|  |
| --- |
| IE643: Deep Learning: Theory and Practice Aug-Dec 2020  End-term Project Report: Pun-GAN: Generative Adversarial network for pun generation  Team Name: The Hawks Team member: 19i190008  170110062 |

**Abstract**

This project report contains the details on our project which aims to generate pun sentences via generative adversarial network (GAN). We describe the deep learning tools involved in our project. In particular, we mainly explain about the 2 neural network architectures, the loss functions and activation functions used and the training algorithm used. We are trying a new deep learning architecture and superior results are presented using the proposed architecture.

**1. Introduction**

In this project, we focus on the task of generating a pun sentence given a pair of word senses. Punning is an ingenious way to make conversation enjoyable and plays important role in entertainment, advertising and literature [1]. 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. A major challenge for pun generation is the lack of large-scale pun corpus to guide the supervised learning. To remedy this, we followed the proposed algorithm of adversarial generative network for pun generation (Pun-GAN). 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.

We provide a survey of existing literature in Section 2. Our proposal for the project is described in Section 3. We Explained all details on the data sets used for the project in section 4 and give details on experiments in Section 5. A description of future work is given in Section 7. We conclude with a short summary and pointers to forthcoming work in Section 8.

**2. Literature Survey**

Our project draws inspiration from a closely related work by Fuli Luo [1]. In their paper [1], the author briefly talks about the traditional methods of natural language generation which is mainly based on templates and grammatical rules which lacks in creativity and hence we can’t use these methods for the generation of pun sentences. To tackle this problem the author adopted a constrained neural network model from the previous work of Yu et al. [2]. This adopted model is used to generate creative sentences via GAN, the proposed architecture of GAN is illustrated below in the Figure 1. It mainly consists of a generator adopted from Yu et al. [2] and a discriminator which is adopted from word sense disambiguation model [3]. The training of the network is done using the technique of Reinforcement learning.

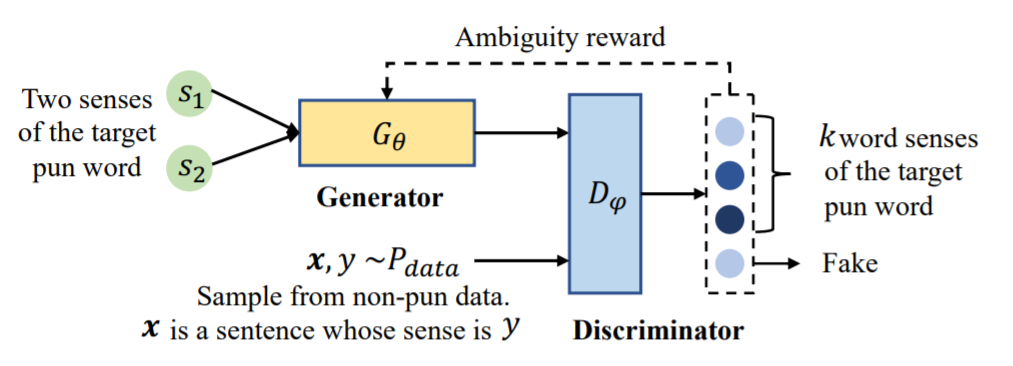


Figure 1 The proposed Pun-GAN framework

The neural network structure consists of two major components one of which is used to automatically generate pun sentences given 2 senses of a target word. The other network discriminator is used to distinguish between a pun sentence and a normal single sense sentence.

Yu et al. [1] was the first attempt towards this automatic pun generation field they used a constrained neural network based on Seq-2-Seq model, it takes one word as input having 2 senses and generates the pun sentence. However, the Seq-2-Seq model itself cannot guarantee the appearance of the target word in the pun sentence and hence we adopted the asynchronous forward/backward generation model proposed by Mou et al. (2015) [4], which employs a mechanism to guarantee some word to appear in the output in seq2seq models. As the name suggests it takes a sequence of items which in our case is a sequence of words and outputs another sequence of items. The architecture of a sequence-to-sequence model is based on encoder- decoder framework [1], where in the sequence of words first got embedded in the form of some vector representation which then fed into the LSTM based encoder and then to another LSTM network which is a decoder to decode the target from a vector to the normal sentence [1]. The model first generates the backward sequence by starting from the target word and then generates all the words before the target sequence. Then we reverse the output of the backward sequence and feed it 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 and then the input and output of the forward model are concatenated to form the generated sentence. After this they proposed a joint beam algorithm to generate an appropriate sentence to convey both the senses of the target word. This integrated model of forward/backward generation model and joint beam algorithm is known as “Joint Model”.

the generation probability at any given time step is given by the following equation where h1t and (h2t) are the hidden state of tth step when taking s1 (s2) as input, f is the SoftMax function, and x<t is the preceding t − 1 words. Therefore, the generation probability of the whole sentence x is formulated as the product of the probability of generation at all timesteps.

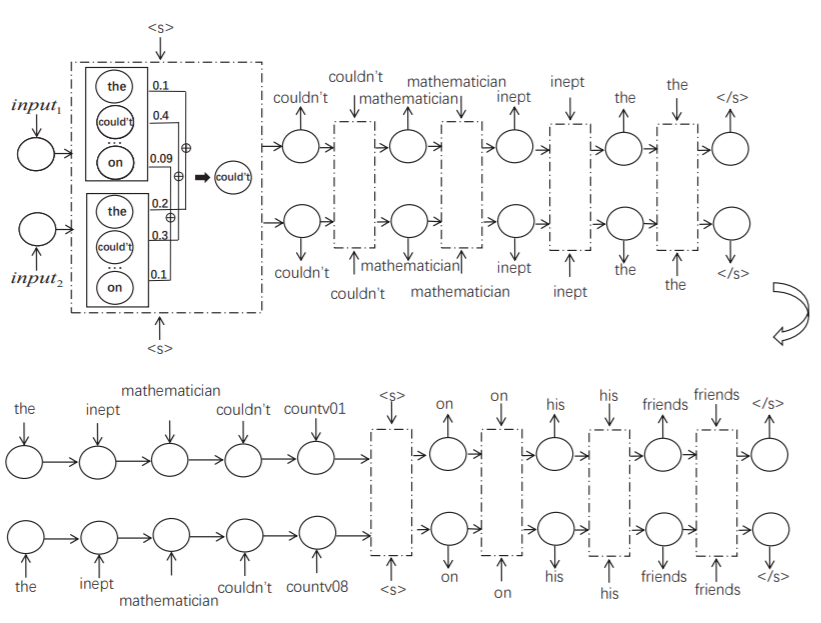


Figure 2 Proposed joint model for the Generator

Now for the discriminator part the paper adopted a word sense disambiguation using a Bidirectional LSTM model [3] which determines the "sense" (meaning) of a word which totally depends on the context. So, to solve this problem we added Bi directional LSTM network in our sequence model, which ensures the state has the information about both the preceding words and succeeding words, which in many cases are absolutely necessary to correctly classify the sense. Now embeddings capture the representation of the word in a higher dimensional plane. Through embeddings, we create a vector representation of the word which is learned by understanding the context of words. So here for embedding we will use the GloVe [3] which is global vector for word representation. Given a document and the position of the target word, i.e. the word to disambiguate, the model computes a probability distribution over the possible senses corresponding to that word. The architecture of the model, depicted in Figure 3, consist of a SoftMax layer, a hidden layer, and a BLSTM. The BLSTM and the hidden layer share parameters over all word types and senses [3], while the SoftMax is parameterized by word type and selects the corresponding weight matrix and bias vector for each type of word respectively.

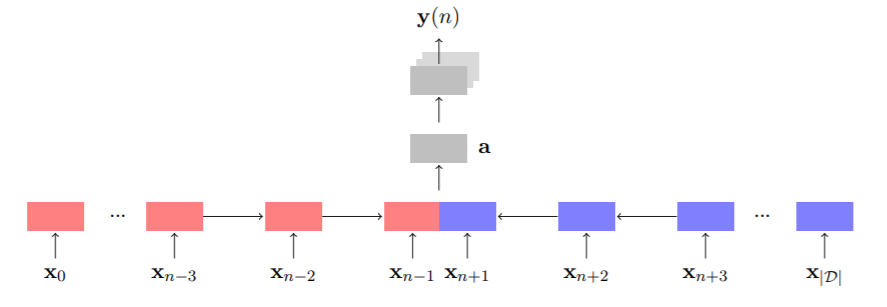


Figure 3 Proposed WSD Bi-LSTM architecture for discriminator.

The above architecture is designed to produce a probability distribution over k+1 class, which is computed as –

where c is the context vector from a bi-directional LSTM when taking x as input, Uw is a word-specific parameter and y is the target label [2]. Therefore, Dφ(y = i|x, i ∈ {1, ..., k}) denotes the probability that it belongs to the real i-th word sense, while Dφ(y = k + 1|x) denotes the probability that it is produced by a pun generator [2].

Now for supervised learning we need large amounts of dataset with tagged 2-word senses. But here we don’t have such a corpus of datasets for puns. And hence here we use the techniques of Reinforcement learning. To train the model we use a policy network which is almost similar to other neural networks. Initially the generator generates random sentences having the target word in it, after that these random sentences are sent to the discriminator to judge whether it’s a pun or a normal sentence and it accordingly gives a reward to the generator. This loop continues and both the networks learn through each other in adversarial manner. The generator’s only motivation is to optimize those rewards as much as possible and finally making it to a global maximum or at-least local maxima. The cost function for discriminator is given as-

The reward given by discriminator to the generator is illustrated as follows-

Where, the probability that given sentence is of sense s1.

The goal of generator is to minimize the negative expected reward, which is given as follows-

**3. Methods and Approaches**

The proposed project is based on the GAN networks and its application in the field of NLP for sentence generation. [1]Yu et al. (2018) is the first endeavour 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 [2]. So to tackle this problem and also due to the lack of tagged pun data sets for training this paper introduced this GAN concept so that it can continuously learn from the rewards given by the discriminator and hence this is a very unique approach to solve the existing problems in this field. Although this paper has adopted the same model of [1]Yu et al. (2018) for the generator part and combined this with a pre-existing neural network for predicting the sense of a ambiguous word used to make a pun sentence.

**WSD Model**

The input to the WSD at position n in document D is computed as

Here, v(wn) is the one-hot representation of the word type corresponding to wn ∈ D [3]. A one-hot representation is a vector with dimension V consisting of |V | − 1 zeros and a single one which index indicate the word type. [3]This will have the effect of picking the column from Wx corresponding to that word type. The resulting vec tor is referred to as a word embedding. Further, Wx can be initialized using pre-trained word embeddings, to leverage large unannotated datasets. In this work GloVe vectors are used for this purpose. The model output will be-

it is basically the predicted distribution over senses for the word at position n, where and are the weights and biases for the softmax layer corresponding to the word type at position n [3]. The loss function is the cross-entropy loss function given as follows-

**3.1 Work done before mid-term project review**

Till mid-term We did an extensive study of previous papers related to the topic and found various problems that give rise to the technique of using GAN networks for generation of pun. We mainly studied 3 papers that are listed here. Now as we discussed earlier, there were 3 main problems which was an obstacle for the generation of pun sentences.

The first one was the lack of creativity and flexibility in sentence generation as traditional methods were only based on some fixed templates, so to solve this problem the paper adopted a Seq2Seq model which is then combined with the constraint that the given word must appear in the sentence. This technique was used in the above listed second number paper which is -” A neural network approach for pun generation (2018)” although it showed a good result but sometimes the input word in generated sentences don’t support 2 senses indeed and hence generating a normal mon pun sentence.

The second problem was the ambiguity of a word which totally depends on the sentence or context, to solve this the paper adopted the technique of sequence modelling using Bi-LSTM network which keeps track the meaning of all the words appearing before and after the given target word hence can find out the sense of the given ambiguous word.

The third problem that arises is for training, now for supervised learning we need large amounts of dataset with tagged 2-word senses. But here we don’t have such a corpus of datasets for puns. And hence here we use the techniques of Reinforcement learning in an adversarial manner so that both the models can learn by each other's mistake.

**3.2 Work done after mid-term project review**

**3.2.1 Discriminator model using WSD**

Word sense disambiguation (WSD), has been a trending area of research in Natural Language Processing and Machine Learning. WSD is basically solution to the ambiguity which arises due to different meaning of words in different context. For example, consider the two sentences-

“The **bank** will not be accepting cash on Saturdays.”

“The river overflowed the **bank.**”

The word**bank**in the first sentence refers to the commercial (finance) banks, while in second sentence, it refers to the river bank. The ambiguity that arises due to this, is tough for a machine to detect and resolve.

**Approach-**

Implemented a simple word sense disambiguation model which takes 2 sentences as an input having a word in common like here in example the word “Bank” which can have 2 meanings, and after that model asks user for queries regarding that word given a query the model will tell the sense of the disambiguate word with respect to user given query. simpleFilter function takes the given query/sentence as an input and returns list of tokens which are lemmatized. Lemmatization refers to deriving the root word which is morphologically correct. There rises a slight confusion between lemmatization and stemming. However, the two words differ in their flavour. Stemming usually refers to a crude heuristic process that chops off the ends of words in the hope of achieving this goal correctly most of the time, and often includes the removal of derivational affixes. Lemmatization usually refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the lemma. The function Stopwords, drops the high frequency words in a language which do not contribute much to the topic of the sentence. In English, such words include, ‘a’, ‘an’, ‘the’, ‘of’, ‘to’, etc. We remove these words and focus on our main subject/topic, to solve ambiguity. The function is applied to the training data sets as well as user input.

Next, we perform similarity check, function: similarityCheck, for the filtered sentence tokens that are returned by the first function. Similarity is checked between the given query/sentence tokens and the training data set tokens. For this, the synonym set is loaded for each token word from wordnet corpus. The depth and closeness of a word is calculated and returned on scale of 0–1. This is the main data that will resolve the ambiguity. The more data you provide, the more accurate it gets. The normalised similarity between sentences is stored. synonymsCreator is a simplistic function to store the synonyms of the given input word. This will be used is storing the synonyms of the given data set and query tokens. The synonyms will also be taken into consideration while performing similarity check for the sentences.

Once the similarity is stored, we apply the next level filter, function: filteredSentence , to apply lemmatization over stemmed tokens and again removing stop words. In the filtered sentence list, we now store the token word along with its synonyms for more precise matching/similarity check. Next, we put all these together. The training is done on two data set files, first **cricketbat.txt,** which contains few sentences referring to bat used in cricket sport, and second, **vampirebat.txt**, which contains few sentences referring to the mammal bird bat. sent1 stores the lowered case string data from the vampirebat.txt file and sent2 does for cricketbat.txt, sent3 stores user query. Next, the sentences are filtered and similarity is checked using the functions explained above. The comparison is normalised, and output is given accordingly whether the query refers to Cricket bator Mammal bat**.**

**4. Data Set details**

**4.1 WSD Training**

The training is done on two data set files, first **cricketbat.txt,** which contains few sentences referring to bat used in cricket sport, and second, **vampirebat.txt**, which contains few sentences referring to the mammal bird bat. sent1 stores the lowered case string data from the vampirebat.txt file and sent2 does for cricketbat.txt, sent3 stores user query. Next, the sentences are filtered and similarity is checked using the functions explained above. The comparison is normalised, and output is given accordingly whether the query refers to Cricket bator Mammal bat**.**

**5. Experiments**

# References

|  |  |
| --- | --- |
| [1] | J. T. a. X. W. Zhiwei Yu, ". A neural approach to pun generation," Perkin University, 2018. |
| [2] | S. L. X. S. B. C. Fuli Luo1, "Pun-GAN: Generative Adversarial Network for Pun Generation," Key Lab of Computational Linguistics, Peking University, China, 2019. |
| [3] | M. K. ˚. a. a. H. Salomonsson, " Word sense disambiguation using a bidirectional lstm," Computer Science & Engineering, Chalmers University of Technology, Sweden, 2016. |
| [4] | R. Y. G. L. L. Z. a. Z. J. Lili Mou, "Backward and Forward Language Modeling for Constrained sentence generation," Key Laboratory of High Confidence Software Technologies (Peking University),, China, 2015. |