End-to-End Reinforcement Learning for Black-Box Adversarial Text Generation

Austin Nguyen [*1] **Shreyas Chandrashekaran** [*1] **Joan Nwatu** [*1]

Abstract

Adversarial attacks significantly reduce the accuracy of language models by making them consistently produce erroneous predictions when given text input that has been altered on a character, word, or sentence level. With the ubiquity of language models in today's world, it is increasingly important to codify approaches to both attacking and defending such models to understand a future threat model. In our work, we formulate the task of creating adversarial examples as a reinforcement learning problem. We train an RL model to fool a SOTA pre-trained RoBERTa-Base language model finetuned on the AG news classification dataset with the aim of achieving high attack performance comparable to existing attack methods on this task. While we fall short of this goal, we suggest possible reasons for our outcomes and future directions.

1. Introduction

The field of adversarial machine learning originated in computer vision applications. Research like Szegedy et al. (2014) and Goodfellow et al. (2015) show that the addition of small imperceptible perturbations to image inputs causes misclassification in deep learning models for computer vision, showing that these models lack robustness and are in fact quite brittle. Most adversarial work has focused on computer vision due to both the continuous nature of image modalities and the clear visual perceptibility of adversarial vision results.

In the field of NLP, Jia & Liang (2017) investigated the robustness of models for natural language processing using adversarial examples. Adversarial approaches to NLP must necessarily be formulated differently from those used in computer vision due to the differences between the nature of input in both modalities (Zhang et al., 2019). Image data are represented by pixel values which are continuous, while textual data are discrete in nature. Therefore, while perturbed images manifested real image outputs, perturbed textual representations yield invalid or incoherent text outputs. To overcome these limitations, NLP adversarial examples are created using perceptible changes to text inputs in the form of character-level, word-level, or sentence-level alterations.

Adversarial examples for language models have been proven to exist (Wang et al., 2022) (Morris et al., 2020) (Jin et al., 2019) and have been shown to lower the performance of these models. This area has grown significantly in research interest as language models become commonplace in today's enterprise environments (Liu et al., 2023).

In our work, we train a Reinforcement Learning (RL) model using the Double Deep-Q Network (DDQN) algorithm first presented by Hasselt et al. (2016) to produce adversarial examples of the AG news dataset (Zhang et al., 2015) to cause misclassification on a high-performing news topic categorization model. By applying budget constraints such that we limit our model to only making a certain number of target queries, we test the efficiency of our method in different setups, attempting to ameliorate the black-box adversarial NLP shortcoming of high model querying cost.

Our results show that we are able to achieve limited success with multiple iterations of training, but that our model fails to generalize to unseen data, suggesting that no positional replacement pattern exists or that further training is necessary.

2. Related Work

The AG news corpus introduced by Zhang et al. (2015) has been used by many applications, especially text classification. It has previously been used in both adversarial attacks by Yang et al. (2023) and defenses such as Wang et al. (2020).

Mastering this corpus for various NLP functions remains a central task toward a unified understanding of language

[*]Equal contribution [1]Department of Computer Science Engineering, University of Michigan, Ann Arbor, MI, United States of America. Correspondence to: Shreyas Chandrashekaran <shreyasc@umich.edu>, Joan Nwatu <jnwatu@umich.edu>, Austin Nguyen <ngaustin@umich.edu>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023.

since news is a notoriously difficult domain to parse, owing to its to specialized vocabulary. Previous works such as Yang et al. (2023) focused primarily on preserving semantics by minimizing modification cost, which they did by prioritizing both replaceable words and candidate replacements. They achieve quite high attack success rates – in fact, higher than all other adversaries in this domain, so we mainly compare our methods to theirs. Instead of building a custom reward function and focusing on modification cost, we use reinforcement learning to learn a generalizable method to generate adversarial examples.

Other works such as Alzantot et al. (2018) generate adversarial examples for sentiment analysis by picking the top *k* similar embeddings based on a particular embedding framework and selecting the replacement that results in the highest accuracy of the incorrect class. This results in *k* calls to the downstream model with all possible candidates to ascertain the highest-scoring selection. While we do select synonyms in a similar way to this work, our replacement selection process differs.

In terms of reinforcement learning approaches to adversarial example generation, Wong (2017) used the Sequence to Sequence (Seq2Seq) model (Sutskever et al., 2014) and REINFORCE algorithm (Sutton et al., 1999) to rewrite input text to both preserve semantic similarity and force incorrect classification on the simpler spam classification task. Buck et al. (2017) used a similar approach to reformulate questions to boost the performance of question-answering models, finding that reformulated questions by this method are quite different from traditional NLP paraphrases.

Maimon & Rokach (2022) is very similar to our approach in terms of the use of reinforcement learning to select the best adversarial examples. However, this work focuses on the task of creating adversarial examples using a universal adversarial policy for movie review classification and reports an Attack Success Rate (ASR) of 0.3 on a hand-picked subset of examples, so it is difficult to produce a one-to-one comparison between our findings and theirs.

3. Approach

We opt to use RoBERTa-Base, one of the BERT family of models (Devlin et al., 2019), and the DDQN algorithm (Hasselt et al., 2016) in our work. Our approach models adversarial text generation as a Markov Decision Process, where states are characterized by the currently perturbed sentence. Actions are defined discretely, where each action corresponds to switching one of the words in the sentence with a different one. Rewards are structured to encourage the agent to find adversarial texts in the minimal number of actions, or word swaps, which is the main difference between our approach and that of Jin et al. (2019). We next outline each of the aspects of the algorithm in detail.

3.1. Word Candidate Generation

For each input sentence, we define each action as swapping out one of the words in the sentence. More specifically, given a set of legal actions $\{a_i\}_{i=1}^{n+1}$, a_i corresponds with switching out the i^{th} swappable word in the sentence. We also insert an action a_{n+1} that corresponds to a "Stop" action. This corresponds to exiting the MDP and returning the current sentence s_t as an adversarial example. Note that a_i does not necessarily correspond to the i^{th} word in the sentence. Some words in the sentence are not considered candidates to be swapped out. More specifically, we keep a list of stop words considered ineligible as candidates. These stop words consist of various pronouns, prepositions, articles, or any tokens deemed insignificant for natural language processing.

We have shown how to determine which words in the sentence are swappable. The next step is to show how to retrieve word candidates to swap each word with. The eligible words are passed into an embedding space using Counter-Fitted Glove Embeddings (Mrkšić et al., 2016). Then, we retrieve the *N* nearest neighbors in embedding space for each of the words to generate a finite number of synonyms for each word. These words are the candidates for swapping.

We also insert an additional constraint when selecting possible words to replace. When retrieving the nearest neighbors, we choose to only select a certain number of words that have minimal overlap in characters with the original word. This prevents the replacement word from being simply the original word in a different tense. In our experiments, we conduct experiments with and without this additional constraint.

The entire word candidate generation process is shown in Algorithm 3.1.

Algorithm 1 Word Candidate Generation

for *each word* w *in sentence* s *not in stop word set* $\mathcal{S}$ **do**
  if $w \in \mathcal{S}$ **then**
    l = []
  else
    Create list l = []
    Get word embedding w' for w
    select list c of N nearest neighbors to w'
    for *each candidate in* c **do**
      if *overlap(c,w)* $<$ *allowed overlap* **then**
        append c to l
return Dictionary $\{w$: list $l\}$

3.2. Reward Function

The reward is a function of the state, action, and the next state: $r(s, a, s')$. The reward function is intended to prioritize three aspects: getting closer to an adversarial example, finding an adversarial example, and using the least number of actions. Therefore, we create a reward function that is a linear combination of these three aspects. First, we define r_{score} as the difference between the original-label logit of the target model between the two sentences:

$$r_{score} = T(s) - T(s') \tag{1}$$

where T is a function mapping an input sentence to the target model's output ground label logit for that sentence.

Next, we define λ_s as a scalar reward for successfully finding an adversarial sentence and is 0 if not. This can only be achieved by taking the action a_{stop} in the state s where s is an adversarial example. Lastly, $\lambda_{constant}$ is a small, negative scalar that penalizes the agent for taking too many steps to be successful. The total reward function is summarized below, where λ_{score} is a hyperparameter.

$$r(s, a, s') = \lambda_{score} r_{score} + \lambda_{constant} + \lambda_s \tag{2}$$

3.3. Model Architecture

The model architecture for our reinforcement learning approach is shown in Figure 1. After standard preprocessing and cleaning, input sentences are tokenized and converted into GloVe embeddings (Pennington et al., 2014). We then obtain the candidate words and convert them to their embeddings correspondingly. We introduce a list of indicators, a one-hot vector of length equal to the number of tokens in the sentence used to match sentence-level tokens to swappable candidates. Suppose the first nonzero value is located at the j^{th} position of the indicator vector – this communicates that the first candidate word passed in is associated with the j^{th} token in the sentence.

The sentence and candidate embeddings are then passed into an LSTM. The outputs from each of the words in the sentence and candidates are passed into separate fully connected layers, with the addition of the indicator vector. Then, the entire output is concatenated and passed through a final Multi-Layer Perceptron (MLP).

3.4. Reinforcement Learning Algorithm

The reinforcement learning model takes a string representing the current sentence state, the swappable word candidates, and indicators telling the network which words correspond to which indices in the sentence. In order to solve the MDP, we decide to use the DDQN algorithm to optimize the rewards shown in Equation 2. A training loop is first executed with a set number of training sentence examples to transform. After a certain number of epochs E of training, an evaluation loop is constructed where we measure the attack's ASR on both the training set and additional, unseen testing data.

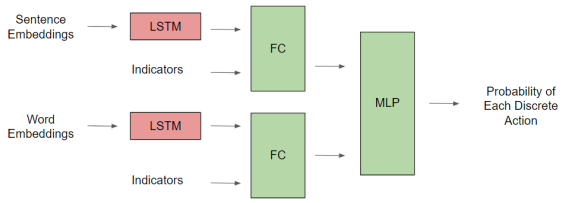

Figure 1. The model architecture consists of two LSTM modules, respectively taking the sentence embeddings and word embeddings, two fully connected layers that take in the indicator variables, and a multi-layer perception to output a probability distribution over actions. In the case of the Q-network, the outputs correspond to the Q-values for each action, where an ε -greedy policy is used.

The DDQN algorithm is outlined in Algorithm 2.

3.5. Adding Constraints

We also wanted to experiment with constraining the number of words we were allowed to swap within each sentence. The implementation previously discussed simply allowed the policy to switch out as many words as was allowed by the sentence when limited by the list of stop words. However, we wanted to see if constraining the action space to only $L(s)$ actions would achieve similar results with the added benefit of lower cost. The number of actions $L(s)$ is a function of the original sentence. We let L be a hyperparameter denoting the maximum number of swappable words and define $L(s) = max(L, num\_swappable(s))$.

Next, we show how we select the $L(s)$ swappable words from s. We take inspiration from Jin et al. (2019), and mask each of the swappable tokens $s[i]$ with the $< unk >$ token one at a time. This results in $num\_swappable(s)$ sentences. We then query each of these sentences into the target model and gather the ground truth label's logit changes. We then take the top *L* swappable words that had the highest absolute effects on the logits. The formal approach is outlined in Algorithm 3.

4. Experiments and Results

4.1. Effects of Excessive Training

One of the first difficulties of training was the susceptibility of the model to plummet in performance after a certain number of training epochs. This is not an unheard-of issue

Algorithm 2 DDQN Training Algorithm

Input: Input sentence s_0, Q-network Q_θ, Target Q-network $Q_{\theta'}$, Empty buffer $\mathcal{B}$, discount factor γ, target update frequency T

for *number of epochs* **do**
  for *each original sentence* s_0^i **do**
    for *each timestep t* **do**
      $\mathcal{W}_t^i$ = WordCandidates(s_t^i) from Algorithm 3.1
      $\mathcal{I}_i$ = Indicators(s_t^i)
      s_e, w_e = Embeddings(s, $\mathcal{W}$)
      a_t = EpsilonGreedy($Q_\theta(s_e, w_e, \mathcal{I})$)
      Apply a_t to $s_t^i \rightarrow s_{t+1}^i$
      $r_t = r(s_t, a_t, s_{t+1})$ from Equation 2
      d = is 1 if a_t == STOP
      Store transition $(s_t^i, a_t, r_t, s_{t+1}^i, d)$

      if *Update the model* **then**
        Sample batch $b = (s, a, r, s', d) \sim \mathcal{B}$
        $Q_{tar} = r + \gamma Q_{\theta'}(s', argmax_a Q_\theta(s', a))$
        $Q_{pred} = Q_\theta(s, a)$
        $J(\theta)$ = MSE(Q_{pred}, Q_{tar})
        $\theta = \theta - \nabla J(\theta)$

      if *Update target model every* T **then**
        $\theta' = \theta$

Algorithm 3 Adding Constraints

Input: options = empty maxheap (sorted by value), sentence s, target model $\mathcal{M}$

for *swappable token s in swappable token set* **do**
  s_m = sentence, mask s with unk
  f = $\mathcal{M}(s_m)$ - $\mathcal{M}(s)$
  add (key=s, value=$|f|$) to options
return *options[:N]*

when training deep reinforcement learning models, but the brittleness was particularly noticeable here. In Figure 2, we show how there is a drastic drop in both rewards and ASR after around 220 epochs of training. Note that these metrics are reported on the training dataset itself.

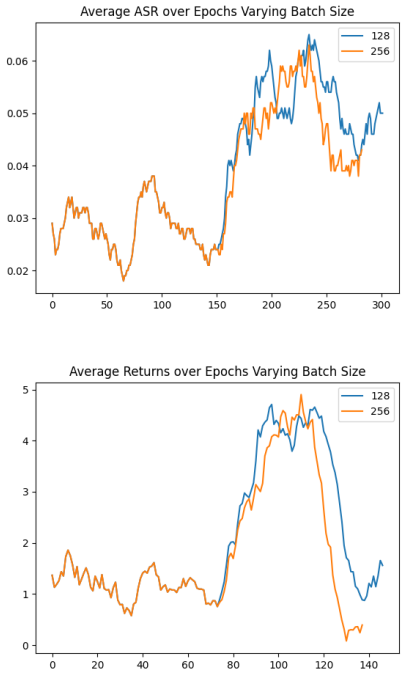

Figure 2. Training curves for different batch sizes on 50 training examples for 400 epochs. The ASR is averaged over 20 epochs while the rewards are averaged over 100. The results highly justify stopping training early.

As we will show later on, this instability is likely due to the large, discrete action space of the task. The model is provided a mask on the output logits to know which actions are valid. This is needed as some sentences may have fewer swappable words than the number of output actions from the model. Even then, some sentences with a large number of swappable words face an issue where, as training progresses, the distribution of actions becomes highly concentrated on a small subset of actions. This is simply due to the convergence of the algorithm and annealing of the epsilon exploration. However, this shift in distribution makes other, less likely chosen actions to be increasingly inaccurate in their Q-value predictions. This problem is exacerbated by a large action space. As a result, this causes training instability towards the end as gradients for less likely transitions are far higher than more likely ones.

All subsequent experiments have been tuned so that the algorithm halts at the optimal epoch. The data from tuning each of the experiments is omitted from the report for brevity.

Hyperparameters	
Epochs	Varies
Learning Rate	3e-4
Gamma	.975
Batch Size	128
Max Length Rollout	250
Warm Up Steps	2e5
Update Target Frequency	5000
Max Buffer Size	3e5
Epsilon Initial	1
Epsilon Final	.1
Epsilon Steps	1e5
λ_{score}	100
$\lambda_{constant}$	-.02
λ_s	10
LSTM Hidden Layer Size	128
LSTM Num Hidden Layers	1
LSTM Output Size	200
MLP Width	200
Optimizer	Adam

Figure 3. Hyperparameters used for all experiments unless otherwise specified.

4.2. Without Budget Constraints

Here, we show the results of the algorithm on a training set of 50 input sentences from the Roberta-base AG News dataset. We also performed a parameter search in order to find the most optimal training parameters in terms of training time and reward garnered. The optimal parameters found are enumerated in Figure 3.

In Figure 4 we show the training trajectory and final performance of the adversarial agent. Note that this agent uses sentence transformations where each word is the nearest neighbor to the original word's embedding. Future sections will go over results when we enforce additional characteristics of the replacement word as described in Section 3.5.

We note a strong positive correlation between rewards per episode and the attack success rate over training epochs. This indicates that the agent's optimization over the rewards successfully entails better sentence example generation, assuming adversarial examples are possible given the word candidates. It is difficult to say whether the relatively low ASR can be attributed to the learning process or to the word candidate choices themselves. We investigate this in a future section. As shown in the previous section, training was incredibly fickle. The training trend likely would have plummeted shortly after stopping it early.

The final results for the two runs are shown in Figure 5. The run where the batch size of 128 had a stronger final performance than that of the batch size of 64. This is likely because of lower variance updates to the Q-network policy.

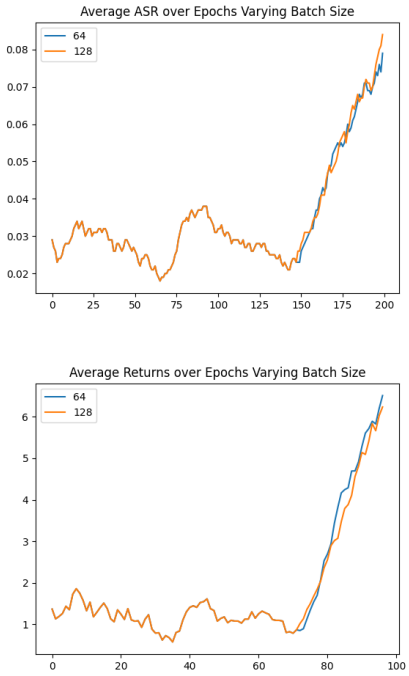

Figure 4. Training curves for different batch sizes on 50 training examples for 220. The ASR is averaged over 20 epochs while the rewards are averaged over 100. We cut off training at 220 epochs in order to prevent the aforementioned plummet in performance.

However, even then, we only reach an ASR of 12%. Even if there are signs of learning, this low ASR can be attributed to two different things. Firstly, the method by which we are choosing the word candidates. As elaborated earlier, we still have yet to remove candidates that are far too similar to the original words. The RL method is only as powerful as the word options it is given, so this would be a promising way to improve performance. Furthermore, it is possible that the action space is currently too large to efficiently explore. Adding budget constraints as mentioned in Section 4.5 would be helpful.

We also provide a few example perturbations that the policy generates in Figure 4.2. Note that many of the generated

Testing Results		
Batch Size	64	128
Original Accuracy	98%	98%
Accuracy Under Attack	90%	86%
ASR	8.16%	12.24%
Average Perturbed Word %	12.84%	13.23%

Figure 5. Testing result after varying the batch size between 64 and 128 and training on 50 examples for 220 epochs. The final testing set is identical to the training set in this experiment.

Example 1

Original Sentence: Teenage T. rex's monster growth **Tyrannosaurus** rex **achieved** its massive size due to an **enormous** growth spurt during its adolescent years.

Perturbed Sentence: Teenage T. rex's monster growth **Dinosaur** rex **attained** its massive size due to an **immense** growth spurt during its adolescent years.

Example 2

Original Sentence: Rivals Try to Turn Tables on Charles Schwab By MICHAEL LIEDTKE SAN FRANCISCO (AP) – With its **low** prices and iconoclastic attitude, discount stock broker Charles Schwab Corp. (SCH) represented an **annoying** stone in Wall Street's wing-tipped shoes for decades...

Perturbed Sentence: Rivals Try to Turn Tables on Charles Schwab By MICHAEL LIEDTKE SAN FRANCISCO (AP) – With its **scant** prices and iconoclastic attitude, discount stock broker Charles Schwab Corp. SCH represented an **pesky** stone in Wall Street's wing-tipped shoes for decades...

Figure 6. Examples of successfully perturbed texts. While swapping words with ones with similar meanings may seem promising, it may not preserve grammatical correctness.

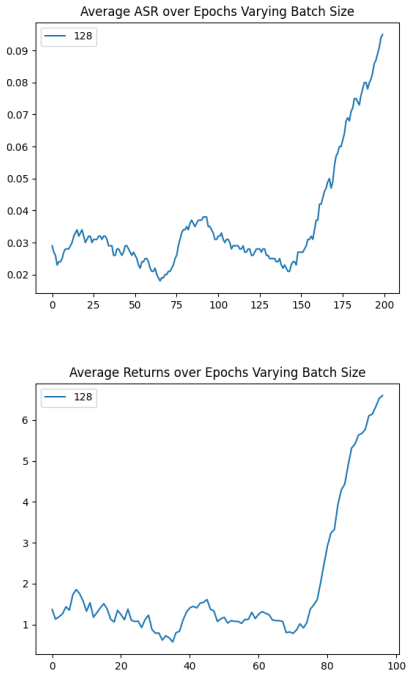

Figure 7. Training curves for the experiment where we train for the evaluation on unseen data. ASR is averaged over 10 epochs while returns are averaged over 100.

adversarial examples were intended to preserve semantic meaning. However, does not imply preservation of grammatical structure nor correctness. Swapping at the word level, without consideration of surrounding context, can yield sentences that, although the individual word may be similar, are no longer grammatically accurate.

4.3. Testing on Unseen data

It is important to note that the previous results are still showing performance on the training data; all metrics on reward and ASR have been with respect to already seen data. As the purpose of this method is to see whether reinforcement learning can be used to find patterns that may generate adversarial examples for a particular model, it is vital we separate the given data into training and testing environments.

As a result, we split the dataset as follows: 50 training sentences and 70 testing sentences, where the latter includes the training sentences as well. In other words, there are 20 sentences that were unseen in the training process. We show the training curves of this approach on the test set in Figure 7 along with the final performance in Figure 8.

Unfortunately, our approach is unable to generalize to unseen data. While it is a common problem that deep reinforcement learning methods tend to overfit their given data, we believe there must be some mechanism that allows for the RL agent to detect patterns within the data to best exploit new sentence examples. Future work that could address this issue would be to introduce auxiliary tasks that force the reinforcement learning agent to recognize token-level patterns. Another way would be to introduce a different state representation to better capture the patterns within each sentence. While this seems like we are giving the model additional information, it may aid its ability to generalize past training examples.

4.4. Getting Better Word Candidates

We next test the approach's efficacy by modifying the word candidate generation process. As noted in previous sections,

Testing Results	
Batch Size	128
Original Accuracy	98%
Accuracy Under Attack	90%
ASR	8.7%
Average Perturbed Word %	13.86%

Figure 8. We show the final performance of the trained policy on the 70 evaluation examples. The 70 examples include 50 training examples and 20 unseen examples. The policy is unable to generalize past its training examples.

up until now we have been using the nearest neighbor to the original word's embedding to use as a swapping candidate. However, what commonly happens is this generates the original word in a different tense. This small perturbation is less likely to significantly alter the sentence's classification label. As result, we modify the approach by, for each word in the sentence, generating a list of neighboring words and taking the neighbor that shares the least number of characters with the original word. The results for this modification are shown in Figure 9.

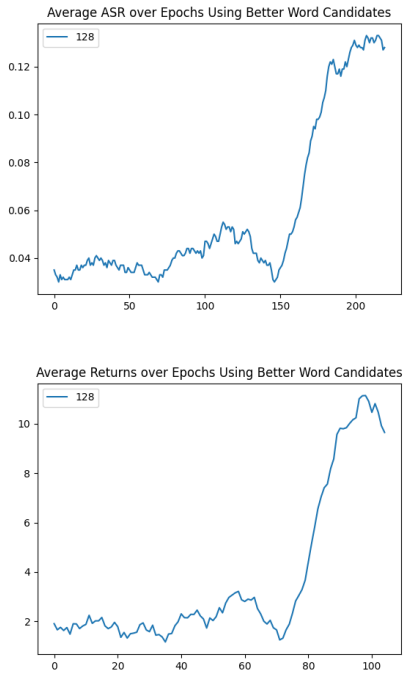

Figure 9. Training curves for the experiment where we apply additional constraints on the word candidate generation. We see that performance is stronger by a commendable margin.

It is evident that the ASR is higher than when we used the previous approach to word candidate generation. This can be attributed to the higher likelihood of generating a holistically different sentence from the original, increasing the chance of changing the output label.

Testing Results	
Original Accuracy	98%
Accuracy Under Attack	82%
ASR	16.33%
Average Perturbed Word %	20.29%

Figure 10. We show the final performance of the trained policy on the 50 training examples when using the revised word candidate generation method. We see higher ASR than when using the original method.

What is interesting to note is that the average perturbed word is notably higher in this approach than when using the previous word candidate generation method. This might come as a surprise, because we would expect that, with words that are more dissimilar, each word would yield more effect on the output label. However, if we look at the reward function, we encourage the RL agent to swap out words *only if* an adversarial example is possible. If it is not possible, then the reward function encourages the agent to use the "stop" action quickly to prevent itself from consistently receiving $\lambda_{constant}$. Since having better word candidates makes it more likely for adversarial examples to be possible given an input sentence, it is less likely to use the "stop" action earlier.

Due to time constraints, we were unable to construct ablation tests on the number of neighbors that we generated for each word. This would be a promising direction to pursue in future work.

4.5. With Budget Constraints

Lastly, we test what happens when we insert constraints on the number of words we could swap out in each example sentence. We construct experiments where we vary the number of c words we were allowed to swap out. Note that c could be greater than the number of swappable words in a given sentence. Therefore, we ensure that the used value $c_{applied}$ on a given input sentence s is $c_{applied}(s) = min(c, num\_swappable\_words(s))$. We show training curves for various values of c in Figure 11. The final testing metrics are also reported in Figure 12.

We notice that having less constraint (meaning a higher value for c) provided a higher attack success rate far quicker. However, having a value of 10 for c achieved a higher reward after a certain amount of epochs. This return difference may simply be due to a smaller action space. We also combine the constraint with the generation of stronger word candidates. This yields far stronger performance than all previous approaches so far. It is promising that the combination of the two revisions yields stronger results. Future work would be geared towards tuning these parameters and analyzing their effects on performance further.

4.6. Discussion

While our results were not extremely promising for this method, we still believe that RL-based approaches can perform comparably to other works at adversarial example generation. We attribute our poor performance to a few possible factors. Primarily, the choice of dataset and limit of swappable words hampers our adversarial example generation, as we are only able to generate valid adversarial examples slightly more than 10% of the time. The median length of the input is 45 words and the median number of swappable

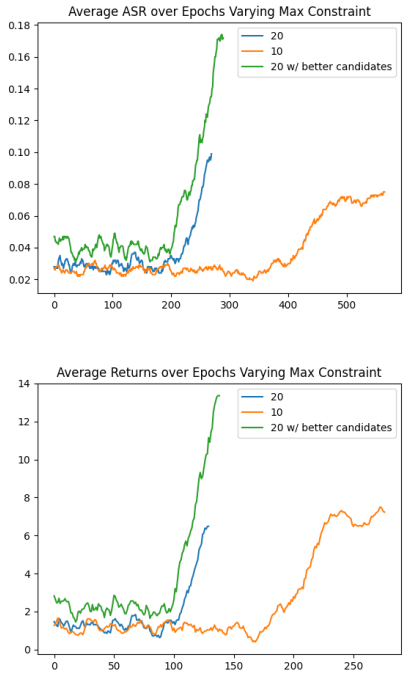

Figure 11. The training curves with varying values for c and toggling whether we use the revised word candidate generation algorithm.

Testing Results			
	c = 10	c = 20	c = 20 w/ cands
Original Accuracy	98%	98%	98%
Accuracy Under Attack	91.84%	89.8%	81.63%
ASR	8.16%	10.2%	18.37%

Figure 12. We show the final performance of the trained policy on the 50 training examples when we vary the max constraint and whether or not we use the word candidate generation method.

words is 20; compared to related approaches, this number is quite low and therefore suggests a reasonable explanation for a lack of valid misclassifications. This seems an artifact of the data itself, as the dataset was originally conceived to be used in character-level situations and therefore includes examples that are quite short. In fact, we were unable to swap a single word in at least one example. Further, since we use a limited replacement set, it is distinctly possible that certain words lack replacements that meaningfully force mis-classifications in a relevant direction, especially since we have a 4-class downstream task as opposed to the typical binary task. These issues combined with the lack of model generalization seem to be the main cause of our results.

5. Conclusion and Future Work

In this paper, we approach adversarial text generation as a reinforcement learning task. Using a Markov Decision process, we define the states and actions to be the sentence to be perturbed and the swapping of words in the sentence, respectively. We then formulate our rewards to teach our agent to create adversarial examples using the least possible actions.

While our results show limited success in the different setups we staged for our experiment, we point out several factors that could be responsible. We suggest that further work on different datasets or with better constraints for swappable words could potentially yield better results.

References

Alzantot, M., Sharma, Y., Elgohary, A., Ho, B.-J., Srivastava, M., and Chang, K.-W. Generating natural language adversarial examples. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2890–2896, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1316. URL https://aclanthology.org/D18-1316.

Buck, C., Bulian, J., Ciaramita, M., Gesmundo, A., Houlsby, N., Gajewski, W., and Wang, W. Ask the right questions: Active question reformulation with reinforcement learning. *CoRR*, abs/1705.07830, 2017. URL http://arxiv.org/abs/1705.07830.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

Goodfellow, I. J., Shlens, J., and Szegedy, C. Explaining and harnessing adversarial examples, 2015.

Hasselt, H. v., Guez, A., and Silver, D. Deep reinforcement learning with double q-learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, pp. 2094–2100. AAAI Press, 2016.

Jia, R. and Liang, P. Adversarial examples for evaluating reading comprehension systems, 2017.

Jin, D., Jin, Z., Zhou, J. T., and Szolovits, P. Is BERT really robust? natural language attack on text classification and entailment. *CoRR*, abs/1907.11932, 2019. URL http://arxiv.org/abs/1907.11932.

Langley, P. Crafting papers on machine learning. In Langley, P. (ed.), *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pp. 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.

Liu, P., Zhang, L., and Gulla, J. A. Pre-train, prompt and recommendation: A comprehensive survey of language modelling paradigm adaptations in recommender systems, 2023.

Maimon, G. and Rokach, L. A universal adversarial policy for text classifiers. *Neural Networks*, 153:282–291, sep 2022. doi: 10.1016/j.neunet.2022.06.018. URL https://doi.org/10.10162Fj.neunet.2022.06.018.

Morris, J. X., Lifland, E., Yoo, J. Y., and Qi, Y. Textattack: A framework for adversarial attacks in natural language processing. *CoRR*, abs/2005.05909, 2020. URL https://arxiv.org/abs/2005.05909.

Mrkšić, N., Séaghdha, D. , Thomson, B., Gašić, M., Rojas-Barahona, L., Su, P.-H., Vandyke, D., Wen, T.-H., and Young, S. Counter-fitting word vectors to linguistic constraints, 2016.

Pennington, J., Socher, R., and Manning, C. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1162. URL https://aclanthology.org/D14-1162.

Sutskever, I., Vinyals, O., and Le, Q. V. Sequence to sequence learning with neural networks, 2014.

Sutton, R. S., McAllester, D., Singh, S., and Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. In Solla, S., Leen, T., and Müller, K. (eds.), *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 1999. URL https://proceedings.neurips.cc/paper_files/paper/1999/file/464d828b85b0bed98e80ade0a5c43b0f-Paper.pdf.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. Intriguing properties of neural networks, 2014.

Wang, X., Yang, Y., Deng, Y., and He, K. Adversarial training with fast gradient projection method against synonym substitution based text attacks, 2020.

Wang, X., Wang, H., and Yang, D. Measure and improve robustness in nlp models: A survey, 2022.

Wong, C. Dancin seq2seq: Fooling text classifiers with adversarial text example generation, 2017. URL https://arxiv.org/abs/1712.05419.

Yang, X., Liu, W., Bailey, J., Tao, D., and Liu, W. Semantic-preserving adversarial text attacks, 2023.

Zhang, W. E., Sheng, Q. Z., Alhazmi, A., and Li, C. Adversarial attacks on deep learning models in natural language processing: A survey, 2019.

Zhang, X., Zhao, J., and LeCun, Y. Character-level convolutional networks for text classification. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, pp. 649–657, Cambridge, MA, USA, 2015. MIT Press.