A Neural Approach to Pun Generation

Zhiwei Yu and **Jiwei Tan** and **Xiaojun Wan**
Institute of Computer Science and Technology, Peking University
The MOE Key Laboratory of Computational Linguistics, Peking University
{yuzw,tanjiwei,wanxiaojun}@pku.edu.cn

Abstract

Automatic pun generation is an interesting and challenging text generation task. Previous efforts rely on templates or laboriously manually annotated pun datasets, which heavily constrains the quality and diversity of generated puns. Since sequence-to-sequence models provide an effective technique for text generation, it is promising to investigate these models on the pun generation task. In this paper, we propose neural network models for homographic pun generation, and they can generate puns without requiring any pun data for training. We first train a conditional neural language model from a general text corpus, and then generate puns from the language model with an elaborately designed decoding algorithm. Automatic and human evaluations show that our models are able to generate homographic puns of good readability and quality.

1 Introduction

Punning is an ingenious way to make conversation enjoyable and plays important role in entertainment, advertising and literature. A pun is a means of expression, the essence of which is in the given context the word or phrase can be understood in two meanings simultaneously (Mikhalkova and Karyakin, 2017). Puns can be classified according to various standards, and the most essential distinction for our research is between homographic and homophonic puns. A homographic pun exploits distinct meanings of the same written word while a homophonic pun exploits distinct meanings of the same spoken word. Puns can be homographic, homophonic, both, or neither (Miller and Gurevych, 2015).

Puns have the potential to combine novelty and familiarity appropriately, which can induce pleasing effect to advertisement (Valitutti et al., 2008). Using puns also contributes to elegancy in literary writing, as laborious manual counts revealed that puns are one of the most commonly used rhetoric of Shakespeare, with the frequency in certain of his plays ranging from 17 to 85 instances per thousand lines (Miller and Gurevych, 2015). It is not an overstatement to say that pun generation has significance in human society. However, as a special branch of humor, generating puns is not easy for humans, let alone automatically generating puns with artificial intelligence techniques. While text generation is a topic of interest in the natural language processing community, pun generation has received little attention.

Recent sequence-to-sequence (seq2seq) framework is proved effective on text generation tasks including machine translation (Sutskever et al., 2014), image captioning (Vinyals et al., 2015), and text summarization (Tan et al., 2017). The end-to-end framework has the potential to train a language model which can generate fluent and creative sentences from a large corpus. Great progress has achieved on the tasks with sufficient training data like machine translation, achieving state-of-the-art performance. Unfortunately, due to the limited puns which are deemed insufficient for training a language model, there has not been any research concentrated on generating puns based on the seq2seq framework as far as we know.

The inherent property of humor makes the pun generation task more challenging. Despite decades devoted to theories and algorithms for humor, computerized humor still lacks of creativity, sophistication of language, world knowledge,

Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Long Papers), pages 1650–1660
Melbourne, Australia, July 15 - 20, 2018.

empathy and cognitive mechanisms compared to humans, which are extremely difficult to model (Hossain et al., 2017).

In this paper, we study the challenging task of generating puns with seq2seq models without using a pun corpus for training. We propose a brand-new method to generate homographic puns using normal text corpus which can result in good quality of language model and avoid considerable expense of human annotators on the limited pun resources. Our proposed method can generate puns according to the given two senses of a target word. We achieve this by first proposing an improved language model that is able to generate a sentence containing a given word with a specific sense. Based on the improved language model, we are able to generate a pun sentence that is suitable for two specified senses of a homographic word, using a novel joint beam search algorithm we propose. Moreover, based on the observed characteristics of human generated puns, we further enhance the model to generate puns highlighting intended word senses. The proposed method demonstrates the ability to generate homographic puns containing the assigned two senses of a target word.

Our approach only requires a general text corpus, and we use the Wikipedia corpus in our experiment. We introduce both manual ways and automatic metrics to evaluate the generated puns. Experimental results demonstrate that our methods are powerful and inspiring in generating homographic puns.

The contributions of our work are as follows:

- To our knowledge, our work is the first attempt to adopt neural language models on pun generation. And we do not use any templates or pun data sets in training the model.
- We propose a brand-new algorithm to generate sentences containing assigned distinct senses of a target word.
- We further ameliorate our model with associative words and multinomial sampling to produce better pun sentences.
- Our approach yields substantial results on generating homographic puns with high accuracy of assigned senses and low perplexity.

2 Related Work

2.1 Pun Generation

In recent decades, exploratory research into computational humor has developed to some extent, but seldom is research specifically concerned with puns. Miller and Gurevych (2015) found that most previous studies on puns tend to focus on phonological or syntactic pattern rather than semantic pattern. In this subsection we briefly review some prior work on pun generation.

Lessard and Levison (1992) devised a program to create Tom Swifty, a type of pun which is present in a quoted utterance followed by a punning adverb. Binsted and Ritchie (1994) came up with an early prototype of pun-generator Joke Analysis and Production Engine (JAPE). The model generates question-answer punning with two types of structures: schemata for determining relationships between key words in a joke, and templates for producing the surface form of the joke. Later its successor JAPE-2 (Binsted, 1996; Binsted et al., 1997) and STANDUP (Ritchie et al., 2007) introduced constructing descriptions. The Homonym Common Phrase Pun generator (Venour, 1999) could create two-utterance texts: a one-sentence set-up and a punch-line. Venour (1999) used schemata to specify the required lexical items and their intern relations, and used templates to indicate where to fit the lexical items in a skeleton text (Ritchie, 2004). McKay (2002) proposed WISCRAIC program which can produce puns in three forms: question-answer form, single sentence and a two-sentence sequence. The Template-Based Pun Extractor and Generator (Hong and Ong, 2009) utilized phonetic and semantic linguistic resources to extract word relationships in puns automatically. The system stores the extracted knowledge in template form and results in computer-generated puns.

Most previous research on pun generation is based on templates which is convenient but lacks linguistic subtlety and can be inflexible. None of the systems aimed to be creative as the skeletons of the sentences are fixed and the generation process based on lexical information rarely needs world knowledge or reasoning (Ritchie, 2004). Recently more and more work focuses on pun detection and interpretation (Miller et al., 2017; Miller and Gurevych, 2015; Doogan et al., 2017), rather than pun generation.

2.2 Natural Language Generation

Natural language generation is an important area of NLP and it is an essential foundation for the tasks like machine translation, dialogue response generation, summarization and of course pun generation.

In the past, text generation is usually based on the techniques like templates or rules, probabilistic models like n-gram or log-linear models. Those models are fairly interpretable and well-behaved but require infeasible amounts of hand-engineering to scale with the increasing training data (Xie, 2017). In most cases larger corpus reveals better what matters, so it is natural to tackle large scale modeling (Józefowicz et al., 2016).

Recently, neural network language models (Bengio et al., 2003) have shown the good ability to model language and fight the curse of dimensionality. Cho et al. (2014) propose the encoder-decoder structure which proves very efficient to generate text. The encoder produces a fixed-length vector representation of the input sequence and the decoder uses the representation to generate another sequence of symbols. Such model has a simple structure and maps the source to the target directly, which outperforms the prior models in text generation tasks.

3 Our Models

The goal of our pun generation model is to generate a sentence containing a given target word as homographic pun. Give two senses of the target word (a polyseme) as input, our model generates a sentence where both senses of the word are appropriate in the sentence. We adopt the encoder-decoder framework to train a conditional language model which can generate sentences containing each given sense of the target word. Then we propose a joint beam search algorithm to generate an appropriate sentence to convey both senses of the target word. We call this **Joint Model** whose basic structure is illustrated in Figure 1. We further propose an improved model to highlight the different senses of the target word in one sentence, by reminding people the specific senses of the target word, which may not easily come to mind. We achieve this by using Pointwise Mutual Information (PMI) to find the associative words of each sense of the target word and increase their probability of appearance while decoding. To improve the diversity of the generated sentence, we use multinomial sampling to decode words in the decoding process. The improved model is named the **Highlight Model**.

3.1 Joint Model

3.1.1 Conditional Language Model

For a given word as input, we would like to generate a natural sentence containing the target word with the specified sense. We improve the neural language model to achieve this goal, and name it conditional language model.

The conditional language model for pun generation is similar to the seq2seq model with an input of only one word. We use Long Short-Term Memory (LSTM) as encoder to map the input sequence (target word) to a vector of a fixed dimensionality, and then another LSTM network as decoder to decode the target sequence from the vector (Sutskever et al., 2014).

Our goal is to generate a sentence containing the target word. However, vanilla seq2seq model cannot guarantee the target word to appear in the generated sequence all the time. To solve this problem, we adopt the asynchronous forward/backward generation model proposed by Mou et al. (2015), which employs a mechanism to guarantee some word to appear in the output in seq2seq models. The model first generates the backward sequence starting from the target word w_t at position t of the sentence (i.e., the words before w_t), and ending up with "</s>" at the position 0 of the sentence. The probability of the backward sequence is denoted as $p(\boldsymbol{w}_t^1)$. Then we reverse the output of the backward sequence as the input to the forward model. In this process, the goal of the encoder is to map the generated half sentence to a vector representation and the decoder will generate the latter part accordingly. The probability of the forward sequence is denoted as $p(\boldsymbol{w}_t^n)$. Then the input and output of the forward model are concatenated to form the generated sentence. In the asynchronous forward/backward model, the probability of the output sentence can be decomposed as:

$$p(\overset{\boldsymbol{w}_t^1}{\underset{\boldsymbol{w}_t^n}{=}}) = p(w_t) \prod_{i=0}^{t} p^{(bw)}(w_{t-i}|\cdot) \prod_{i=0}^{m-t+1} p^{(\mathrm{fw})}(w_{t+i}|\cdot), \tag{1}$$

where p(≒) denotes the probability of a particular backward/forward sequence (Mou et al., 2015). $p^{(\mathrm{bw})}(w_t|\cdot)$ or $p^{(\mathrm{fw})}(w_t|\cdot)$ denotes the probabil-

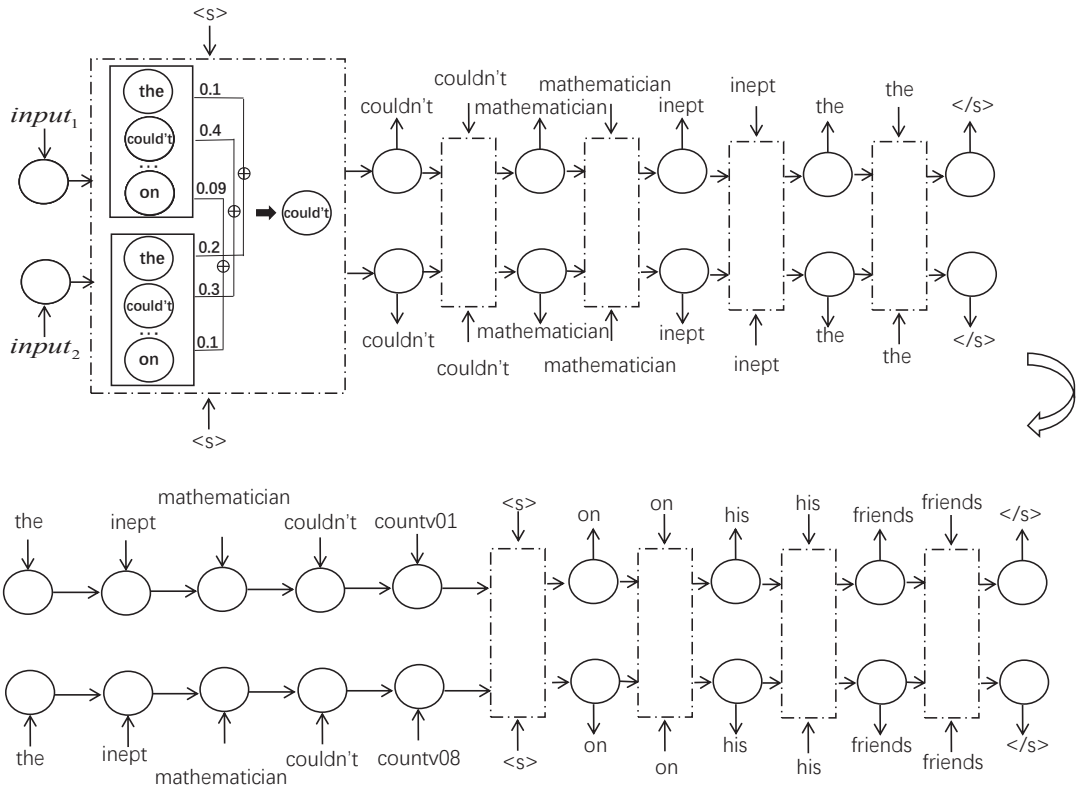

Figure 1: Framework of the proposed Joint Model. (*Top*) Two senses of the target word $input_1$ and $input_2$ (e.g. "*countv01*" and "*countv08*") are firstly provided to the backward model, to generate the backward sequence starting from the target senses and ending up with "</s>". (*Bottom*) Then the backward sequence are reversed and inputted to the forward model, to generate the forward sequence. The inputs and outputs of the forward model are concatenated to form the final output sentence. Joint beam search algorithm is used to generate each word that has the potential to make the generated sentence suitable for both input senses.

ity of w_t given previous sequence $\cdot$ in the backward or forward model respectively. The above model can only guarantee the target word to appear in the generated sentence. Since we hope to generate a sentence containing the specified word sense, we treat different senses of the same word as independent new pseudo-words. We label the senses of words with Word Sense Disambiguation (WSD) tools, and then we train the language model using the corpus with labeled senses so that for each word sense we can generate a sentence accordingly. We use the Python Implementations of WSD Technologies[1] for WSD. This tool can return the most possible sense for the target word based on WordNet (Miller, 1995). We attach the sense label to the word and form a new pseudo-word accordingly. Taking *"count"* for example, *"countv01"* means "*determine the number or amount of*", while "*countv08*" means "*have faith or confidence in*".

3.1.2 Decoding with Joint Beam Search Algorithm

Beam search is a frequently-used algorithm in the decoding stage of seq2seq models to generate the output sequence. It can be viewed as an adaptation of branch-and-bound search that uses an inadmissible pruning rule. In the beam search algorithm, only the most promising nodes at each level of the search graph are selected and the rest nodes are permanently removed. This strategy makes beam search able to find a solution within practical time or memory limits and work well in practical tasks (Zhou and Hansen, 2005; Freitag and Al-Onaizan, 2017).

We also use beam search in our pun generation model. According to the definition of homographic puns, at least two senses of the target word

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

should be interpreted in one sentence. We hope to generate a same sentence for distinct senses of the same word, and in this way the target word in the sentence can be interpreted as various senses. Provided with two senses of a target word as inputs to the encoder in the backward generation process, e.g. "*countv01*" as $input_1$ and "*countv08*" as $input_2$, we decode two output sentences in parallel, and the two sentences should be the same except for the input pseudo-words. Assume $\boldsymbol{h}_{t,i}^{(s)}$ denotes the hidden state of the i-th beam at time step t, when given the s-th pseudo-word as input (s =1 or 2). In the traditional beam search algorithm, softmax layer is applied on the hidden state to get the probability distribution on the vocabulary, and the log likelihood of the probability is used to get a word score distribution $\boldsymbol{d}_{t,i}^{(s)}$:

$$\boldsymbol{d}_{t,i}^{(s)} = log(softmax\_layer(\boldsymbol{h}_{t,i}^{(s)})). \quad (2)$$

The accumulated score distribution on the i-th beam is:

$$\boldsymbol{p}_{t,i}^{(s)} = \boldsymbol{u}_{t-1,i}^{(s)} + \boldsymbol{d}_{t,i}^{(s)}, \quad (3)$$

$|V|$ denotes the vocabulary size. $\boldsymbol{u}_{t-1,i}^{(s)}$ is a $|V|$-dimensional vector whose values are all equal to the accumulated score of the generated sequence till time step $t-1$. Assume the beam width is b, $\boldsymbol{p}_t^{(s)}$ is the concatenation of $\boldsymbol{p}_{t,i}^{(s)}$ on all beams and its dimension size is $|V|*b$. The beam search algorithm selects b candidate words at each time step according to $\boldsymbol{p}_t^{(s)}$(s =1 or 2). When decoding for $input_1$ and $input_2$ in parallel, at each time step there will be b candidates for each input according to $\boldsymbol{p}_t^{(1)}$ and $\boldsymbol{p}_t^{(2)}$ respectively. Since $input_1$ and $input_2$ are different, the candidates for two inputs will hardly be the same. However, our goal is to choose candidate words which have the potential to result in candidate sentences suitable for both senses. Our joint beam search algorithm selects b candidates while decoding for the two inputs according to the joint score distribution on all beams. The joint score distribution on the i-th beam is:

$$\boldsymbol{o}_{t,i} = \boldsymbol{p}_{t,i}^{(1)} + \boldsymbol{p}_{t,i}^{(2)}. \quad (4)$$

The summation of the log scores can be viewed as the product of original probabilities, which represents the joint probability if the two probability distributions are viewed independent. Given the b candidates selected according to the joint score distribution, our joint beam search algorithm is similar to the vanilla beam search algorithm, which generates the candidate sequences step by step. If any beam selects "</s>" as the candidate, we regard this branch has finished decoding. The decoding process will be finished after all the beams have selected "</s>". The joint beam search algorithm is described in Algorithm 1.

Algorithm 1 Joint Beam Search Algorithm

b denotes the beam width. l denotes the number of unfinished beams. $BeamId$ records which beams the candidates come from. $WordId$ records the indices of candidates in the vocabulary where 1 is the index of "<s>". $BEAM_t[i]$ denotes the i-th beam history till time step t. $|V|$ denotes the vocabulary size. $Copy(m, n)$ aims to make an n-dimensional vector by replicating m for n times. The initial states of the decoder ($\boldsymbol{h}_{-1,i}^{(1)}$,$\boldsymbol{h}_{-1,i}^{(2)}$) are equal to the final states of the encoder accordingly. $\boldsymbol{m} \uplus \boldsymbol{n}$ denotes appending $\boldsymbol{n}$ to $\boldsymbol{m}$.

$BEAM_{-1}[i]$= [], i=0, 1, ..., $b-1$
$\boldsymbol{u}_{-1,i}^{(1)} = \boldsymbol{u}_{-1,i}^{(2)} = Copy(0, |V|)$,$i = 0, 1, ..., b-1$
$BeamId$ = $[0, 1, ..., b-1]$
$WordId$ = $[1, .., 1] \in \mathbb{R}^b$
$Outputs$ = []; $t = 0$; $l = b$
while $l > 0$ **do**
  $\boldsymbol{o}$=[]
  for i= 0 to $b-1$ **do**
    $\boldsymbol{x}_{t,i}$ is the word embedding corresponding to $WordId[i]$
    $\boldsymbol{h}_{t,i}^{(1)}$=LSTM($\boldsymbol{x}_{t,i}, \boldsymbol{h}_{t-1,i}^{(1)}$)
    $\boldsymbol{h}_{t,i}^{(2)}$=LSTM($\boldsymbol{x}_{t,i}, \boldsymbol{h}_{t-1,i}^{(2)}$)
    $\boldsymbol{p}_{t,i}^{(1)} = \boldsymbol{u}_{t-1,i}^{(1)} + \log(softmax\_layer(\boldsymbol{h}_{t,i}^{(1)}))$
    $\boldsymbol{p}_{t,i}^{(2)} = \boldsymbol{u}_{t-1,i}^{(2)} + \log(softmax\_layer(\boldsymbol{h}_{t,i}^{(2)}))$
    $\boldsymbol{o}_{t,i} = \boldsymbol{p}_{t,i}^{(1)} + \boldsymbol{p}_{t,i}^{(2)}$
    $\boldsymbol{o} \uplus \boldsymbol{o}_{t,i}$
  end for
  $WordId$ = the indices of words with the top b scores in $\boldsymbol{o}$
  $BeamId$ = the indices of source beams w.r.t. $WordId$
  for i= 0 to $b-1$ **do**
    $BEAM_t[i] = BEAM_{t-1}[BeamId[i]] \uplus WordId[i]$
    $\boldsymbol{u}_{t,i}^{(1)} = \boldsymbol{u}_{t,i}^{(2)} = Copy(\boldsymbol{o}_{t,BeamId[i]}[WordId[i]], |V|)$
    if $WordId[i]$ represents "</s>"
      $l = l - 1$
      $Outputs = Outputs \uplus BEAM_t[i]$
    end if
  end for
  $t = t + 1$
return top b items in $Outputs$

3.2 Highlight Model

3.2.1 Word Association

The joint model we described above is able to generate sentences suitable for both given senses of the target word. But we found this model is prone to generate monotonous sentences, making it difficult to discover that the target word in the sentence can be understood in two ways. For example, in the sentence "*He couldn't count on his friends*", people can easily realize that the common meaning "*have faith or confidence in*" of the

word "*count*", but may ignore other senses of the word. If we add some words and modify the sentence as "*The inept mathematician couldn't count on his friends*", people can also come up with the meaning "*determine the number or amount of*" due to the word "*mathematician*". Comparing the examples above, the two senses are proper in both sentences, but people may interpret "*count*" in the two sentences differently. Based on such observations, we improve the pun generation model by adding some keywords to the sentence which could remind people some special sense of the target word. We call those keywords associative words, and the improved model is named as Highlight Model.

To extract associative words of each sense of the target word, we first build word association norms in our corpus by using pointwise mutual information (PMI). As mutual information compares the probability of observing w_1 and w_2 together (the joint probability) with the probabilities of observing w_1 and w_2 independently (chance) (Church and Hanks, 1990), positive PMI scores indicate that the words occur together more than would be expected under an independence assumption, and negative scores indicate that one word tends to appear solely when the other does not (Islam and Inkpen, 2006). In this case we take top k associative words for each sense with relatively high positive PMI scores, which are calculated as follows:

$$PMI(w_1, w_2) = \log_2 \frac{p(w_1, w_2)}{p(w_1) \cdot p(w_2)}. \quad (5)$$

During decoding we increase the probability of the associative words to be chosen according to their PMI scores. For each sense of the target word, we normalize the PMI scores of the associative words as follows:

$$Asso(w_t^{(s)}, c_p) = \sigma(\frac{PMI(w_t^{(s)}, c_p)}{max_{c_j} PMI(w_t^{(s)}, c_j)}), \quad (6)$$

where $w_t^{(s)}$ represents the s-th sense of the target word w_t, and c_p is the p-th associative word for $w_t^{(s)}$. To smooth the PMI scores we use sigmoid function σ which is differentiable and widely used in the neural network models. The final PMI score for each associative word is denoted as $Asso(w_t^{(s)}, c_p)$. As we choose candidates according to a score distribution on the whole vocabulary, we need a PMI score distribution ($\boldsymbol{S}(w_t^{(s)})$) rather than single scores, and the value at position q is supposed to be:

$$\boldsymbol{S}\left(w_t^{(s)}\right)[q] = \begin{cases} Asso\left(w_t^{(s)}, v_q\right), v_q \in AssoTK(w_t^{(s)}); \\ 0, \qquad\qquad else, \end{cases} \quad (7)$$

where v_q denotes the q-th word in the vocabulary, and $AssoTK(w_t^{(s)})$ denotes the top k associative words of $w_t^{(s)}$.

3.2.2 Multinomial Sampling

In our highlight model, we add $\boldsymbol{S}(w_t^{(1)})$ and $\boldsymbol{S}(w_t^{(2)})$ to $\boldsymbol{o}_{t,i}$, as:

$$\widetilde{\boldsymbol{o}}_{t,i} = \boldsymbol{o}_{t,i} + \alpha_1 \cdot \boldsymbol{S}(w_t^{(1)}) + \alpha_2 \cdot \boldsymbol{S}(w_t^{(2)}), \quad (8)$$

where we use α_1 and α_2 as coefficient weights to balance the PMI scores of the two assigned senses and the joint score. In the Highlight Model, we first select $2b$ candidates according to the scores of words from Eq. 8. Then we use multinomial sampling to select the final b candidates. Sampling is useful in cases where we may want to get a variety of outputs for a particular input. One example of a situation where sampling is meaningful would be in a seq2seq model for a dialog system (Neubig, 2017). In our pun generation model we hope to produce relatively more creative sentences, so we use multinomial sampling to increase the uncertainty when generating the sentence. The multinomial distribution can be seen as a multivariate generalization of the binomial distribution and it is prone to choose the words with relatively high probabilities. If an associative word of one sense has been selected, we decay the scores for all associative words of this sense. In this way we can prevent the sentence obviously being prone to reflect one sense of the target word.

4 Experiments

4.1 Data Preprocessing

Most text generation tasks using seq2seq model require large amount of training data. However, for many tasks, like pun generation, it is difficult to get adequate data to train a seq2seq model. In this study, our pun generation model does not rely on training data of puns. We only require a text corpus to train the conditional language model,

which is very cheap to get. In this paper, we use the English Wikipedia corpus to train the language model. The corpus texts are firstly lowercased and tokenized, and all numeric characters are replaced with "#". We split the texts into sentences and discard the sentences whose length is less than 5 words or more than 50 words. We then select polysemes appearing in the homographic pun data set (Miller et al., 2017) and pun websites. Those polysemes in the corpus are replaced by the labeled sense. We restrict that each sentence can be labeled with at most two polysemes in order to train a reliable language model. If there are more polysemes in one sentence, we keep the last two because in our observation we found pun words tend to occur near the end of a sentence. After labeling, we keep the 105,000 most frequently occurring words and other words are replaced with the "<unk>" token. We discard the sentences with two or more "<unk>" tokens. There are totally 3,974 distinct labeled senses corresponding to a total of 772 distinct polysemes. We assume those reserved senses are more likely to generate puns of good quality.

While training the language model we use 2,595,435 sentences as the training set, and 741,551 sentences as the development set to decide when to stop training.

4.2 Training Details

The number of LSTM layers we use in the seq2seq model is 2 and each layer has 128 units. To avoid overfitting, we set the dropout rate to 0.2. We use Stochastic Gradient Descent (SGD) with a decreasing learning rate schedule as optimizer. The initial learning rate is 1.0 and is halved every 1k steps after training for 8k steps, which is the same as Luong et al. (2017). We set beam size $b = 5$ while decoding. For each sense we select at most 30 associative words (k=30). To increase the probability of choosing the associative words, we set $\alpha_1 = 6.0$ and $\alpha_2 = 6.0$. If an associative word of some sense of a target word has been chosen, its corresponding α will be set to zero for all the associative words of this sense.

4.3 Baselines

Since there is no existing neural model applied on this special task, we implement two baseline models for comparison. We select 100 target words and two senses for each word to test the quality of those models.

Normal Language Model: It is trained with an encoder-decoder model and uses beam search while decoding. In the training process, inputs are unlabeled target words and outputs are sentences containing the target words.

Pun Language Model: We use the data set of homographic puns from Miller et al. (2017). The model is trained on the data set in asynchronous forward/backward way. As the pun data set is limited, the pun language model has no creativity, which means if we input a word appearing in the training data, then the output will usually be an existing sentence from the training data. Therefore, we remove the sentences which contain words in the 100 target words from the pun data set, and then train the model for test.

4.4 Automatic Evaluation

We select 100 target words and two senses for each word for test. We use the language modeling toolkit SRILM[2] to train a trigram model with another 7,746,703 sentences extracted from Wikipedia, which are different from the data set used before. The perplexity scores (PPL) of our models and baseline models are estimated based on the trained language model, as shown in Table 1. Normal Language Model has no constraint of generating sentences suitable for both senses. This means at each time step the beam search algorithm can select the candidates with highest probabilities. And thus it is natural that it obtains the lowest perplexity. Taking the constraint of senses into consideration, the perplexity scores of Joint Model and Highlight Model are still comparable to that of Normal Language Model. However, Pun Language Model could not be trained well considering the limit of the pun training data, so it gets the highest perplexity score. This result reveals that it is not feasible to build a homographic pun generation system based on the pun data set since pun data is far from enough. In the table, We further compare the diversity of the generated sentences of four models following Li et al. (2016). Distinct-1 (d.-1) and distinct-2 (d.-2) are the ratios of the distinct unigrams and bigrams in generated sentences, i.e., the number of distinct unigrams or bigrams divided by the total number of unigrams or bigrams. The results show our models are more creative than Normal Language and

[2]http://www.speech.sri.com/projects/srilm/

Model	PPL	d.-1(%)	d.-2(%)
Highlight	91.80	**27.13**	**62.85**
Joint	63.48	22.13	50.59
Normal Language	**62.66**	19.60	41.62
Pun Language	889.07	14.78	23.11

Table 1: Results of automatic evaluation.

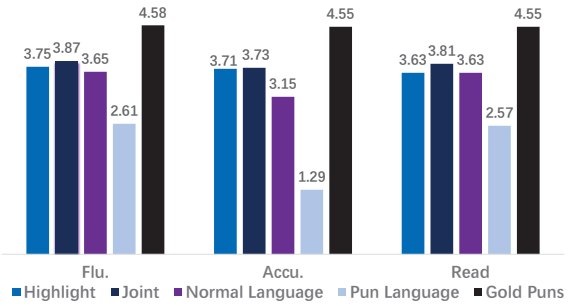

Figure 2: Results of human evaluation.

Pun Language models, and Highlight Model can generate sentences with the best diversity.

4.5 Human Evaluation

Because of the subtle and delicate structure of puns, automatic evaluation is not enough. So we sample one sentence for each word from four models mentioned above and then get 100 sentences of each model generated from the target words, together with 100 puns containing the same target words from homographic pun data set in Miller et al. (2017). We ask judges on Amazon Mechanical Turk to evaluate all the sentences and the rating score ranges from 1 to 5. Five native English speakers are asked to give a score on each sentence in three aspects with the following information: **Readability** indicates whether the sentence is easy to understand semantically; **Accuracy** indicates whether the given senses are suitable in a sentence; **Fluency** indicates whether the sentence is fluent and consistent with the rules of grammar.

The results in Figure 2 show that pun data is not enough to train an ideal language model, while Normal Language Model has enough corpus to train a good language model. But Normal Language Model is unable to make the given two senses appear in one sentence and in a few cases even can not assure the appearance of the target words. Joint Model and Highlight Model can generate fluent sentences for the assigned two senses. Although Highlight Model could remind people

Model	# sentences	avg. score
Highlight	15	0.98
Joint	12	0.87
Gold Puns	28	1.38

Table 2: Results of Soft Turing Test.

specific senses of the target words in most cases, in few cases sampled words make the whole sentence unsatisfactory and get a relatively lower score of accuracy. As to the Readability, the Joint Model performs better than other three models. Both Joint model and Highlight model outperform Normal Language Model and Pun Language Model.

To test the potential of the sentences generated by our models to be homographic puns, we further design a Soft Turing Test. We select 30 sentences generated by Joint Model and 30 sentences generated by Highlight Model independently, together with 30 gold puns from the homographic pun data set. We mix them up, and give the definition of homographic pun and ask 10 people on Amazon Mechanical Turk to judge each sentence. People can judge each sentence as one of three categories: definitely by human, might by human and definitely by machine. The three categories correspond to the scores of 2, 1 and 0, respectively. If the average score of one sentence is equal or greater than 1, we regard it as judged to be generated by human. The number of sentences judged as by human for each model and the average score for each model are shown in Table 2.

Due to the flexible language structure of Highlight Model, the generated homographic puns outperform those generated by Joint Model in the Soft Turing Test, however still far from gold-standard puns. Our models are adept at generating homographic puns containing assigned senses but weak in making homographic puns humorous.

4.6 Examples

We show some examples generated by different models in Table 3. For the two senses of *"pitch"*, Highlight Model generates a sentence which uses *"high"* to remind readers the sense related to sound and uses *"player"* to highlight the sense related to throwing a baseball. Joint Model returns a sentence that can be understood in both way roughly only if we give the two senses in advance, otherwise readers may only think of the

Model	Sample
pitch: 1) the property of sound that arise with variation in the frequency of vibration; 2) the act of throwing a baseball by a pitcher to a batter.	
Highlight	in one that denotes player may have had a high pitch in the world
Joint	the object of the game is based on the pitch of the player
Normal Language	this is a list of high pitch plot
Pun Language	our bikinis are exciting they are simply the tops on the mouth
Gold Puns	if you sing while playing baseball you won't get a good pitch
square: 1) a plane rectangle with four equal sides and four right angles, a four-sided regular polygon; 2) someone who doesn't understand what is going on.	
Highlight	little is known when he goes back to the square of the football club
Joint	there is a square of the family
Normal Language	the population density was # people per square mile
Pun Language	when the pirate captain's ship ran aground he couldn't fathom why
Gold Puns	my advanced geometry class is full of squares
problem: 1) a source of difficulty; 2) a question raised for consideration or solution.	
Highlight	you do not know how to find a way to solve the problem which in the state
Joint	he is said to be able to solve the problem as he was a professor
Normal Language	in # he was appointed a member of the new york stock exchange
Pun Language	those who iron clothes have a lot of pressing veteran
Gold Puns	math teachers have lots of problems

Table 3: Examples of outputs by different models.

sense related to baseball. For Normal Language Model, it is difficult to be interpreted in two senses we assigned. Pun Language Model has no ability to return a sentence containing the assigned word at all. Observing the gold pun, the context describes a more vivid scene which we need to pay attention to. For "*square*", sentences generated by Highlight Model and Joint Model can be interpreted in two senses and Highlight Model results in a sentence with dexterity. Normal Language Model give a sentence where "*square*" means neither of the two given senses. Pun Language Model cannot return a sentence we need with no surprise. For "*problem*", both Highlight Model and Joint Model can generate sentences containing assigned two senses while Normal Language Model and Pun Language Model can not return sentences with the target word. Compare to our generated sentences, we find gold puns are more concise and accurate, which takes us into consideration on the delicate structure of puns and the conclusion is still in exploration.

5 Conclusion and Future Work

In this paper, we proposed two models for pun generation without using training data of puns. Joint Model makes use of conditional language model and the joint beam search algorithm, which can assure the assigned senses of target words suitable in one sentence. Highlight Model takes associative words into consideration, which makes the distinct senses more obvious in one sentence. The produced puns are evaluated using automatic evaluation and human evaluation, and they outperform the sentences generated by our baseline models.

For future work, we hope to improve the results by using the pun data and design a more proper way to select candidates from associative words.

Acknowledgment

This work was supported by National Natural Science Foundation of China (61772036, 61331011) and Key Laboratory of Science, Technology and Standard in Press Industry (Key Laboratory of Intelligent Press Media Technology). We thank the anonymous reviewers for their helpful comments. Xiaojun Wan is the corresponding author.

References

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. 2003. A neural probabilistic language model. *Journal of Machine Learning Research* 3:1137–1155. http://www.jmlr.org/papers/v3/bengio03a.html.

Kim Binsted. 1996. Machine humour: An implemented model of puns .

Kim Binsted, Helen Pain, and Graeme D Ritchie. 1997. Children's evaluation of computer-generated punning riddles. *Pragmatics & Cognition* 5(2):305–354.

Kim Binsted and Graeme Ritchie. 1994. An implemented model of punning riddles. In *Proceedings of the 12th National Conference on Artificial Intelligence, Seattle, WA, USA, July 31 - August 4, 1994, Volume 1.*. pages 633–638. http://www.aaai.org/Library/AAAI/1994/aaai94-096.php.

Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint* arXiv:1406.1078. http://arxiv.org/abs/1406.1078.

Kenneth Ward Church and Patrick Hanks. 1990. Word association norms, mutual information, and lexicography. *Computational Linguistics* 16(1):22–29.

Samuel Doogan, Aniruddha Ghosh, Hanyang Chen, and Tony Veale. 2017. Idiom savant at semeval-2017 task 7: Detection and interpretation of english puns. In *Proceedings of the 11th International Workshop on Semantic Evaluation, SemEval@ACL 2017, Vancouver, Canada, August 3-4, 2017*. pages 103–108. https://doi.org/10.18653/v1/S17-2011.

Markus Freitag and Yaser Al-Onaizan. 2017. Beam search strategies for neural machine translation. In *Proceedings of the First Workshop on Neural Machine Translation, NMT@ACL 2017, Vancouver, Canada, August 4, 2017*. pages 56–60. https://aclanthology.info/papers/W17-3207/w17-3207.

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*. Association for Computational Linguistics, Stroudsburg, PA, USA, CALC '09, pages 24–31. http://dl.acm.org/citation.cfm?id=1642011.1642015.

Nabil Hossain, John Krumm, Lucy Vanderwende, Eric Horvitz, and Henry A. Kautz. 2017. Filling the blanks (hint: plural noun) for mad libs humor. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*. pages 638–647. https://aclanthology.info/papers/D17-1067/d17-1067.

Aminul Islam and Diana Inkpen. 2006. Second order co-occurrence PMI for determining the semantic similarity of words. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation, LREC 2006, Genoa, Italy, May 22-28, 2006.*. pages 1033–1038.

Rafal Józefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. 2016. Exploring the limits of language modeling. *arXiv preprint* arXiv:1602.02410. http://arxiv.org/abs/1602.02410.

Greg Lessard and Michael Levison. 1992. Computational modelling of linguistic humour: Tom swifties. In *In ALLC/ACH Joint Annual Conference, Oxford*. pages 175–178.

Jiwei Li, Michel Galley, Chris Brockett, Jianfeng Gao, and Bill Dolan. 2016. A diversity-promoting objective function for neural conversation models. In *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego California, USA, June 12-17, 2016*. pages 110–119. http://aclweb.org/anthology/N/N16/N16-1014.pdf.

Minh-Thang Luong, Eugene Brevdo, and Rui Zhao. 2017. Neural machine translation (seq2seq) tutorial. *https://github.com/tensorflow/nmt* .

Justin McKay. 2002. Generation of idiom-based witticisms to aid second language learning. In *In Stock et al.*. pages 77–87.

Elena Mikhalkova and Yuri Karyakin. 2017. Punfields at semeval-2017 task 7: Employing roget's thesaurus in automatic pun recognition and interpretation. *arXiv preprint* arXiv:1707.05479. http://arxiv.org/abs/1707.05479.

George A Miller. 1995. Wordnet: a lexical database for english. *Communications of the ACM* 38(11):39–41.

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, July 26-31, 2015, Beijing, China, Volume 1: Long Papers*. pages 719–729. http://aclweb.org/anthology/P/P15/P15-1070.pdf.

Tristan Miller, Christian F. 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-2017)*. pages 59–69.

Lili Mou, Rui Yan, Ge Li, Lu Zhang, and Zhi Jin. 2015. Backbone language modeling for constrained natural language generation. *arXiv preprint* arXiv:1512.06612. http://arxiv.org/abs/1512.06612.

Graham Neubig. 2017. Neural machine translation and sequence-to-sequence models: A tutorial. *arXiv preprint* arXiv:1703.01619. http://arxiv.org/abs/1703.01619.

Graeme Ritchie. 2004. *The linguistic analysis of jokes*. Routledge.

Graeme Ritchie, Ruli Manurung, Helen Pain, Annalu Waller, Rolf Black, and Dave O'Mara. 2007. A practical application of computational humour. In *Proceedings of the 4th International Joint Conference on Computational Creativity*. pages 91–98.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*. pages 3104–3112. http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.

Jiwei Tan, Xiaojun Wan, and Jianguo Xiao. 2017. Abstractive document summarization with a graph-based attentional neural model. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*. Association for Computational Linguistics, pages 1171–1181. https://doi.org/10.18653/v1/P17-1108.

Alessandro Valitutti, Carlo Strapparava, and Oliviero Stock. 2008. Textual affect sensing for computational advertising. In *Creative Intelligent Systems, Papers from the 2008 AAAI Spring Symposium, Technical Report SS-08-03, Stanford, California, USA, March 26-28, 2008*. pages 117–122.

Chris Venour. 1999. The computational generation of a class of puns. In *Master's thesis, Queen's University,Kingston, Ontario*.

Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. 2015. Show and tell: A neural image caption generator. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*. IEEE Computer Society, pages 3156–3164. https://doi.org/10.1109/CVPR.2015.7298935.

Ziang Xie. 2017. Neural text generation: A practical guide. *arXiv preprint arXiv:1711.09534* .

Rong Zhou and Eric A. Hansen. 2005. Beamstack search: Integrating backtracking with beam search. In *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS 2005), June 5-10 2005, Monterey, California, USA*. pages 90–98. http://www.aaai.org/Library/ICAPS/2005/icaps05-010.php.