Neural machine translation systems tend to fail on less decent inputs despite its significant efficacy, which may significantly harm the credibility of these systems—fathoming how and when neural-based systems fail in such cases is critical for industrial maintenance. Instead of collecting and analyzing bad cases using limited handcrafted error features, here we investigate this issue by generating adversarial examples via a new paradigm based on reinforcement learning. Our paradigm could expose pitfalls for a given performance metric, e.g., BLEU, and could target any given neural machine translation architecture. We conduct experiments of adversarial attacks on two mainstream neural machine translation architectures, RNN-search, and Transformer. The results show that our method efficiently produces stable attacks with meaning-preserving adversarial examples. We also present a qualitative and quantitative analysis for the preference pattern of the attack, demonstrating its capability of pitfall exposure.

1 Introduction

Neural machine translation (NMT) based on the encoder-decoder framework, such as RNN-Search (Bahdanau et al., 2014; Luong et al., 2015, RNNSearch) or Transformer (Vaswani et al., 2017, Transformer), has achieved remarkable progress and become a de-facto in various machine translation applications. However, there are still pitfalls for a well-trained neural translation system, especially when applied to less decent real-world inputs compared to training data (Belinkov and Bisk, 2017). For example, typos may severely deteriorate system outputs (Table 1). Moreover, recent studies show that a neural machine translation system can also be broken by noisy synthetic inputs (Belinkov and Bisk, 2017; Lee et al., 2018). Due to the black-box nature of a neural system, it has been a challenge to fathom when and how the system tends to fail.

Intuitively, researchers seek to apprehend such failures by the analysis of handcrafted error indicating features (Zhao et al., 2018; Karpukhin et al., 2019). This strategy is costly because it requires expert knowledge for both linguistics and the target neural architecture. Such features are also less applicable because some common errors in deep learning systems are hard to formulate, or very specific to certain architectures.

Instead of designing error features, recent researchers adopt ideas from adversarial learning (Goodfellow et al., 2014) to generate adversarial examples for mining pitfalls of NLP systems (Cheng et al., 2018a, Ebrahimi et al., 2018; Zhao et al., 2017). Adversarial examples are minor perturbed inputs that keep the semantic meaning, yet yield degraded outputs. The generation of valid adversarial examples provides tools for error analysis that is interpretable for ordinary users, which can contribute to system maintenance. Though it has achieved success concerning continuous input, e.g., images, there are following major issues for NLP tasks.

First, it is non-trivial to generate valid discrete tokens for natural language, e.g., words or characters. Cheng et al. (2018a) follow Goodfellow et al. (2014) to learn noised representation then sample tokens accordingly. However, there is no guaranteed correspondence between arbitrary representation and valid tokens. Therefore, it may gen-
Two man are playing on the street corner.

Zwei Männer spielen an einer Straßenecke.

Table 2: Example of undesirable perturbation in adversarial examples for machine translation in (Zhao et al., 2017), though it yields very different output compare to the origin, it does not indicate system malfunction.

● Our paradigm can achieve the adversarial example generation with outclassed efficiency by only given source data. Since our method is model-agnostic and free of handcrafted error feature targeting architectures, it is also viable among different machine translation models.

● We also present some analysis upon the state-of-the-art Transformer based on its attack, showing our method’s competence in system pitfall exposure.

2 Preliminaries

2.1 Neural Machine Translation

The most popular architectures for neural machine translation are RNN-search (Bahdanau et al., 2014) and Transformer (Vaswani et al., 2017). They share the paradigm to learn the conditional probability $P(Y|X)$ of a target translation $Y = [y_1, y_2, ..., y_m]$ given a source input $X = [x_1, x_2, ..., x_n]$. A typical NMT architecture consists of an encoder, a decoder and attention network. The encoder encodes the source embedding $X_{emb} = [emb_1, emb_2, ..., emb_n]$ into hidden representation $H = [h_1, h_2, ..., h_n]$. Then a decoder $f_{dec}$ with attention network attentively accesses $H$ for an auto-regressive generation of each $y_i$ until the end of sequence symbol (EOS) is generated:

$$P(y_i|y_{<i}, X) = \text{softmax}(f_{dec}(y_{i-1}, s_t, c_t; \theta_{dec}))$$

(1)

where $c_t$ is the attentive result for current decoder state $s_t$ given $H$.

2.2 Actor-Critic for Reinforcement Learning

Reinforcement learning (Sutton and Barto, 2018, RL) is a widely used machine learning technique following the paradigm of explore and exploit, which is apt for unsupervised policy learning in many challenging tasks (e.g., games (Mnih et al., 2015)). It is also used for direct optimization for non-differentiable learning objectives (Wu et al., 2018; Bahdanau et al., 2016) in NLP.

Actor-critic (Konda and Tsitsiklis, 2000) is one of the most popular RL architectures where the agent consists of a separate policy and value networks called actor and critic. They both take in environment state $s_t$ at each time step as input, while...
actor determines an action $a_t$ among possible action set $\mathcal{A}$ and critic yields value estimation $V_t(s_t)$.

In general, the agent is trained to maximize discounted rewards $R_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$ for each state, where $\gamma \in (0, 1]$ is the discount factor. Such goal can be further derived as individual losses applied to actor and critic. Thus the actor policy loss $L^\pi$ on step $t$ is:

$$L^\pi_t(\pi) = \log P(a_t|s_t)A_t(s_t, a_t); a_t \in \mathcal{A}$$

where $\pi$ denotes actor parameters, $A_t(s_t, a_t)$ denotes general advantage function (Schulman et al., 2015) on state $s_t$ for action $a_t$ given by

$$\sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k V(s_{t+k}) - V(s_t),$$

which can be further derived as:

$$A_t(s_t, a_t) = \gamma A_{t+1}(s_{t+1}, a_{t+1}) + r_t + \gamma V_{t+1}(s_{t+1}) - V_t(s_t)$$

Meanwhile, critic learns to estimate $R_t$ via minimizing a temporal difference loss $L^\pi$ on each step $t$:

$$L^\pi_t(\pi_c) = \frac{1}{2}(r_t + \gamma R_{t+1} - V_t(s_t))^2$$

where $\pi_c$ denotes critic parameter.

Usually, the training is regularized by maximizing policy entropy $H^\pi$ to avoid exploration failure before exploiting optimum policy (Ziebart, 2010). Thus the total loss becomes:

$$L(\theta) = \sum_t (\alpha L^\pi_t - L^\pi_t - \beta H^\pi(s_t))$$

where $\alpha$ and $\beta$ are hyperparameters for value loss and entropy coefficients.

### 2.3 Adversarial examples in NLP

A general adversarial example generation can be described as the learning process to find a perturbation $\delta$ on input $X$ that maximize system degradation $L_{adv}$ within a certain constraint $C(\delta)$:

$$\arg\max_{\delta} L_{adv}(X + \delta) - \lambda C(\delta)$$

where $\lambda$ denotes the constraint coefficient, $L_{adv}$ is determined by the goal of the attack. However, currently effective adversarial generation for NLP is to search by maximizing a surrogate gradient-based loss:

$$\arg\max_{1 \leq i \leq n, x \in \text{vocab}} L_{adv}(x_0, x_1, ... x_i, ... x_n)$$

where $L_{adv}$ is a differentiable function indicating the adversarial object. Due to its formidable search space, this paradigm simply perturbs on a small ratio of token positions and greedy search by brute force among candidates. Note that adversarial example generation is fundamentally different from noised hidden representation in adversarial training (Cheng et al., 2019; Sano et al., 2019), which is not to be concerned in this work.

### 3 Approach

In this section, we will describe our reinforced learning and generation of adversarial examples (Figure 1) in detail. Overall, the victim model is a part of the environment (denoted as $Env$), which yields rewards indicating overall degradation based on modified inputs. A reinforced agent...
learns to modify every source position from left to right sequentially. Meanwhile, a discriminator in \( Env \) provides every-step survival signals by determining whether SRC is ill-perturbed.

### 3.1 Environment

We encapsulate the victim translation model with a discriminative reward process as an \( Env \) for a reinforced agent to interact.

#### 3.1.1 Environment State

The state of the \( Env \) is described as \( s_t = (SRC, t) \), where \( SRC = [src_0, src_1, ..., src_N] \) are \( N \) sequences processed by victim model’s vocabulary and tokenization. Each sequence \( src_i = [x_1, x_2, ..., x_n] \) is concatenated with BOS, EOS, which indicate the begin and end of the sequence, then padded to same length. Time step \( t \in [1, n] \) also indicates the token position to be perturbed by the agent. \( Env \) will consecutively loop for all token positions and update \( s_t \) based on the agent’s modification. \( Env \) also yields reward signals until the end or intermediately terminated. That is, all sequences in \( SRC \) are determined by \( D \) as ill-perturbed during the reward process. Once the \( Env \) terminates, it finishes the current episode and reset its state with a new batch of sequences as \( SRC \).

#### 3.1.2 Reward Process with Discriminator

The reward process is only used during training. It consists of a survival reward \( r_s \) on every step and a final degradation \( r_d \) concerning an overall metric if the agent survives till the end. Overall, we have:

\[
 r_t = \begin{cases} 
 -1, & \text{terminated} \\
 \frac{1}{N} \sum_N a \cdot r_s, & \text{survive} \land t \in [1, n] \\
 \frac{1}{N} \sum_N (a \cdot r_s + b \cdot r_d), & \text{survive} \land t = n 
\end{cases}
\]

(8)

where \( a, b \) are hyper parameters that keeps the overall \( r_s \) and \( r_d \) within similar magnitude.

Instead of direct optimization of the constrained adversarial loss in Eq.6, we model discriminator \( D \)'s output as survival rewards similar to that in gaming (Mnih et al., 2015). That is, the agent must survive for its goal by also fooling \( D \), which attempts to terminate ill-perturbed modifications. We define an ill-perturbed source by determining whether it still matches the original target \( tgt \).

**Discriminator** As it is shown in Figure 1(b), discriminator \( D \) consists of bi-directional GRU encoders for both source and target sequence. Their corresponding representation is averaged and concatenated before passed to a feedforward layer with dropout. Finally, the output distribution is calculated by a softmax layer. Once \( D \) determines the pair as positive, its corresponding possibility is regarded as the reward, otherwise 0:

\[
 r_s = \begin{cases} 
 P(\text{positive}|(src', tgt); \theta_d), & \text{positive} \\
 0, & \text{otherwise} 
\end{cases}
\]

(9)

As long as the environment survives, it yields averaged reward among samples from \( SRC \) (Eq.8) to mitigate rewards’ fluctuation that destabilize training.

**Discriminator Training** Similar to GAN training, the environment’s \( D \) must update as the agent updates. During its training, the agent’s parameter is freezeed to provide training samples. For every \( D \)'s training epoch, we randomly choose half of the batch and perturb its source using the current agent as negative samples. During \( D \)'s updates, we randomly generate a new batch of pairs from parallel data likewise to test its accuracy. \( D \) is updated at most \( step_D \) epochs, or until its test accuracy reaches \( acc_{bound} \).

\( Env \) only\(^1\) yields -1 as overall terminal rewards when all sequences in \( SRC \) are intermediately terminated. For samples classified as negative during survival, their follow-up rewards and actions are masked as 0. If the agent survives until the end, \( Env \) yields additional averaged \( r_d \) as final rewards for an episode. We follow Michel et al. (2019) to adopt relative degradation:

\[
 r_d = \frac{\text{score}(y, \text{refs}) - \text{score}(y', \text{refs})}{\text{score}(y, \text{refs})}
\]

(10)

where \( y \) and \( y' \) denote original and perturbed output, \( \text{refs} \) are references, and \( \text{score} \) is a translation metric. If \( \text{score}(y, \text{refs}) \) is zero, we return zero as \( r_d \). To calculate score we retokenize perturbed \( SRC \) by victim models vocabulary and tokenizer before translation.

\(^1\)It is commonly accepted that frequent negative rewards result in agents’ tendency to regard zero-reward as optimum and fail exploration, which further leads to training failure.
3.2 Agent

As it is shown in Figure 1 (c), the agent’s actor and critic share the same input layers and encoder, but later processed by individual feedforward layers and output layers. Actor takes in \( SRC \) and current token with its surrounding \((x_{t-1}, x_t, x_{t+1})\), then yields a binary distribution to determine whether to attack a token on step \( t \), while critic emits a value \( V(s_t) \) for every state. Once the actor decides to perturb a specific token, this token will be replaced by another token in its candidate set.

Candidate Set We collect at most \( K \) candidates for each token in the victim’s vocabulary within a distance of \( \epsilon \). \( \epsilon \) is the averaged Euclidean distance of \( K \)-nearest embedding for all tokens in victim vocabulary. We note that there shall always be candidates for a token in test scenarios that are beyond victim’s vocabulary, for those without a nearby candidate, we assign UNK as its candidate. Once the agent chooses to replace a token with UNK, we follow Michel et al. (2019) to present a valid token that is also UNK to the victim’s vocabulary.

Agent Training The agent is trained by algorithm in appendix A. Since the agent is required to explore with stochastic policy during training, it will first sample based on its actor’s output distribution on whether to perturb the current position, then randomly choose among its candidates. The agent and discriminator take turns to update. We assume the training is converged when test accuracy for \( D \) does not reach over a certain value within certain continuous learning rounds of agent and discriminator.

Agent Inference To generate adversarial examples, the agent will take in source sequences and perturb on each position based on the actor’s output from left to right, then choose the nearest candidate. As the agent’s critic learns to estimate expected future rewards for a step, only when it yields positive value will agent perturb, otherwise it indicates an undesirable perturbation; thus, the agent is muted.

4 Experiments

4.1 Data Sets

We test our adversarial example generations on Zh→En, En→Fr, and En→De translation tasks, which provide relatively strong baselines for victim models and mass test samples.

We train our agent using only parallel data that is used for victims’ training. We train on LDC Zh→En\(^2\) (1.3M pairs), WMT14 En→De\(^3\) (4.5M pairs) and WMT15 En→Fr\(^4\) (2.2M pairs) for victim models respectively. For subword level translation, we apply byte pair encoding (Sennrich et al., 2015, BPE) for both source and target languages with the vocabulary size of 37k. We also use join-BPE for En-De and En-Fr experiments with 34k and 33k vocabulary size, respectively. For word-level translation, we use NLPIR-ICTCLAS and Moses tokenizer for Chinese and English tokenization, respectively. We adopt 30k as vocabulary size for both source and target language. We adopt NIST test sets \(^5\) for Zh→En and WMT test sets for En→De and En→Fr, then generate adversarial examples for these sources for analysis.

4.2 Victim Models

We choose the state-of-the-art RNN-search and Transformer as victim translation models. For RNN-search, we train subword level models and strictly follow the architecture in Bahdanau et al. (2014). As for Transformer, we train both word-level and subword-level model for Zh→En and only subword-level models for En→De and En→Fr with the architecture and the base parameter settings by Vaswani et al. (2017). For the above models, we apply the same batch scheme and Adam optimizer following Vaswani et al. (2017). We choose MT03, newsdiscuss2015 and newstest2013 for Zh→En, En→Fr, En→De as validation set respectively.

4.3 Metrics

We first report attack results both in terms of char-level BLEU (chrBLEU) of perturbed source by the origin to indicate modification rate, and relative decrease in target BLEU (RD):

\[
RD = \frac{\text{BLEU}(y, ref_s) - \text{BLEU}(y', ref_s)}{(1 - \text{chrBLEU}(x', x)) \times \text{BLEU}(y, ref_s)}
\]

We adopt sacreBLEU (Post, 2018) to test case-insensitive BLEU on detokenized targets.

\(^2\) ldc2002E18, ldc2003E14, ldc2004T08, ldc2005T06
\(^3\) https://nlp.stanford.edu/projects/nmt/
\(^4\) Europarl-v7, news-commentary-v10
\(^5\) MT02, 03, 04, 05, 06
As Michel et al. (2019) suggest, there is a tradeoff between achieving high RD and maintaining semantic. One can achieve rather high RD by testing with mismatched references, making degradation less meaningful. Therefore, we also test source semantic similarity with human evaluation (HE) ranging from 0 to 5 used by Michel et al. (2019) by randomly sampling 10% of total sequences mixed with baselines for a double-blind test.

### 4.4 Results

We implement state-of-the-art adversarial example generation by gradient search (Michel et al., 2019) (GS) as a baseline, which can be currently applied to various translation models. We also implemented random synthetic noise injection (Karpukhin et al., 2019) (RSNI) as an unconstrained contrast. Both baselines are required to provide a ratio for the amount of tokens to perturb during an attack, where we present the best results. Unlike our paradigm can generate on monolingual data, GS also requires target annotations, where we use one of the references to provide a strong baseline. Note that RSNI can significantly break semantics with distinctly lower HE to achieve rather high RD, which we do not consider as legit adversarial example generation and noted with “∗” for exclusion.

As it is shown in Table 3 and 4, our model stably generate adversarial examples without significant change in semantics by the same training setting among different models and language pairs, achieving stably high HE (≥3.7) without any handcrafted semantic constraints, while search methods (GS) must tune for proper ratio of modification, which can hardly strike a balance between semantic constraints and degradation. Unlike search paradigm relying on reference and victim gradients, our paradigm is model-agnostic yet still achieving comparable RD with relatively high HE.

### 5 Analysis

#### 5.1 Efficiency

As it is shown in Figure 2, given the same amount of memory cost, our method is significantly more
Table 5: (a) an example of perturbed number and quantifier severely damaging outputs in Zh→En translation, where we highlight the changes. “五” is the character for “5”, “七” for “7”, “名” and “位” are both commonly used quantifiers for people. However, search-based attack achieves degradation by some significant changes of semantics, where number “16” is changed to “6”, and “外交部长” means “foreign minister”. (b) an example of changed suffix which breaks the result. “方” and “者” are common suffixes (K) sharing same meaning used for people. Our model spots that victim model’s fragility upon such perturb, while search does not.

5.2 Attack Patterns

NMT systems may have different robustness over different parts of the inputs, thus some researchers implement input preprocessing targeting certain empirically weak parts, e.g., named entities (Li et al., 2018). Since the agent’s policy is to attack without handcrafted error features, we can further investigate vulnerability by its attack preferences of different parts of speech. We choose Chinese, for example, and adopt LTP POS tagger to label NIST test sets, then check the modification rate for each POS. To ensure the reliability of our analysis, we run three rounds of experiments on both baselines and our agent with similar modification rate targeting state-of-the-art Transformer with BPE, and collect overall results. We also present random synthetic noise injection (Karpukhin et al., 2019) (RSNI), which is not intended for any preference as an additional baseline.

As it is shown in Figure 3, our reinforced paradigm shows distinct preference upon certain POS tags, indicating pitfalls of a victim translation system. At the same time, RSNI distributed almost evenly upon different POS tags. Though the search paradigm (GS) does expose some types of pitfall, our method can further expose those omitted by the search. Note that unlike existing work relying on feature engineering to indicate errors, we have no such features implemented for an agent. However, our agent can still spot error patterns by favoring some of the POS, such as

Figure 2: Time consumption of different methods: we limit memory usage to 2.5G on single Nvidia 1080, and generate adversarial examples for the same 800 inputs in Zh→En MT with different methods, our method significantly outclasses the state-of-the-art search paradigm (GS).
Figure 3: Attack preferences of different paradigms targeting Zh→En Transformer-BPE model. All share a similar modification rate. Our agent shows a significant preference for some POS (e.g., Ni, Nh, Nz, I), which are commonly regarded as hard-to-translate phrases among industrial implementations, while some (e.g., K) are less noticed. Preference among different choices.

<table>
<thead>
<tr>
<th>Attack by</th>
<th>BLEU(Δ)</th>
<th>Zh-En MT02-06</th>
</tr>
</thead>
<tbody>
<tr>
<td>RNN-search-BPE</td>
<td>-</td>
<td>40.90</td>
</tr>
<tr>
<td></td>
<td>agent-RNN</td>
<td>31.58(-9.32)</td>
</tr>
<tr>
<td></td>
<td>agent-TF</td>
<td>32.14(-8.76)</td>
</tr>
<tr>
<td>Transformer-BPE</td>
<td>-</td>
<td>44.06</td>
</tr>
<tr>
<td></td>
<td>agent-TF</td>
<td>35.48(-8.58)</td>
</tr>
<tr>
<td></td>
<td>agent-RNN</td>
<td>33.14(-10.92)</td>
</tr>
</tbody>
</table>

Table 6: Attacks targeting different architecture from the trained one. We note agent with the architecture that is trained with (e.g., agent-RNN stands for agent trained by targeting RNN-search).

Ni (organization name), Nh (person name), Nl (location name), M (numbers), which are commonly accepted as hard-to-translate parts. Moreover, the agent also tends to favor K (suffix) more, which is less noticed.

5.3 Attack Generalization

We additionally test agents by attacking different model architecture from the one that it’s trained. As it is shown in Table 6, we perturb the inputs by agents trained to attack a different architecture, then test for degradation. The results show that our agent trained by targeting Transformer architecture can still achieve degradation on RNN-search, and vice-versa.

<table>
<thead>
<tr>
<th>Clean test</th>
<th>Noisy test</th>
<th>IWSLT11-17</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transformer-BPE</td>
<td>44.06</td>
<td>35.48</td>
</tr>
<tr>
<td>Tuned</td>
<td>43.60(-0.46)</td>
<td>40.31(+4.83)</td>
</tr>
</tbody>
</table>

Table 7: Tuning Zh→En Transformer-BPE model with adversarial examples. We generate adversarial examples for every training sources for tuning, achieving overall improvements for noisy tests.

5.4 Tuning with Adversarial Examples

Since the agent generates meaning-preserving adversarial examples efficiently, we can directly tune the original model with those samples. We choose Zh→En Transformer-BPE, for example, and generate the same amount of adversarial examples given original training sources (1.3M pairs), then paired with initial targets. We mix the augmented pairs with original pairs for a direct tuning. We test the tuned model on original test data and noisy test data generated by the attacking agent. We additionally test on IWSLT11-17 Zh→En test data, which is not used for training or tuning, to verified robustness improvement. As Table 7 shows, our agent can achieve substantial improvement (+4.83) on noisy tests with only minor loss on clean tests (-0.46). The improvement on the IWSLT test also indicates the adversarial tuning contributes to not only defending the agent’s attack, but also overall robustness.

5.5 Reinforced Examples for Machine translation

We additionally switched the episodic rewards in the environment, then ignored all modifications that induce UNK tokens to train an agent, hoping to generate minor perturbed samples that can improve the translation metric. Though we failed to achieve overall improvements, we do succeed for quite a portion of samples, as shown in Table 8. Similar to adversarial examples, we call them reinforced examples. Such improvement is different from adversarial training that tunes model
Table 8: Example of minor perturbed samples that improves machine translation for Zh→En Transformer-BPE model. The “" in first sample is modified to “"”, then model yields the omitted “钱中琛 (qian qi chen)”. The “停泊” in second sample is modified to “停车”, where they both mean “parking”, then comes the omitted “in southern district” for “在南区”.

### 6 Related Work

Cheng et al. (2018a) and Cheng et al. (2018b) applied continuous perturbation learning on token’s embedding and then manage a lexical representation out of a perturbed embedding. Zhao et al. (2017) learned such perturbation on the encoded representation of a sequence, and then decode it back as an adversarial example. These methods are applicable for simple NLP classification tasks, while failing machine translation which requires higher semantic constraints. Zhao et al. (2017) further attempted to constrain semantic in such paradigm by introducing multi-task modeling with accessory annotation, which further limits applicability.

On the other hand, Ebrahimi et al. (2018), Chaturvedi et al. (2019) and Cheng et al. (2019) regarded it as a search problem by maximizing surrogate gradient losses. Due to the formidable gradient computation, such methods are less viable to more complex neural architectures. Cheng et al. (2019) introduced a learned language model to constrain generation. However, a learned language model is not apt for common typos or UNK. Another pitfall of this paradigm is that surrogate losses defined by a fixed tokenization for non-character level systems, risks being invalidated once the attack changes tokenization. Therefore, Ebrahimi et al. (2018) simply focused on character level systems, while Michel et al. (2019) specially noted to exclude scenarios where attack changes tokenization in their paradigm.

Other works turn to more sophisticated generation paradigms, e.g., Vidnerová and Neruda (2016) adopts a genetic algorithm for an evolutionary generation targeting simple machine learning models. Zang et al. (2019) consider adversarial generation as a word substitution-based combinatorial optimization problem tackled by particle swarm algorithm. Our paradigm shares some common ideology with Miao et al. (2019) and Xiao et al. (2018), which iteratively edit inputs constrained by generative adversarial learning.

### 7 Conclusion

We propose a new paradigm to generate adversarial examples for neural machine translation, which is capable of exposing translation pitfalls without handcrafted error features. Experiments show that our method achieves stable degradation with meaning preserving adversarial examples over different victim models.

It is noticeable that our method can generate adversarial examples efficiently from monolingual data. As a result, the mass production of adversarial examples for the victim model’s analysis and further improvement of robustness become convenient. Furthermore, we notice some exceptional cases which we call as “reinforced samples”, which we leave as the future work.

### Acknowledgement

We would like to thank the anonymous reviewers for their insightful comments. Shujian Huang is the corresponding author. This work is supported by National Science Foundation of China (No. 61672277, 61772261), the Jiangsu Provincial Research Foundation for Basic Research (No. BK20170074).

### References


Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. In EMNLP.


A Training Details for Agent

We adopt commonly accepted translation metric BLEU as score in Eq.9. We use 50 sequence pairs per batch both in environment initialization and training of discriminator and agent. It is essential to train on batches of sequences to stabilize reinforced training. Furthermore, note that $D$ can be too powerful during the early training stage compared to the agent’s actor that it can quickly terminate an exploration. Therefore, we must train on batches and determine an overall terminal signal as aforementioned to ensure early exploration. The $step_D$ and $step_A$ are set as 807 and 120. $acc_{bound}$ for discriminator training is set to 0.85. The $a$ and $b$ in Eq.8 are set to 0.5 and 10. The dimension of feedforward layers in the agent’s actor-critic and discriminator are all 256. We initialize the embedding of both agent and discriminator by the victim’s embedding.

For reinforcement learning, we adopt asynchronous learning with an additional global agent with an additional set of parameter $\theta^\Omega$, we set discount factor $\gamma$ to 0.99, $\alpha$ and $\beta$ in Eq.5 to 0.5 and 0.05 respectively. As for the stop criterion, we set $patience_{round}$ to 15 with convergence boundary for $acc_D$ to 0.52. We adopt Adafactor (Shazeer and Stern, 2018) for training, which is a memory-efficient Adam. The learning rate for agent’s optimizer is initialized as 0.001 and scheduled by rsqrt with 100 steps of warmup. The $K$ for the candidate set is 12.

Our agent takes around 30 hours to converge on a single Nvidia 1080ti. Note that higher $acc_{bound}$ and lower convergence boundary for $D$ indicates higher semantic constraints, which will increase training time.

B Search-based Attack

Search-based adversarial generation is currently widely applied in various robustness machine translation system. We generally follow the strategy of Ebrahimi et al. (2018); Michel et al. (2019) which is applicable for both RNN-search and Transformer. More specifically, the $L_{adv}$ in Eq.7 is derived as:

$$\arg\max_{1 \leq i \leq n, elm_i \in \text{vocab}} |elm' - elm_i| \nabla_{elm_i} L_{adv},$$ (12)

$$L_{adv}(X', Y) = \sum_{t=1}^{|y|} \log (1 - P(y_t | X', y_1...y_{t-1}))$$

Algorithm 1: Reinforced training for agent

Result: A learned global agent $\pi_{\theta^\Omega}$

1. Assume global agent as $\pi_{\theta^\Omega}$ with parameter $\theta^\Omega$
2. Assume agent as $\pi_{\theta}$ with parameter set $\theta$
3. initialize: $Env$ with $D$, $\theta^\Omega$, $\theta$;
4. while not Stop Criterion do
   5. for $step_D$ do
      6. train $D$ with current agent $\pi_{\theta}$;
      7. if $acc_D > acc_{bound}$ break;
   8. end
   9. test current $D$’s accuracy $acc_D$ for stop criterion;
   10. for $step_A$ do
      11. initialize $Env$ state $s_0$;
      12. synchronize $\pi_{\theta}$ with $\pi_{\theta^\Omega}$;
      13. $t = t_{\text{start}}$;
      14. while $s_t$ survive and $t-t_{\text{start}} < t_{\text{max}}$ do
         15. get $out^\text{actor}_t$, $V_t = \pi_{\theta}(s_t)$;
         16. compute entropy $H(out^\text{actor}_t)$;
         17. sample $a_t$ based on $out^\text{actor}_t$;
         18. perform $a_t$ and receive $r_t$ and $s_{t+1}$;
         19. $t \leftarrow t + 1$;
      20. end
      21. $R = \begin{cases} 
      0 & \text{for terminal } s_t \\
      V(s_t) & \text{for non-terminal } s_t
      \end{cases}$
      22. for $i \in \{t - 1, ..., t_{\text{start}}\}$ do
         23. $R \leftarrow \gamma R + r_i$;
         24. accumulate $L^*_i(\theta)$;
         25. accumulate $L^*_t(\theta)$;
      26. end
      27. compute overall loss $L(\theta)$;
      28. perform asynchronous updates on $\theta^\Omega$ with gradient $\frac{\partial L(\theta)}{\partial \theta^\Omega}$;
   29. end
30. end

Three times the average convergence episodes to train a discriminator with initial agent by the given batch size.
where each $P(y_t|X)$ is calculated by Eq. 1 given a corresponding target. For every source sequence, a small ratio of positions is sampled for search. Then we greedy search\(^8\) by the corresponding loss upon those positions with given candidates. For better comparison, we adopt the candidate set used in our model instead of na"ive KNN candidates. Both baseline and our model share the same UNK generation for presentation. We use homophone replacement for Chinese, and strategy by Michel et al. (2019) for English.

\(^8\)Ebrahimi et al. (2018) suggest that greedy search is a good enough approximation.