_box): True
)
```

Kuleshov (Kuleshov et al., 2018)

```
Attack(
  (search_method): GreedySearch
  (goal_function): UntargetedClassification
  (transformation): WordSwapEmbedding(
    (max_candidates): 15
    (embedding_type): paragramcf
  )
  (constraints):
    (0): MaxWordsPerturbed(
        (max_percent): 0.5
        (compare_against_original): True
      )
    (1): ThoughtVector(
        (embedding_type): paragramcf
        (metric): max_euclidean
        (threshold): −0.2
        (window_size): inf
        (skip_text_shorter_than_window): False
        (compare_against_original): True
      )
    (2): GPT2(
        (max_log_prob_diff): 2.0
        (compare_against_original): True
      )
    (3): RepeatModification
    (4): StopwordModification
  (is_black_box): True
)
```

MORPHEUS (Tan et al., 2020)

```
Attack(
  (search_method): GreedySearch
  (goal_function): MinimizeBleu(
    (maximizable): False
    (target_bleu): 0.0
  )
  (transformation): WordSwapInflections
  (constraints):
    (0): RepeatModification
    (1): StopwordModification
  (is_black_box): True
)
```

Particle Swarm Optimization (Zang et al., 2020)

```
Attack(
  (search_method): ParticleSwarmOptimization
  (goal_function): UntargetedClassification
  (transformation): WordSwapHowNet(
    (max_candidates): −1
  )
  (constraints):
    (0): RepeatModification
    (1): StopwordModification
    (2): InputColumnModification(
        (matching_column_labels): ['premise', 'hypothesis']
        (columns_to_ignore): {'premise'}
      )
  (is_black_box): True
)
```

Pruthi Keyboard Char-Swap Attack (Pruthi et al., 2019)

```
Attack(
  (search_method): GreedySearch
  (goal_function): UntargetedClassification
  (transformation): CompositeTransformation(
    (0): WordSwapNeighboringCharacterSwap(
        (random_one): False
      )
    (1): WordSwapRandomCharacterDeletion(
        (random_one): False
      )
    (2): WordSwapRandomCharacterInsertion(
        (random_one): False
      )
    (3): WordSwapQWERTY
    )
  (constraints):
    (0): MaxWordsPerturbed(
        (max_num_words): 1
        (compare_against_original): True
      )
    (1): MinWordLength
    (2): StopwordModification
    (3): RepeatModification
  (is_black_box): True
)
```

PWWS (Ren et al., 2019)

```
Attack(
  (search_method): GreedyWordSwapWIR(
    (wir_method): pwws
  )
  (goal_function): UntargetedClassification
  (transformation): WordSwapWordNet
  (constraints):
    (0): RepeatModification
    (1): StopwordModification
  (is_black_box): True
)
```

seq2sick (Cheng et al., 2018)

```
Attack(
  (search_method): GreedyWordSwapWIR(
    (wir_method): unk
  )
  (goal_function): NonOverlappingOutput
  (transformation): WordSwapEmbedding(
    (max_candidates): 50
    (embedding_type): paragramcf
  )
  (constraints):
    (0): LevenshteinEditDistance(
        (max_edit_distance): 30
        (compare_against_original): True
      )
    (1): RepeatModification
    (2): StopwordModification
  (is_black_box): True
)
```

TextBugger (Li et al., 2019)

```
Attack(
  (search_method): GreedyWordSwapWIR(
    (wir_method): unk
  )
  (goal_function): UntargetedClassification
  (transformation): CompositeTransformation(
    (0): WordSwapRandomCharacterInsertion(
        (random_one): True
      )
    (1): WordSwapRandomCharacterDeletion(
        (random_one): True
      )
    (2): WordSwapNeighboringCharacterSwap(
        (random_one): True
      )
    (3): WordSwapHomoglyphSwap
    (4): WordSwapEmbedding(
        (max_candidates): 5
        (embedding_type): paragramcf
      )
    )
  (constraints):
    (0): UniversalSentenceEncoder(
        (metric): angular
        (threshold): 0.8
        (window_size): inf
        (skip_text_shorter_than_window): False
        (compare_against_original): True
      )
    (1): RepeatModification
    (2): StopwordModification
  (is_black_box): True
)
```

TextFooler (Jin et al., 2019)

```
Attack(
  (search_method): GreedyWordSwapWIR(
    (wir_method): del
  )
  (goal_function): UntargetedClassification
  (transformation): WordSwapEmbedding(
    (max_candidates): 50
    (embedding_type): paragramcf
  )
  (constraints):
    (0): WordEmbeddingDistance(
        (embedding_type): paragramcf
        (min_cos_sim): 0.5
        (cased): False
        (include_unknown_words): True
        (compare_against_original): True
      )
    (1): PartOfSpeech(
        (tagger_type): nltk
        (tagset): universal
        (allow_verb_noun_swap): True
        (compare_against_original): True
      )
    (2): UniversalSentenceEncoder(
        (metric): angular
        (threshold): 0.840845057
        (window_size): 15
        (skip_text_shorter_than_window): True
        (compare_against_original): False
      )
    (3): RepeatModification
    (4): StopwordModification
    (5): InputColumnModification(
        (matching_column_labels): ['premise', 'hypothesis']
        (columns_to_ignore): {'premise'}
      )
  (is_black_box): True
)
```