A Reinforced Generation of Adversarial Samples for Neural Machine Translation

Anonymous ACL submission

Abstract

Neural machine translation systems tend to fail on less decent inputs despite its great efficacy, which may greatly harm the credibility of these systems. Fathoming how and when neural-based systems fail on 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 samples 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 samples. We also present a qualitative and quantitative analysis for the preference pattern of the attack, showing its capability of pitfall exposure.

1 Introduction

Neural machine translation (NMT) based on the encoder-decoder framework, such as RNN-Search (**??**, RNNSearch) or Transformer (**?**, 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. For example, typos may severely deteriorate system outputs (Table 1). Moreover, recent studies show that a neural machine translation system can also be broke by synthetic noisy inputs (**??**). Due to the black-box nature of a neural system, it has been a challenge to fathom when and how it tends to fail.

Instead of analyzing a system based on reported bad cases, researchers seek to apprehend such errors in advance. A straightforward strategy (**?**) is to induce a set of handcrafted error features which are likely to cause system failures. Such strategy is very expensive because it requires the expert knowledge for both linguistics and the targeted neural architecture. Handcrafted features are also less applicable because some common errors in deep learning systems can be hard to formulate, while some others are very specific to certain architectures.

in	ye lu sa leng fa sheng zi sha ***bao zha*** shi jian
out	suicide bombing in jerusalem
in	ye lu sa leng fa sheng zi sha ***bao*** shi jian
out	eastern jerusalem explores a case of eastern europe

Table 1: Fragility of neural machine translation. A typo leaving out a Chinese character "zha" leads to significant change in English translation. Both "bao" and "bao zha" mean "bombing" in English.

Instead of designing error features, recent researchers adopt ideas from adversarial learning (**?**) to generate adversarial samples to mine NLP system pitfalls (**???**). Adversarial samples are minor perturbed inputs which keep the semantic meaning of the input, yet yield degraded outputs. Despite of the success for continuous input, e.g. images, there are two major issues for generating valid adversarial samples in NLP tasks.

One issue is to generate discrete tokens for natural language, e.g. words or characters. **?** follow the adversarial learning paradigm in computer vision to learn perturbed continuous representation, then sample discrete tokens accordingly. However, there is no guaranteed correspondence between the perturbed representation and valid tokens. Therefore, sampling may generate tokens departing from perturbed representation, which undermines the generation process. **?** turn to a search paradigm by a brute-force search for perturbations directly on the token level. To lead the search, a gradient-based **surrogate** loss must be designed upon every token modification indicating pitfalls. However, this paradigm is inefficient due to the formidable computation for gradients over every modified input. Furthermore, surrogate losses defined upon each token

in	Two man are playing on the street corner.
adv in	Two man are playing ***frisbee in the park***.
out	Zwei Männer spielen an einer Straßenecke.
adv out	Zwei Männer spielen frisbee im park.

Table 2: Example of undesirable perturbation in adversarial samples for machine translation in (**?**), though it yields very different output compare to the origin, it does not indicate malfunction in system.

risks being invalidated by any perturb that changes tokenization, which will affect the search process.

Another issue is to keep the semantics of original inputs. Different from the fact that minor noises on images does not change the semantics, sampling discrete tokens from arbitrary perturbed representation (**?**) may generate tokens with different semantics and lead to ill-perturbed samples (Table 2). Searching for the perturbed input also requires a semantic constraint of the search space, for which handcrafted constraints are employed (**?**). Though constraints can also be introduced by multitask modeling with additional annotations (**?**), this is still not sufficient for tasks requiring strict semantic equivalence, such as machine translation.

In this paper, we adopt a novel paradigm that generates more reasonable tokens and secures semantic constraints as much as possible. Our contributions can be summarized as the following:

- We develop a reinforcement learning (**?**, RL) paradigm, which learns to perform discrete perturbations on token level, aiming for direct overall degradation. That is, the victim translation model is regarded as an interactive environment for an agent with the aim to directly maximize its final degradation on specific translation evaluation.

- We combine a GAN-style (**?**) discriminator in environment for the terminal signal in our architecture to further constrain semantics, which is free of additional annotations. Experiments show that our approach not only achieves semantic constrained adversarial samples but also effective attacks for machine translation.

- Since our method does not need to inspect inner of the victim architecture and free of feature engineering targeting architectures, it is viable among different machine translation models. Furthermore, our method outclasses the state-of-the-art adversarial sample generation in efficiency.

- 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

Neural Machine Translation

The most popular architectures for neural machine translation are RNN-search (**?**) and Transformer (**?**). Generally 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 networks. The encoder encodes the source embedding $X_{emb} = [emb_1, emb_2, ...emb_n]$ into hidden representation $H = [h_1, h_2, ..., h_n]$:

$$H = f_{enc}(X_{emb}; \theta_{enc})$$

where θ_{enc} denotes the encoder parameter set, and f_{enc} denotes the encoder network. Then a decoder with attention network f_{dec} attentively access source hidden representations for an auto-regressive generation of each y_i until the end of sequence symbol (EOS) is generated:

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

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

Actor-Critic for Reinforcement Learning

Reinforcement learning (**?**) is a widely used machine learning technique which follows the paradigm of **explore and exploit**. Unlike supervised learning, that is to collect training signals through stochastic policies (explore) and reinforce reward-oriented policies (exploit). Thus reinforcement learning is apt for unsupervised policy learning in many challenging tasks (e.g. games (**?**)). It is also used for direct optimization for non-derivative learning objectives (**??**) in NLP.

Actor-critic (**?**) is one of the most popular reinforcement learning architectures where the agent consists of 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 $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$ 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^π on step t is:

$$L_t^\pi(\theta_\pi) = \log P(a_t|s_t) A_t(s_t, a_t); a_t \in \mathcal{A} \quad (2)$$

where θ_π denotes actor parameters, $A_t(s_t, a_t)$ denotes general advantage function (**?**) on state s_t for action a_t given by $\sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k V(s_{s+k}) - V(s_t)$, which can be further derived as:

$$\begin{aligned} A_t(s_t, a_t) = \gamma A_{t+1}(s_{t+1}, a_{t+1}) + r_t \\ +\gamma V_{t+1}(s_{t+1}) - V_t(s_t) \end{aligned} \quad (3)$$

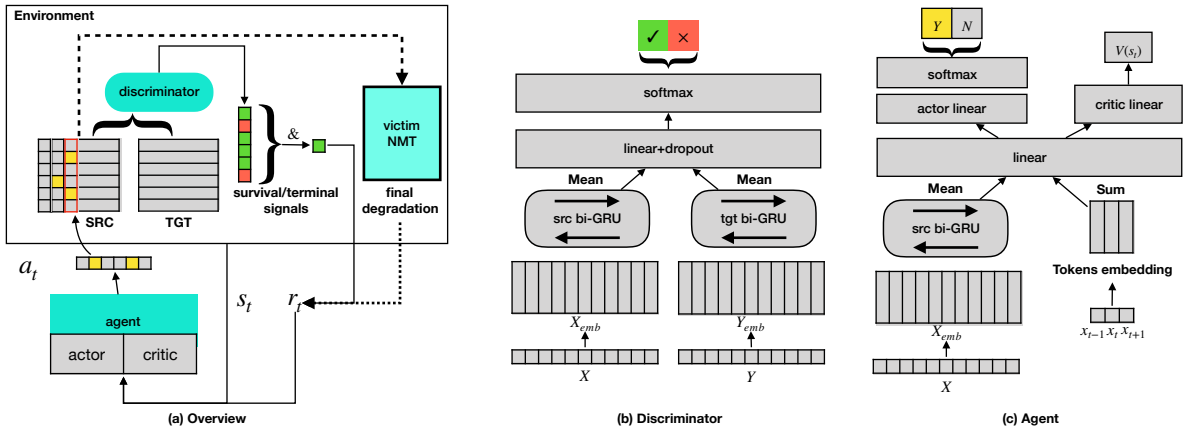

Figure 1: Overall, the victim NMT model is regarded as a part of the environment which yields a rewards indicating degradation based on agent's modification on inputs. Discriminator provides survival signals by determining whether SRC is ill-perturbed. The agent sequentially decides whether to attack on a token from left to right until it reaches the end of sequence or terminated by the discriminator in the environment.

On the other hand, critic learns to estimate expected discounted rewards R_t via minimizing a temporal difference loss L^v on each step t:

$$L_t^v(\theta_v) = \frac{1}{2}(r_t + \gamma R_{t+1} - V_t(s_t))^2 \quad (4)$$

where θ_v denotes critic parameter.

One of the reasons for a vanilla reinforcement learning model to fail is that the early failure of exploration before exploiting optimum policy during training. Maximum entropy actor-critic (**?**) promotes policy entropy to ensure training exploration, that is, to also maximize policy entropy H^π of a stochastic actor during learning. Thus the total loss becomes:

$$L(\theta) = \sum_t (\alpha L_t^v - L_t^\pi - \beta H^\pi(\cdot|s_t)) \quad (5)$$

where α and β are hyper parameters for value loss and entropy coefficients.

Adversarial Samples in NLP

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

$$\underset{\delta}{\text{argmax}} L_{adv}(X+\delta) - \lambda C(\delta) \quad (6)$$

where λ denotes the constraint coefficient. L_{adv} is determined by the goal of the attack. With the perturbed representation, tokens with the nearest embedding are sampled as the results. However, currently popular adversarial generation for NLP is to search by maximizing a surrogate gradient-based loss:

$$\underset{1\le i\le n, x'\in vocab}{\text{argmax}} \quad L_{adv}(x_0, x_1, ...x_i'...x_n) \quad (7)$$

where L_{adv} is a differentiable function indicating adversarial object. Due to its formidable search space, this paradigm simply perturb on a small ratio of token positions and greedy search by brute force among candidates.

3 Approach

In this section, we will describe our reinforced generation of adversarial samples (Figure 1) in details.

Environment

We encapsulate the victim translation model with a reward process as an **environment** Env for an reinforced **agent** to interact.

Environment State

Env is initialized with N sequences $SRC = [src_0, src1, ..., src_N]$ with similar length, which are processed based on victim translation's vocabulary and tokenization. It is essential to train on batches of sequences to stabilize reinforced training and avoid early exploration failure, which will be further explained in the reward process. Each sequence $src_i = [x_1, x_2, ..., x_n]$ is concatenated with BOS, EOS, which indicates the begin and end of the sequence. Finally, the batch is padded to same length with a mask indicating valid inputs. The state of the Env is described as $s_t = (SRC, t)$, where time step $t \in [1, n]$ also indicates the token position to be perturbed by the agent. Environment will consecutively update s_t and yield reward signals until t reaches the end, or intermediately terminated. That is, **all** sequences in SRC is determined as ill-perturbed during reward process. Once the Env is terminated, it finishes the current episode and reset its state with a new batch of sequences as SRC.

Reward Process with Discriminator

Reward process is only used during training an agent. 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 denote reward as:

$$r_t = \begin{cases} -1, & \text{terminated} \\ \frac{1}{N}\sum_N a \cdot r_{\mathrm{s}}, & \text{survive } \& \ t \in [1, n) \\ \frac{1}{N}\sum_N (a \cdot r_{\mathrm{s}} + b \cdot r_{\mathrm{d}}), & \text{survive } \& \ t = n \end{cases} \tag{8}$$

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 (**?**). That is, the agent must survive for its goal by also fooling D which attempts to terminate ill-perturbed modifications. We define ill-perturbed source by determining whether it still matches original target tgt. We do not follow traditional discriminator training in GAN using original source and perturbed source as inputs, because it promotes agent to do nothing for survival which will result in early training failure.

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 0.5 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:

$$r_{\mathrm{s}} = \begin{cases} P(\text{positive}|(src', tgt); \theta_d), & \text{positive} \\ 0, & \text{otherwise} \end{cases} \tag{9}$$

As long as the environment survives, it yields averaged reward among samples from SRC (Eq.8) to mitigate rewards' fluctuation that destabilize training. Note that D can be too powerful during early training stage compared to agent's actor that it can easily terminate an exploration. Therefore, we must train on batches and determine an overall terminal signal as aforementioned to ensure early exploration.

Discriminator Training Similar to GAN training, environment's D must be updated as the agent updates. During its training, the agent's parameter is freezed to provide training samples. We treat original pair (src, tgt) as positive sample, while (src', tgt) as negative. For every D training epoch, we randomly choose half of the batch and perturb its source using current agent as negative samples. After few epochs of updates, we randomly generate a new batch of pairs from parallel data likewise to test accuracy. D is updated at most $step_D$ epochs, or its test accuracy reaches acc_bound.

Environment **only** yields -1 [1] as overall terminal rewards when all sequences in SRC is **intermediately** terminated. For samples classified as negative during survival, their follow-up rewards and actions are masked as 0. If the environment survives until the end, it yields additional averaged r_{d} as final rewards for an episode. We adopt the relative degradation like (**?**), that is:

$$r_{\mathrm{d}} = \frac{score(y) - score(y')}{score(y)} \tag{10}$$

where y and y' denote original and perturbed output, and $score$ is a translation metric. If $score(y)$ is zero, we return zero as r_{d}. To calculate $score$ we **retokenize** perturbed SRC by victim models vocabulary and tokenizer before translation.

Agent

As it is shown in Figure 1 (c), agent's actor and critic shares 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 victim's vocabulary within a distance ϵ. ϵ is the averaged Euclidean distance of K-nearest embedding for all tokens in victim vocabulary. For those without a nearby candidate, we assign UNK as its candidate. Because we note that there shall always be candidates for a token in test scenarios that are beyond victim's vocabulary. Once the Agent choose to replace a token with UNK, we follow (**?**) to generate a valid token to present adversarial samples that is also UNK to victim's vocabulary using homophone.

Agent Training & Adversarial Generation

The agent is trained by algorithm in appendix A. Since 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. Agent and discriminator take turns to update. If test accuracy for discriminator does not reach over a certain value within certain continuous learning rounds of agent and discriminator, we assume the training is converged.

[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.

To generate adversarial samples, the agent will take in source sequences and decide whether to perturb on each position based on actor's distribution from left to right. As 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

Data Set

We conduct all our adversarial sample generation experiments on Zh→En, En→Fr, and En→De translation tasks, which provide relative strong baselines for victim models and mass test samples.

We train our agent using only parallel data that is used for victims' training. For Zh→En translation, our training set consists of about 1.3 million sentence pairs from LDC[2]. For subword level translation, we apply byte pair encoding (**?**, BPE) for both source and target languages with the vocabulary size of 37k. For word level translation, we use NLPIR-ICTCLAS for Chinese tokenization and Moses tokenizer for English tokenization, and adopt 30k as vocabulary size for both source and target language. We adopt NIST test sets [3] for Zh→En translation and then generate adversarial samples for these sources for analysis.

Victim Model

We choose the state-of-the-art RNN-search and Transformer as victim translation models. For RNN-search, we train a subword level model and strictly follow the architecture in **?** with GRU as recurrent units on every layer and 80 as maximum sequence length. As for Transformer, we train both word-level and subword-level model and strictly follow the architecture and the hyper parameter settings in **?** with 128 as maximum sequence length. For above models, we apply the same batch scheme and Adam optimizer following **?** with MT03 as validation set.

Baseline Attack

We choose the search-based adversarial generation which is currently widely applied in various robustness machine translation system as our baseline. We generally follow the strategy of **??** which is applicable for both RNN-search and Transformer. More specifically, the L_{adv} in Eq.7 is derived as:

$$\underset{1\leq i\leq n, emb'_i \in vocab}{\operatorname{argmax}} |emb' - emb_i| \nabla_{emb_i} L_{adv}, \quad (11)$$

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

[2]LDC2002E18, LDC2003E14, LDC2004T08, LDC2005T06

[3]MT02,03,04,05,06,08

	Avg MT02 – 08			
	BLEU	HE	chrF1	RD
Transformer-word	39.75			
Search (0.2)	32.42	3.22	0.82	0.184
Search (0.3)	28.83	2.45	0.77	0.275
Ours	**33.28**	**3.69**	**0.80**	**0.163**
Transformer-BPE	43.38			
Search (0.2)	34.27	3.87	0.89	0.210
Search (0.4)	27.27	2.91	0.80	0.371
Ours	**31.35**	**3.66**	**0.80**	**0.277**
RNN-search-BPE	39.38			
Search (0.2)	31.83	3.83	0.89	0.192
Search (0.4)	26.13	2.82	0.79	0.336
Ours	**31.18**	**3.60**	**0.83**	**0.208**

Table 3: Experiment results for Zh→En MT attack. Note that sequence length for word level system is shorter, thus we search by ratio 0.3 which shares similar chrF1 with search on subword level system with ratio 0.4. An ideal adversarial sample generation must achieve degradation with respect to higher semantic similarity with origin inputs (HE).

where each $P(y_t|X)$ is calculated by Eq.1 given a corresponding reference, therefore we choose the first reference given by corresponding test set during generation. For every source sequence, a small ratio of positions are sampled for search. Then we greedy search[4] by the corresponding loss upon those positions with given candidates. For better comparison, we adopt the candidate set used in our model instead of naive KNN candidates. Both baseline and our model share the same UNK generation strategies.

Results

We adopt sacreBLEU [5] to test case-insensitive BLEU. For adversarial sample evaluation, we follow **?** to use character-level F1 score (chrF1) to indicate modification rates of the inputs when reporting relative degradation (RD) of BLEU. Since the search paradigm attacks by a predefined ratio indicating modification rate, while our reinforced agent actively determines the modification, we compare the search results with similar F1 score. We also test source semantic similarity with human evaluation (HE) ranging from 0 to 5 used by **?** by randomly sample 20% of total sequences from each results mixed with corresponding baseline results for double-blind test.

As it is shown in Table 3, our model can stably generate adversarial samples without significant change in semantics **with the same training setting for different models**, while search methods must tune for proper ratio of modification, which can hardly strike a balance between semantic constraints and degradation.

For word level translation (Transformer-word), both search results and our methods share similar chrF1,

[4]**?** suggest that greedy search is a good enough **approximation**.

[5]https://github.com/awslabs/sockeye/tree/master/contrib/sacrebleu

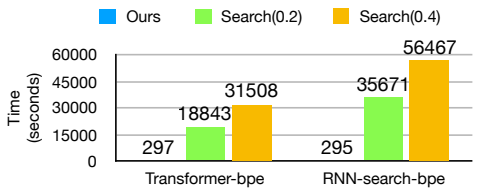

Figure 2: Time consumption of different methods: we limit memory usage to 2.5G on single nvidia 1080, and generate adversarial samples for the same 800 inputs with different methods, our method significantly outclass the state-of-the-art search paradigm.

however they achieve degradation with lower human evaluation scores, while our method achieves degradation more effectively with higher human evaluation. For subword level Transformer (Transformer-BPE), our model achieves significantly more BLEU degradation with very little sacrifice in semantics compared to search (0.2), while search (0.4) shares the similar modification rate with tremendous loss in human evaluation. For subword level RNN-search, our model also achieves similar degradation compared to search (0.2), while search (0.4) with similar modification rate still suffers tremendous loss in human evaluation while achieving the most degradation.

5 Analysis

Efficiency

As it is shown in Figure 2, given same amount of memory cost, our method is significantly more efficient compared to search paradigm. Gradient computation with respect to every modified source sequence can cost greatly in time or space for a state-of-the-art system, which could be even worse for systems with recurrent units. When it comes to mass production of adversarial samples for a victim translation system, our method can also generate by given only monolingual inputs, while search methods must be given same amount of well-informed targets.

Attack Patterns

To further analyze pitfalls, we first adopt LTP POS tagger[6] to label NIST test sets, then check the modification rate for each POS. To ensure the reliability of our analysis, we run three sets of experiments on both baseline and our agent with **similar modification rate** targeting state-of-the-art Transformer with BPE, and collect overall results.

As it is shown in Figure 3 our reinforced paradigm shows specific preference upon certain POS tags, indicating pitfalls of a victim translation system, while search paradigm distributed almost evenly upon different POS tags. Note that unlike existing work relying on feature engineering, we have no error features implemented for agents. However, our agent can still spot error patterns by favoring some of the POS, such as Ni (organization name), Nh (person name), Nl (location name), M (numbers), which are commonly accepted as hard-to-translate parts. Moreover, the agent also tend to favor K (suffix) more, which is less noticed.

[6]https://github.com/HIT-SCIR/ltp

6 Related Work

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

On the other hand, **?** and **?** 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. **?** introduced a learned language model to constrain generation. However, a learned language model is not apt for adversarial samples involving 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, **?** simply focused on char-level systems, while **?** specially noted to exclude scenarios in their search paradigm where attack changes tokenization.

7 Conclusion

We propose a new paradigm to generate adversarial samples 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 samples over different victim models.

Please notice that our method can generate adversarial samples efficiently from monolingual data. As a result, mass production of adversarial samples for victim model's analysis and further improvement of robustness become convenient, which we leave as the future work.

References

Alfred V. Aho and Jeffrey D. Ullman. 1972. *The Theory of Parsing, Translation and Compiling*, volume 1. Prentice-Hall, Englewood Cliffs, NJ.

American Psychological Association. 1983. *Publications Manual*. American Psychological Association, Washington, DC.

Rie Kubota Ando and Tong Zhang. 2005. A framework for learning predictive structures from multiple tasks

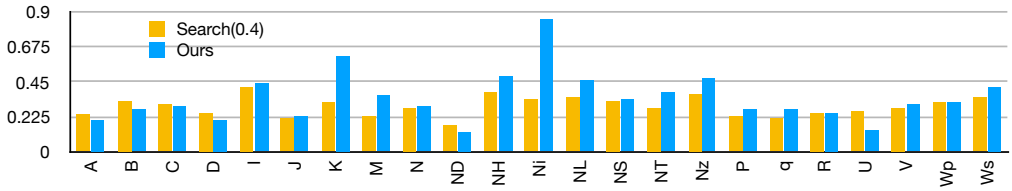

Figure 3: Attack preferences for search (0.4) and our agent on Transformer-BPE model upon different POS. Two attacks share similar modification rate (chrF1). Our agent show a significant preference for some POS (e.g. Ni, Nh, Nz, I), which are commonly regarded as hard-to-translate phrases among industrial implementations, while some (e.g. K) are less noticed.

and unlabeled data. *Journal of Machine Learning Research*, 6:1817–1853.

Galen Andrew and Jianfeng Gao. 2007. Scalable training of L1-regularized log-linear models. In *Proceedings of the 24th International Conference on Machine Learning*, pages 33–40.

Benjamin Borschinger and Mark Johnson. 2011. A particle filter algorithm for Bayesian wordsegmentation. In *Proceedings of the Australasian Language Technology Association Workshop 2011*, pages 10–18, Canberra, Australia.

Ashok K. Chandra, Dexter C. Kozen, and Larry J. Stockmeyer. 1981. Alternation. *Journal of the Association for Computing Machinery*, 28(1):114–133.

Association for Computing Machinery. 1983. *Computing Reviews*, 24(11):503–512.

James Goodman, Andreas Vlachos, and Jason Naradowsky. 2016. Noise reduction and targeted exploration in imitation learning for abstract meaning representation parsing. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1–11. Association for Computational Linguistics.

Dan Gusfield. 1997. *Algorithms on Strings, Trees and Sequences*. Cambridge University Press, Cambridge, UK.

Mary Harper. 2014. Learning from 26 languages: Program management and science in the babel program. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, page 1. Dublin City University and Association for Computational Linguistics.

Mohammad Sadegh Rasooli and Joel R. Tetreault. 2015. Yara parser: A fast and accurate dependency parser. *Computing Research Repository*, arXiv:1503.06733. Version 2.