Pun-GAN: Generative Adversarial Network for Pun Generation

Fuli Luo[1]*, Shunyao Li[1]*, Pengcheng Yang[1], Lei li[1], Baobao Chang[1,2], Zhifang Sui[1,2], Xu Sun[1]
[1]Key Lab of Computational Linguistics, Peking University
[2]Peng Cheng Laboratory, China
{luofuli, lishunyao, yang_pc, lilei_nlp, chbb, szf, xusun}@pku.edu.cn

Abstract

In this paper, we focus on the task of generating a pun sentence given a pair of word senses. A major challenge for pun generation is the lack of large-scale pun corpus to guide the supervised learning. To remedy this, we propose an adversarial generative network for pun generation (Pun-GAN), which does not require any pun corpus. It consists of a generator to produce pun sentences, and a discriminator to distinguish between the generated pun sentences and the real sentences with specific word senses. The output of the discriminator is then used as a reward to train the generator via reinforcement learning, encouraging it to produce pun sentences which can support two word senses simultaneously. Experiments show that the proposed Pun-GAN can generate sentences that are more ambiguous and diverse in both automatic and human evaluation.[1]

1 Introduction

Generating creative and interesting text is a key step towards building an intelligent natural language generation system. A pun is a clever and amusing use of a word with two meanings (word senses), or of words with the same sound but different meanings (Miller and Gurevych, 2015). In this paper, we focus on the former type of pun, i.e., homographic pun. For example, "I used to be a banker but I lost *interest*" is a pun sentence because the pun word "*interest*" can be interpreted as either *curiosity* or *profits*.

An intractable problem for pun generation is the lack of a large-scale pun corpus in which each pun sentence is labeled with two word senses. Early researches (Hong and Ong, 2009; Valitutti et al., 2013; Petrovic and Matthews, 2013) are mainly based on templates and rules, thus lacking creativity and flexibility. Yu et al. (2018) is the first endeavor to apply neural network to this task, which adopts a constrained neural language model (Mou et al., 2015) to guarantee that a pre-given word sense to appear in the generated sequence. However, Yu et al. (2018) only integrates the generation probabilities of two word senses during the *inference* decoding process, without ***detecting*** whether the generated sentences can support the two senses indeed during *training*. Promisingly, Word Sense Disambiguate (WSD) (Pal and Saha, 2015) which aims at identifying the correct meaning of the word in a sentence via a multi-class classifier, can help the detection of pun sentences to some extent.

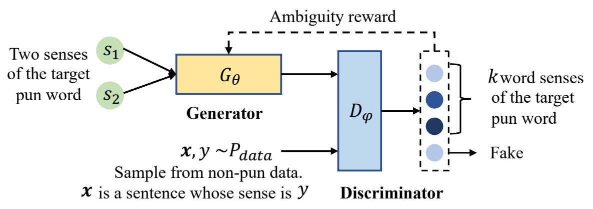
Ambiguity reward
Two senses of the target pun word
s_1
s_2
G_θ
Generator
$x, y \sim P_{data}$
Sample from non-pun data.
x is a sentence whose sense is y
D_φ
Discriminator
k word senses of the target pun word
Fake

Figure 1: The proposed Pun-GAN framework.

Based on the above motivations, we introduce Generative Adversarial Net (Goodfellow et al., 2014) into pun generation task. Specifically, the generator can be any model that is able to generate a pun sentence containing a given word with two specific senses. The discriminator is a word sense classifier to classify the real sentence to its correct word sense label and classify a generated pun sentence to a fake label. With such a framework, the discriminator can provide a well-designed ambiguity reward to the generator, thus encouraging the ambiguity of the generated sentence via reinforcement learning (RL) to achieve the goal of punning, without using any pun corpus.

*Equal Contribution.

[1]The code is available at: https://github.com/lishunyao97/Pun-GAN.

Evaluation of the pun generation is also challenging. We conduct both automatic and human evaluations. The results show that the proposed Pun-GAN can generate a higher quality of pun sentence, especially in ambiguity and diversity.

2 Model

The sketch of the proposed Pun-GAN is depicted in Figure 1. It consists of a pun generator G_θ and a word sense discriminator D_ϕ. The following sections will elaborate on the architecture of Pun-GAN and its training algorithm.

2.1 Model Structure

2.1.1 Generator

Given two senses (s_1, s_2) of a target word w, the generator G_θ aims to output a sentence $\boldsymbol{x}$ which not only contains the target word w but also express the two corresponding meanings. Considering the simplicity of the model and the ease of training, we adopt the neural constrained language model of Yu et al. (2018) as the generator. Due to space constraints, we strongly recommend that readers refer to the original paper for details. Compared with traditional neural language model, the main difference is that the generated words at each timestep should have the maximum sum of two probabilities which are calculated with s_1 and s_2 as input, *respectively*. Formally, the generation probability over the entire vocabulary at t-th timestep is calculated as

$$G_\theta(x_t|\boldsymbol{x}_{<t}) = f(Wh_t^1 + b) + f(Wh_t^2 + b) \quad (1)$$

where h_t^1 (h_t^2) is the hidden state of t-th step when taking s_1 (s_2) as input, f is the softmax function, and $\boldsymbol{x}_{<t}$ is the preceding $t-1$ words.

Therefore, the generation probability of the whole sentence $\boldsymbol{x}$ is formulated as

$$G_\theta(\boldsymbol{x}|s_1, s_2) = \prod_t G_\theta(x_t|\boldsymbol{x}_{<t}) \quad (2)$$

To give a warm start to the generator, we pretrain it using the same general training corpus in the original paper.

2.1.2 Discriminator

The discriminator is extended from the word sense disambiguation models (Kågebäck and Salomonsson, 2016; Luo et al., 2018a,b). Assuming the pun word w in sentence $\boldsymbol{x}$ has k word senses, we add a new "generated" class. Then, the discriminator is designed to produce a probability distribution over $k+1$ classes, which is computed as

$$D_\phi(y|\boldsymbol{x}) = \mathrm{softmax}(U_w c + b') \quad (3)$$

where c is the context vector from a bi-directional LSTM when taking $\boldsymbol{x}$ as input, U_w is a word-specific parameter and y is the target label.

Therefore, $D_\phi\big(y = i|x, i \in \{1, ..., k\}\big)$ denotes the probability that it belongs to the real i-th word sense, while $D_\phi(y = k+1|x)$ denotes the probability that it is produced by a pun generator.

2.2 Training

We follow the training techniques of Salimans et al. (2016) which applys GAN to semi-supervised learning. For real sentence $\boldsymbol{x}$, if it is sense labeled, D_ϕ should classify $\boldsymbol{x}$ to its correct word sense label y, otherwise D_ϕ should classify $\boldsymbol{x}$ to anyone of the k labels. For generated sentence $\boldsymbol{x}$, D_ϕ should classify $\boldsymbol{x}$ to the $(k+1)$-th generated label. Thus, the training objective of the discriminator is to minimize:

$$\begin{aligned}\mathcal{J}(\phi) = &- \mathbb{E}_{\boldsymbol{x},y\sim p_{\text{data}}(\boldsymbol{x},y)} \log p_\phi(y|\boldsymbol{x}) \\ &- \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})} \log p_\phi(y < k+1|\boldsymbol{x}) \\ &- \mathbb{E}_{\boldsymbol{x}\sim G_\theta} \log p_\phi(y = k+1|\boldsymbol{x})\end{aligned} \quad (4)$$

where p_{data} denotes the sentence which only supports one word sense.

To encourage the generator to produce pun text, the discriminator is required to assign a higher reward to the ambiguous pun text which can be interpreted as two meanings simultaneously. For pun sentence, the probability of the target two sense $D_\phi(s_1|\boldsymbol{x})$ and $D_\phi(s_2|\boldsymbol{x})$ should not only *have a small gap*, but also *account for the most*. For example, (0.1, 0.5, 0.4) and (0.1, 0.8, 0.1) are two probability distributions outputted from D_ϕ. The former is more likely to be a pun with the second (0.5) and third (0.4) meaning, while the latter is mostly a generic single sense sentence with the second meaning (0.8). Based on the above observations, the reward is designed as

$$r = \frac{D_\phi(s_1|\boldsymbol{x}) + D_\phi(s_2|\boldsymbol{x})}{|D_\phi(s_1|\boldsymbol{x}) - D_\phi(s_2|\boldsymbol{x})| + 1} \quad (5)$$

where 1 is a coefficient that avoids the denominator being zero.

Then, the goal of generator training is to minimize the negative expected reward.

$$\mathcal{L}(\theta) = -\sum_k r^{(k)} G_\theta(\boldsymbol{x}^{(k)}|s_1, s_2) \quad (6)$$

where $\boldsymbol{x}^{(k)}$ is the k-th sampled sequence, $r^{(k)}$ is the reward of $\boldsymbol{x}^{(k)}$.

By means of policy gradient method (Williams, 1992), for each pair of senses (s_1, s_2), the expected gradient of Eq. 6 can be approximated as:

$$\nabla_\theta \mathcal{L}(\theta) \simeq -\frac{1}{K}\sum_{k=1}^{K} r^{(k)} \nabla_\theta \log\big(G_\theta(\boldsymbol{x}^{(k)})\big) \quad (7)$$

where K is the sample size.

Similar to other GANs (Salimans et al., 2016; Yu et al., 2017), the generator and discriminator are trained alternatively.

3 Experiment

3.1 Dataset

Training Dataset: To keep in line with previous work (Yu et al., 2018), we use a generic corpus – English Wikipedia to train Pun-GAN. For generator, we first tag each word in the English Wikipedia corpus with *one* word sense using an unsupervised WSD tool[2]. Then we use the 2,595K tagged corpus to pre-train our generator. For discriminator, we use several types of data for training: 1) SemCor (Luo et al., 2018a,b) which is a manually annotated corpus for WSD, consisting of 226K sense annotations[3] (first part in Eq.4); 2) Wikipedia corpus as unlabeled corpus (second part in Eq.4); 3) Generated puns (third part in Eq.4).

Evaluation Dataset: We use the pun dataset from SemEval 2017 task7 (Miller et al., 2017) for evaluation. The dataset consists of 1274 human-written puns where target pun words are annotated with two word senses. During testing, we extract the word sense pair as the input of our model.

3.2 Experimental Setting

The generator is the same as Yu et al. (2018). The discriminator is a single-layer bi-directional LSTM with hidden size 128. We randomly initialize word embeddings with the dimension size of 300. The sample size K is set as 32. Batch size is 32 and learning rate is 0.001. The optimization algorithm is SGD. Before adversarial training, we pre-train the generator for 5 epochs and pre-train the discriminator for 4 epochs. In adversarial training, the generator is trained every 1 step and the discriminator is trained every 5 steps.

[2] https://github.com/alvations/pywsd

[3] The reason why we don't use SemCor to train generator is that this dataset is too small for training a language model.

Model	Unusualness	Dist-1	Dist-2
LM (Mikolov et al., 2010)	-	6.8	15.4
CLM (Mou et al., 2015)	0.45	8.3	17.9
CLM+JD (Yu et al., 2018)	0.05	8.8	19.8
Pun-GAN	**0.50**	**11.3**	**26.2**
Human	1.38	27.9	73.5

Table 1: Automatic evaluation results.

Model	Ambiguity	Fluency	Overall
LM (Mikolov et al., 2010)	1.6	3.1	2.5
CLM (Mou et al., 2015)	2.0	2.1	2.0
CLM+JD (Yu et al., 2018)	3.4	3.6	3.5
Pun-GAN	**3.9**	**3.7**	**3.8**
Human	4.3	4.6	4.5

Table 2: Human evaluation results.

3.3 Baselines

We compare with the following systems:

LM (Mikolov et al., 2010): It is a normal recurrent neural language model which takes the target pun word as input.

CLM (Mou et al., 2015): It is a constrained language model which guarantees that a pre-given word will appear in the generated sequence.

CLM+JD (Yu et al., 2018): It is a state-of-the-art model for pun generation which extends a constrained language model by jointly decoding conditioned on two word senses.

3.4 Evaluation Metrics

Automatic evaluation: We use two metrics to automatically evaluate the creativeness of the generated puns in terms of unusualness and diversity. Following Pauls and Klein (2012) and He et al. (2019)[4], the unusualness is measured by subtracting the log-probability of training sentences from the log-probability of generated pun sentences. Following Yu et al. (2018), the diversity is measured by the ratio of distinct unigrams (Dist-1) and bigrams (Dist-2) in generated sentences.

Human evaluation: Three annotators score the randomly sampled 100 outputs of different systems from 1 to 5 in terms of three criteria. Ambiguity evaluates how likely the sentence is a pun. Fluency measures whether the sentence is fluent. Overall is a comprehensive metric.

Model	**Ours**	**No Pref.**	**Others**
Pun-GAN vs CLM+JD	57	19	24
Pun-GAN vs Human	8	13	79

Table 3: Results from human A/B testing of different pairs of models. Each cell indicates the times that a judge preferred one of the models or no preference of them among 100 sentences.

Model	**Unusualness**	**Dist-1**	**Dist-2**
Full Model	**0.50**	**11.3**	**26.2**
- adversarial leaning	0.46	10.9	25.1

Table 4: Ablation study.

3.5 Results

Table 1 and Table 2 show the results of automatic evaluation and human evaluation, respectively. We find that: 1) Pun-GAN achieves the best ambiguity score. This is in line with our expectations that adversarial training can better achieve the aim of punning; 2) Compared with CLM+JD which is actually the same as our pre-trained generator, Pun-GAN has a large improvement in unusualness. We assume that it is because the discriminator can promote to generate more creative and unexpected sentences to some extent via adversarial training; 3) Pun-GAN can generate more diverse sentence with different tokens and words. This phenomenon accords with previous work of GANs (Wang and Wan, 2018).

In addition, Table 3 shows the A/B tests between two the models. It shows that Pun-GAN can generate more vivid pun sentences compared with the previous best model CLM+JD. However, there still exists a big gap between generated puns and human-written puns. To conclude, both automatic evaluation and human evaluation show the effectiveness of the proposed Pun-GAN, especially in ambiguity and diversity.

3.6 Ablation Study

In order to validate the effectiveness of adversarial learning, we fix the discriminator after pretraining. Table 4 shows the results, from which we can conclude that adversarial leaning can help improve the creativeness of generated puns.

3.7 Case Study

Figure 2 shows the randomly sampled examples of state-of-the-art model (CLM+JD) and Pun-GAN. Human-written puns are also given. It demonstrates that, compared with CLM+JD, Pun-GAN can generate puns which are closer to the funniness and creativeness of human-written puns. However, both CLM+JD and Pun-GAN may sometimes generate short sentences. Since too short sentences lack sufficient context, they always tend to ambiguous. More analysis can be found in Section 3.8.

[4] He et al. (2019) is a contemporaneous work.

Model	Example
touch	
s_1: the event of something coming in contact with the body.	
s_2: a suggestion of some quality.	
CLM+JD	It is a <u>*touch*</u> in the united states.
Pun-GAN	It is a <u>*touch*</u> with the red sox.
Human	The massage which came with the spa treatment was a nice <u>*touch*</u>.
state	
s_1: an organized political community forming part of a country.	
s_2: mode or condition of being.	
CLM+JD	According to the <u>*state*</u>, he was the first time in the united states.
Pun-GAN	In the <u>*state*</u>, the national assembly was established.
Human	Many people need to learn to be happy with the <u>*state*</u> they are in.

Figure 2: Example outputs of different models.

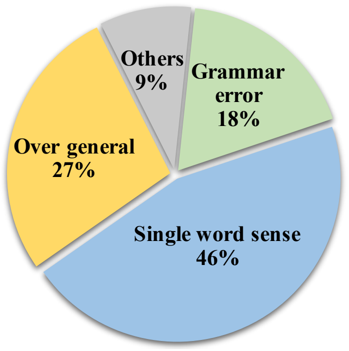

Figure 3: Pie chart of the error types.

3.8 Error Analysis

We carefully analyze the generated results of Pun-GAN with low overall scores in human evaluation. Fig 3 shows the proportion of different error types. The most common type of error is generating a sentence which only supports a single word sense. This accords with our expectations since generating a sentence which can support two word senses without any labeled corpus is very hard. Another common type of error is generating over generic sentences. For example, "*It is a bank*". In most instances, these generic sentences are always very short and they begin with a pronoun like "*It is*" or "*He can*". The reasons are two-fold. One is that these type of sentences can get a high generation probability since the generator is actually a language model. The other is these type of sentences can even get a not bad reward since they are indeed ambiguous. Moreover, grammar error also accounts for about 1/5. We hypothesize that it is caused by the joint generation process in Eq.2.

4 Conclusion and Future Work

In this paper, we propose Pun-GAN: a generative adversarial network for pun generation. It consists of a pun generator and a word sense discriminator, which unifies the task of pun generation and word sense disambiguation. Even though Pun-GAN does not require any pun corpus, it can still enhance the ambiguity of sentence produced by the generator via the reward from the discriminator to achieve the goal of punning. Pun-GAN is generic and flexible, and may be extended to other constrained text generation tasks in future work.

Acknowledgments

This paper is supported by NSFC project 61772040 and 61751201. The contact authors are Baobao Chang and Zhifang Sui.

References

Rahul Dey, Felix Juefei-Xu, Vishnu Naresh Boddeti, and Marios Savvides. 2018. Rankgan: A maximum margin ranking GAN for generating faces. *CoRR*, abs/1812.08196.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014*, pages 2672–2680.

Jiaxian Guo, Sidi Lu, Han Cai, Weinan Zhang, Yong Yu, and Jun Wang. 2018. Long text generation via adversarial training with leaked information. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18).*

He He, Nanyun Peng, and Percy Liang. 2019. Pun generation with surprise. *CoRR*, abs/1904.06828.

Bryan Anthony Hong and Ethel Ong. 2009. Automatically extracting word relationships as templates for pun generation. In *Proceedings of the Workshop on Computational Approaches to Linguistic Creativity*, CALC '09, pages 24–31.

Mikael Kågebäck and Hans Salomonsson. 2016. Word sense disambiguation using a bidirectional lstm. *arXiv preprint arXiv:1606.03568*.

Fuli Luo, Tianyu Liu, Zexue He, Qiaolin Xia, Zhifang Sui, and Baobao Chang. 2018a. Leveraging gloss knowledge in neural word sense disambiguation by hierarchical co-attention. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, 2018*.

Fuli Luo, Tianyu Liu, Qiaolin Xia, Baobao Chang, and Zhifang Sui. 2018b. Incorporating glosses into neural word sense disambiguation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018*.

Tomas Mikolov, Martin Karafiát, Lukás Burget, Jan Cernocký, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *INTERSPEECH 2010, 11th Annual Conference of the International Speech Communication Association, 2010*, pages 1045–1048.

Tristan Miller and Iryna Gurevych. 2015. Automatic disambiguation of english puns. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, Volume 1: Long Papers*, pages 719–729.

Tristan Miller, Christian Hempelmann, and Iryna Gurevych. 2017. Semeval-2017 task 7: Detection and interpretation of english puns. In *Proceedings of the 11th International Workshop on Semantic Evaluation, SemEval@ACL 2017*, pages 58–68.

Lili Mou, Rui Yan, Ge Li, Lu Zhang, and Zhi Jin. 2015. Backbone language modeling for constrained natural language generation. *CoRR*, abs/1512.06612.

Alok Ranjan Pal and Diganta Saha. 2015. Word sense disambiguation: a survey. *CoRR*, abs/1508.01346.

Adam Pauls and Dan Klein. 2012. Large-scale syntactic language modeling with treelets. In *The 50th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference, July 8-14, 2012, Jeju Island, Korea - Volume 1: Long Papers*, pages 959–968. The Association for Computer Linguistics.

Romain Paulus, Caiming Xiong, and Richard Socher. 2017. A deep reinforced model for abstractive summarization. In *Proceedings of the International Conference on Learning Representations, ICLR 2017*.

Sasa Petrovic and David Matthews. 2013. Unsupervised joke generation from big data. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics, ACL 2013, Volume 2: Short Papers*, pages 228–232.

Alessandro Raganato, Claudio Delli Bovi, and Roberto Navigli. 2017. Neural sequence learning models for word sense disambiguation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Tim Salimans, Ian J. Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. 2016. Improved techniques for training gans. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016*, pages 2226–2234.

Alessandro Valitutti, Hannu Toivonen, Antoine Doucet, and Jukka M. Toivanen. 2013. "let everything turn well in your wife": Generation of adult humor using lexical constraints. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics, ACL 2013, Volume 2: Short Papers*, pages 243–248.

Ke Wang and Xiaojun Wan. 2018. Sentigan: Generating sentimental texts via mixture adversarial networks. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018*, pages 4446–4452.

Ronald J. Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Machine Learning*.

Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. 2017. Seqgan: Sequence generative adversarial nets with policy gradient. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, pages 2852–2858.

Zhiwei Yu, Jiwei Tan, and Xiaojun Wan. 2018. A neural approach to pun generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018*.