Rare Query Expansion Through Generative Adversarial Networks in Search Advertising

Mu-Chu Lee*
Carnegie Mellon University
Machine Learning Department
Pittsburgh, PA
muchul@cs.cmu.edu

Bin Gao
Microsoft
Redmond, WA
bingao@microsoft.com

Ruofei Zhang
Microsoft
Sunnyvale, CA
bzhang@microsoft.com

ABSTRACT

Generative Adversarial Networks (GAN) have achieved great success in generating realistic synthetic data like images, tags, and sentences. We explore using GAN to generate bid keywords directly from query in sponsored search ads selection, especially for rare queries. Specifically, in the query expansion (query-keyword matching) scenario in search advertising, we train a sequence to sequence model as the generator to generate keywords, conditioned on the user query, and use a recurrent neural network model as the discriminator to play an adversarial game with the generator. By applying the trained generator, we can generate keywords directly from a given query, so that we can highly improve the effectiveness and efficiency of query-keyword matching based ads selection in search advertising. We trained the proposed model in the clicked query-keyword pair dataset from a commercial search advertising system. Evaluation results show that the generated keywords are more relevant to the given query compared with the baseline model and they have big potential to bring extra revenue improvement.

KEYWORDS

Query Keyword Matching, Generative Adversarial Networks, Online Advertising

ACM Reference Format:
Mu-Chu Lee, Bin Gao, and Ruofei Zhang. 2018. Rare Query Expansion Through Generative Adversarial Networks in Search Advertising. In *KDD '18: The 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, August 19–23, 2018, London, United Kingdom.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3219819.3219850

1 INTRODUCTION

In search advertising (or sponsored search), given a search query, we need to match it to some keywords bid by advertisers and then pull out the related ads for click prediction [13, 19] and ranking. The query keyword matching can be done by some simple matching rules like exact match, broad match, and phrase match, which are all based on matching the similar tokens shared by query and keywords. Besides these match types, smart match is an important yet difficult match type that can associate a query to some relevant keywords even they do not share many similar tokens.

It is well known that search queries follow the power law distribution [29]. That is, the most frequent queries compose the *head* and *torso* of the curve, while the low frequency rare queries make up the *tail* of the curve. Although they are individually rare, tail queries as a whole make up a significant portion of the query volume, and thus have great potential for advertising revenue. For ads selection in search advertising, exact match, broad match, and phrase match can meet most of the requirements for the head and torso queries [5]. For tail (or rare) queries, they are generally harder to predict and interpret, and in most cases there are no keywords or ads that are explicitly associated with them. Thus, we have to reply more on smart match for rare queries. Existing solutions for smart match include query keyword pair mining from various logs [10], query keyword similarity based on various word vectors like *word2vec* [21], machine translation models (such as sequence to sequence LSTM [30]), etc. All the above methods are challenging. For the pair mining models, due to the low volume of individual rare queries, it is difficult to accumulate enough ad clicks to use statistical and explore-exploit methods to identify high quality query keyword matching pairs. For similarity models, we have to calculate query keyword similarities online, which needs high computation cost. For machine translation models, there is still a big gap to improve for its accuracy as well, especially for rare queries.

To avoid the aforementioned problems, we describe a novel smart match model that can directly generate the related keywords for a given query based on the Generative Adversarial Networks (GAN) [11] framework. That is, instead of pair mining and similarity calculation, it directly generates the related keywords in moderate computation cost, and the generated keywords are of higher quality than the machine translation model.

GAN models, which have gained a lot of attention in recent years, have provided a new methodology of training generative models through the guide of a discriminator that is trained simultaneously. In GAN, the training process oscillates between the generator and the discriminator. The generator tries to generate samples that the discriminator cannot distinguish whether it is from the generated distribution or the true distribution. Such training process gives us means to come up with pseudo samples that could expand our data. In the original version of GAN, the generator takes a noise vector as input and generates an output sample. Thus, there is no control on the modes of the data being generated from the input random noise. To deal with this challenge, Conditional Generative Adversarial

*This work was completed during the first author's internship at Microsoft.

KDD '18, August 19–23, 2018, London, United Kingdom

ACM ISBN 978-1-4503-5552-0/18/08…$15.00
https://doi.org/10.1145/3219819.3219850

Nets (CGAN) [22] was proposed, in which by conditioning the model on additional information it is possible to direct the data generation process. Such conditioning could be based on class labels, on some part of data, or even on data from different modality.

In this paper, we propose a new framework called *QE-CGAN* to leverage the CGAN framework to conduct query expansion to match keywords for search ads selection. Specifically, for the generator, we use the sequence to sequence model, in which the input sequence is a random word vector followed by a query vector and the output sequence is the vector of the generated keyword; for the discriminator, we use the parallelized recurrent neural network (RNN) [20] model as a binary classifier. The clicked (query, keyword) pairs from the training data are regarded as positive samples for the discriminator, while both the non-clicked (query, keyword) pairs and the (query, generated keyword) pairs are regarded as negative samples for the discriminator. We leverage the policy gradient [31] strategy to train the model. After the training converges, we can use the generator for online query expansion. That is, given a user query, especially for rare query, we can use the generator to generate a group of keywords by different noise vectors that can match a variety of ads in the inventory. We call the proposed model as Query Expansion based on Conditional Generative Adversarial Networks, which is abbreviated as *QE-CGAN*. In the experiments, we trained the proposed QE-CGAN model in the clicked query-keyword data from a commercial search advertising system. Evaluation results show that the generated keywords (or query expansions) are more relevant to the given queries compared with the baseline model. Therefore, they can help increase the selection recall such that more relevant ads could be delivered to the users.

To sum up, our contributions are listed as below.

- We introduced the GAN framework to the area of computational advertising, which is by far the first GAN work in this area.
- The proposed QE-CGAN model showed good performance in query expansion, which opens a new window for us to enrich the ads selection models in sponsored search.
- The proposed QE-CGAN model might also be used in other query expansion tasks in general search engine and recommender systems.

The rest of the paper is organized as follows. We briefly review the related work in Section 2. In Section 3, we describe the novel QE-CGAN framework. Experimental evaluations are given in Section 4. Conclusions and future work are discussed in the last section.

2 RELATED WORK

2.1 Ads Selection

Existing works on ads selection are mainly based on query keyword (or ad) matching. The basic match types are exact match, broad match, and phrase match, which are all performed by simple rules. Besides, smart match is an important yet difficult match type that can connect a query to those keywords that are relevant but do not share many tokens with the query. Some smart match methods are relevance-based. For example, Broder *et al* [6] enriched both queries and ads with additional knowledge features to improve relevance matching. Broder *et al* [5] also proposed an alternative approach of matching the ads against rare queries. Xu *et al* [35] proposed an approach based on Wikipedia for query augmentation against rare queries in sponsored search. Choi *et al* [8] explored the usage of the landing page content to expand the text stream of bid keywords and ads. Some other methods employ the historical click information to mine the relationship among queries and keywords (ads). For example, Fuxman *et al* [10] conducted keyword suggestion by making use of the query logs of the search engine. In addition, Hillard *et al* [15] introduced a translation model between queries and keywords (ads) to score the query keyword (ad) relevance.

In another important group of methods, queries and keywords are represented by vectors such that we can conduct similarity search in the Euclid space to find the best matched keywords for a query. To reduce the computation cost of the similarity search, we usually leverage the word embedding models like SENNA [9], word2vec [21], GloVe [24], and FastText [4], to map queries and keywords into low-dimensional continuous spaces.

In search advertising industry, the ads selection model is often learned from the ads click data. Specifically, we can extract a set of query keyword pairs in which the associated ads are clicked by users. That is, for each query keyword pair in the set, a user submitted the query to the search advertising system, and then the query was matched to the keyword by some existing matching methods in the system. The keyword pulled out an ad, and the ad was ranked high and shown in the result page. Finally, the user clicked the ad. Similarly, we can also extract a set of query keyword pairs in which the associated ads are *not* clicked by users. Therefore, the set of clicked/non-clicked query keyword pairs can be used as the positive/negative samples to train a classifier like CDSSM [28]. Thus, given a new query, we can test it with the keyword candidates and match it to the keywords with positive test results. Recently, statistical translation models like LSTM [30] was also applied in the query keyword matching task. In these models, query is regarded as the source language and keyword is regarded as the target language, and then the clicked query keyword pairs can be used as the aligned source target pairs to train the machine translation model. Given a new query, we can directly *translate* it to a keyword.

2.2 Deep Learning

With the advance of deep learning, lots of directions of research and application have emerged. New techniques with deep learning, specifically recurrent neural networks (RNNs) [20] have shed light on topics that were deemed hard-to-solve, such as natural language processing, representation learning, etc.

A great focus in the research community of natural language processing is neural machine translations (NMT). A typical end-to-end neural machine translation system is built through an RNN-based encoder-decoder framework. In such framework, the encoder takes the input sequence and encode the sequence into a fixed-length vector as a latent representation. Following the attention mechanism [2] which is the mechanism for decoding the latent representation to a target sequence, the NMT framework is built upon the maximum likelihood estimation (MLE) for training. There are also other training criteria such as minimum risk training [25, 27] and translation reconstruction [32] to obtain more accurate error estimations.

2.3 Generative Adversarial Networks

Besides training with traditional methods, adversarial training [12] has become a new star for training. Adversarial training enjoys some appealing properties, and it provides a mean of likelihood-free training. For example, Generative adversarial networks proposed in [11] is a totally novel method of training generative models. The training procedure consists of two components, the *generator* and the *discriminator*,in which the optimization target is the min-max game between the two components. This framework has gain great success in image generation [37], but still lacks good results in sequence problem, more specifically, natural language processing tasks. However, there are still works such as SeqGAN [36] that tries to overcome the difficulty of training GAN on sequence tasks.

The difficulty of training GAN on sequence generation arises from the inability of back-propagating the error signal directly. A popular way to alleviate is that sequence generation could be viewed as a decision process through an agent (the generator) [1], which has the potential to be solved with reinforcement learning techniques. By modeling the generator as an agent following some policy to pick the next token, policy gradient [31] can be applied to optimize the generator. Reward for the generator could be obtained from the discriminator, which inherently works as a criteria of the min-max game. There also exist works [16, 18] that try to smooth discrete tokens to propagate error signal with Gumbel softmax [16, 18] and temperature annealing.

Another challenge exists where traditionally GAN takes a random Gaussian noise as a seed to generate some random output. This limits GAN to generate output that is out of user's control. Conditional generative adversarial networks [22] have attracted attention from different aspects. In [11, 14, 26], images are generated conditioned on some latent representation obtained from sentences. To a further extent, the work in [37] not only takes condition from the targeted sentence representation, but also takes condition from the whole generating process on the image itself. However, there are still very limited works [34] that involve conditions in sequence-to-sequence models, which we believe is due to the difficulty of training sequence problems and conditional GAN together.

3 QE-CGAN FORMULATION

In this section, we take the inspiration from the CGAN framework and build a novel CGAN based model for query expansion. We first review GAN and CGAN, and then we introduce the generator and the discriminator used in our framework. After that, we describe the proposed QC-GAN framework. Finally, we explain how we train the proposed model.

3.1 GAN and CGAN

Generative adversarial networks were introduced as a novel manner to train a generative model, in which there are two *adversarial* models: a generative model G that captures the data distribution, and a discriminative model D that estimates the probability that a sample came from the training data rather than G. The choices of G and D are flexible; for example, they could be non-liner mapping functions like deep neural networks.

To learn a generator distribution p_g over data x, the generator builds a mapping function from a pre-defined prior noise distribution $p_z(z)$ to data space as $G(z;\theta_g)$, where θ_g contains the parameters of the generator. The discriminator, $D(x;\theta_d)$, takes a sample from the training data of p_g and outputs a single scalar representing the probability that x came from training data rather than p_g. G and D are trained simultaneously: the parameters for G are adjusted to minimize $\log(1 - D(G(z)))$ and the parameters for D are adjusted to minimize $\log(1 - D(x))$. In other words, G and D are following the two-player min-max game with value function $V(G, D)$,

$$\begin{aligned}&\min_G \max_D V(G,D)\\ =&E_{x\sim p_{data}(x)}\left[\log D(x)\right] + E_{z\sim p_z(z)}\left[\log(1 - D(G(z)))\right]. \end{aligned} \quad (1)$$

Conditional generative adversarial nets are extended from the conventional GAN if both the generator and the discriminator are conditioned on some extra information y, which could be any kind of auxiliary information. By feeding y into both the generator and the discriminator as additional input layer, we can get the objective function of CGAN as,

$$\begin{aligned}&\min_G \max_D V(G,D)\\ =&E_{x\sim p_{data}(x)}\left[\log D(x|y)\right] + E_{z\sim p_z(z)}\left[\log(1 - D(G(z|y)))\right]. \end{aligned} \quad (2)$$

3.2 Generator: Sequence-to-Sequence Model

In the task of query expansion for ads selection, we need a generator that could produce a bid keyword from a query provided by a user. For example, the query could be *koi fish* and the bid keyword might be *fish pond supply*. In this scenario, some merchant that sells fish pond supplies might bid the keyword *fish pond supply* for a certain amount of money. The ads selection system correlates the query *koi fish* to the keyword *fish pond supply*, so that the advertisement of the corresponding merchant might show along with the search results of the query and the user would probably click the advertisement.

As the query and the keyword might have different numbers of word tokens, we adopt the Recurrent Neural Network (RNN) based encoder-decoder sequence-to-sequence (seq2seq) model as our base model for the generator. As usual, instead of using one-hot vector representation, we use a pre-trained distributed word representation (or word embedding) to denote the word tokens. Thus, each token is represented by a low-dimension real-valued vector.

Figure 1 shows the structure of the generator model. Suppose the query q contains a sequence of tokens $q_1, q_2, \cdots, q_T$, and the keyword k contains a sequence of tokens $k_1, k_2, \cdots, k_{T'}$. In the encoding process, the standard RNN encoder model computes the thought vector by iterating the following equation,

$$h_t = \sigma_h(W_h q_t + U_h h_{t-1}), \quad (3)$$

where h_t is the hidden layer vector, W_h and U_h are parameter matrices, and σ_h is the activation function. In our generator model, the input is a noise vector z conditioned on the query q. As shown in Figure 1, we choose a simple parallel connection of z and q to compute the thought vector s using the following equation,

$$s = \sigma_z(U_h h_t + U_z z), \quad (4)$$

where U_z is the parameter matrix and σ_z is the activation function.

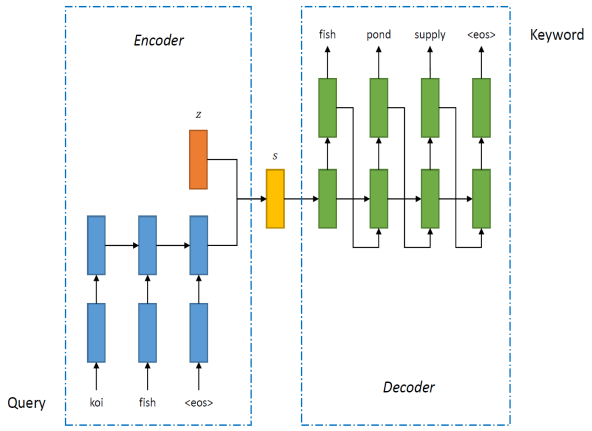

Figure 1: The generator: sequence-to-sequence model.

In the decoding process, the goal is to estimate the conditional probability $p(k_1, k_2, \cdots, k_{T'}|z, q_1, q_2, \cdots, q_T)$. The RNN decoder model computes this conditional probability with a standard language model formulated as below, in which the initial hidden state is set to the thought vector s. That is, the hidden state of decoder at time t is s_t and we let $s_0 = s$.

$$\begin{aligned} & p(k_1, k_2, \cdots, k_{T'}|z, q_1, q_2, \cdots, q_T) \\ = & \prod_{t=1}^{T'} p(k_t|s_{t-1}, k_1, k_2, \cdots, k_{t-1}). \end{aligned} \tag{5}$$

Further simplifying the above language model to the first-order language model, we can write the generator as,

$$\begin{aligned} k &= G(z|q) \\ &\propto p(k_1, k_2, \cdots, k_{T'}|z, q_1, q_2, \cdots, q_T) \\ &= \prod_{t=1}^{T'} p(k_t|s_{t-1}, k_1, k_2, \cdots, k_{t-1}) \\ &= \prod_{t=1}^{T'} p(k_t|s_{t-1}, k_{t-1}), \end{aligned} \tag{6}$$

$$s_t = g(s_{t-1}, k_t, c_t), \tag{7}$$

where g is a certain type of RNN model such as Long Short Term Memory (LSTM) [30] and Gated Recurrent Unit (GRU) [7], and c_t is an attention mechanism [2] that can help improve the model performance. More details about the attention mechanism will be explained in Section 4.3.

Methods of choosing k_t from $p(k_t|s_{t-1}, k_{t-1})$ have been discussed in various works for both the training process and the testing process. During the training process, some methods greedily choose the largest output from $p(k_t|s_{t-1}, k_{t-1})$. However, Bengio *et al* [3] argued that the decoder would not learn well if we only feed the groundtruth as input, while a linear scheduling of feeding the previous step's output and groundtruth would better boost the testing performance. During the testing process, we could greedily feed the previous output k_{t-1} at time t as the next input, or we could perform beam search [33] to keep track of the optimal output. As we aim to generate multiple outputs, we choose to direct our input from the output words of the previous step with respect to the word probability. To do this efficiently, we use the Gumbel trick [16, 18].

Let π_i be the probability of the i-th word in the dictionary being chosen at time t,

$$\pi_i = \frac{e^{w_i}}{\sum_{j=1}^{|W|} e^{w_j}}, \tag{8}$$

where w_i denotes the log-unnormalized probability of the i-th word in the dictionary, and $|W|$ denotes the dictionary size.

The probability density function and the cumulative distribution function of Gumbel distribution are,

$$f(\epsilon; \mu) = e^{-(\epsilon-\mu)-e^{-(\epsilon-\mu)}}, \tag{9}$$

$$F(\epsilon; \mu) = e^{-e^{\epsilon-\mu}}, \tag{10}$$

where μ is the location parameter of Gumbel distribution[1].

By applying the Gumbel trick, we have the Gumbel random variable $\mathbb{G} = -log(-log(\mathbb{U}))$ with $\mathbb{U}$ being a uniform distribution, i.e. $\mathbb{U} \sim uniform(0, 1)$. Suppose $\mathbb{G}_1, \mathbb{G}_2, \cdots, \mathbb{G}_{|W|}$ are i.i.d. from $\mathbb{G}$, we have

$$\epsilon_i = \mathbb{G}_i + log(w_i). \tag{11}$$

Then, suppose we have the outcome of our Gumbel ϵ_i with location $\mu = w_i$, the probability that ϵ_i is the greatest would be,

$$\begin{aligned} & P(\epsilon_i \text{ is } \max|\epsilon_i, \{w_j\}_{j=1}^{|W|}) \\ = & \prod_{j \neq i} e^{-e^{\epsilon_i - w_j}} \\ = & \int e^{-(\epsilon_i - w_i) - e^{\epsilon_i - w_i}} \prod_{j \neq i} e^{-e^{\epsilon_i - w_j}} d\epsilon_i \\ = & \int e^{-\epsilon_i + w_i - e^{-\epsilon_i} \sum_{j=1}^{|W|} e^{w_i}} d\epsilon_i \\ = & \frac{e^{w_i}}{\sum_{j=1}^{|W|} e^{w_j}}, \end{aligned} \tag{12}$$

which allows us to sample directly from the simplex π.

3.3 Discriminator: Parallelized RNN Model

In the task of query expansion for ads selection, given a query-keyword pair (q, k), the discriminator should justify whether the keyword k can be associated with any ads that will be clicked by the user who issued the query q. To fulfill this task, we can use a classifier to judge the query keyword pair.

As aforementioned in Section 2.1, we can choose CDSSM [28] as the classifier. However, in several works about GAN training [11, 14, 26], it has been noticed that maintaining the balance between the generator and the discriminator is a crucial factor for successful training. In most failed GAN training, the discriminator was so strong that the generator could by no means fool the discriminator. In our initial experiments, we also observed that CDSSM is a too strong discriminator to successfully train the GAN model.

Therefore, instead of using CDSSM, we take a much simplified model shown in Figure 2, which is a parallelized RNN model. In this model, query is fed into one RNN structure and keyword is

[1] https://en.wikipedia.org/wiki/Gumbel_distribution

fed into another RNN structure. The thought vectors of these two RNNs are then fully connected to a hidden layer, and the hidden layer produces the final prediction (1/0) output.

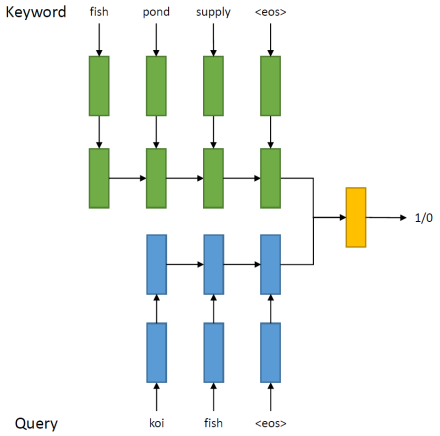

Figure 2: The discriminator: parallelized RNN model.

3.4 QE-GAN

The proposed QE-GAN framework is shown in Figure 3, which consists of two parts: the generator G and the discriminator D. The generator $G(z|q;\theta_g)$ is a sequence-to-sequence model illustrated in Section 3.2, which in our design is a bi-directional GRU parameterized by θ_g. The discriminator $D(k|q;\theta_d)$ is a parallelized RNN model illustrated in Section 3.3, which in our design is a one-directional GRU parameterized by θ_d.

Till now, we can write the objective function of our proposed QE-GAN framework as below, in which the two objectives reflect the adversarial game between G and D,

$$\begin{aligned}&\min_G \max_D V(G,D)\\ =&E_{(q,k)\sim p_{data}(q,k)}\left[\log D(k|q)\right]\\ &+E_{z\sim p_z(z)}\left[\log(1-D(G(z|q)|q))\right].\end{aligned} \tag{13}$$

However, such objective function comes up with a flaw in our task. The discriminator is rewarded if it could discriminate between a generated (q,k) pair and a true (q,k) pair. The flaw is that this does not necessary mean the discriminator is good enough. Given a (q,k) pair which is non-clicked from the true distribution, the discriminator should still be penalized if it deems this as a good pair. Our discriminator should be able to discriminate not only generated or non-generated pairs, but also clicked or non-clicked pairs. Therefore, we add a penalty term in our final objective function,

$$\begin{aligned}&\min_G \max_D V(G,D)\\ =&E_{(q,k)\sim p_{data}(q,k)}\left[\log D(k|q)\right] &(14)\\ &+E_{z\sim p_z(z)}\left[\log(1-D(G(z|q)|q))\right] &(15)\\ &+E_{(q,k)\sim p_{data}(q,k)}\left[\log(1-D(\tilde{k}|q))\right]. &(16)\end{aligned}$$

Here Eq.(14) and Eq.(15) are the objectives for optimizing the *min-max game* of G and D. These loss terms penalize the discriminator D if it is unable to tease out the input from generator G and true distribution. The added loss term is Eq.(16), where $\tilde{k}$ represents the keywords that are non-clicked under user query q. This objective equips the discriminator D with the force that D may not only reject the generator's distribution but also the *incorrectness* from the true distribution.

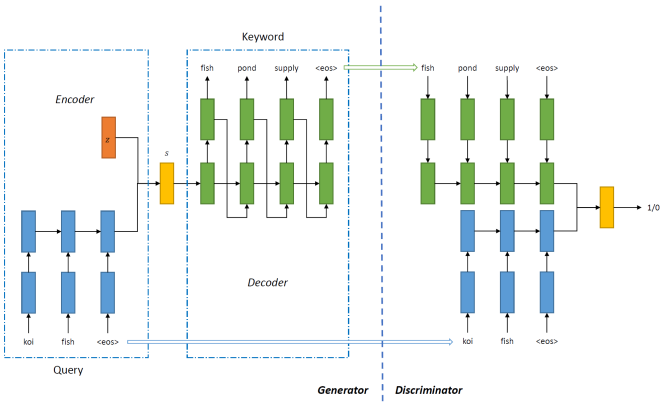

Figure 3: The QE-CGAN framework.

3.5 Policy Gradient for Model Training

The nature of GAN makes training discrete output a difficult task. It has been shown that the back propagation will lead to some failure in training. In other words, when G outputs a discrete sequence $k' = G(z|q)$, it is non-trivial to back-propagate directly from D to G. To remedy the difficulty of back propagation, we choose to use policy gradient [31], which is a popular method in reinforcement learning when no intermediate rewards are available.

Specifically, we view this process as a reinforcement learning problem, where we can take $G(z;\theta_g)$ as a policy with parameter θ_g. We can then solve this problem through policy gradient, more specifically, the REINFORCE [31] algorithm. When we do not have intermediate rewards during training, we can instead adopt the Monte-Carlo method to obtain an intermediate reward to optimize $G(z;\theta_g)$.

When we train G under some user query q and fixed D, we are minimizing the following loss of G,

$$L = E_{k'\sim G(\cdot|q)}\left[\log(1-D(k'|q))\right]. \tag{17}$$

The gradient of the loss L with respect to the parameter θ_g is,

$$\begin{aligned}&\nabla_{\theta_g} L\\ =&\nabla_{\theta_g} E_{k'\sim G(\cdot|q)}\left[\log(1-D(k'|q))\right] &(18)\\ =&E_{k'\sim G(\cdot|q)}\left[\log(1-D(k'|q))\nabla_{\theta_g} logG(k'|q)\right]. &(19)\end{aligned}$$

In Eq.(18), it is impossible for us to evaluate the gradient, since taking the gradient of the expectation requires an explicit evaluation. However, by Eq.(19), we would be able to approximate the gradient by using the Monte-Carlo sampling technique. The gradient of $logG(k'|q)$ is trivial for us, and thus we can sample k' from $G(z;\theta_g)$ to approximately obtain the expected value that could be

Table 1: The statistics of the training set.

	Size	Vocabulary	Average Length
Query	14,636,850	822,990	3.76
Keyword	4,012,500	280,140	3.06

used for evaluating the gradient of L. Figure 4 shows how the gradient is approximated. The generator generates multiple samples, and obtains the value from the discriminator. We could then use those intermediate rewards $(R_1, R_2, \cdots)$ to evaluate our policy and approximate the policy gradient.

We could then update the parameter θ_g by,

$$\theta_g \leftarrow \theta_g - \alpha \nabla_{\theta_g}, \tag{20}$$

where α is a carefully chosen step size.

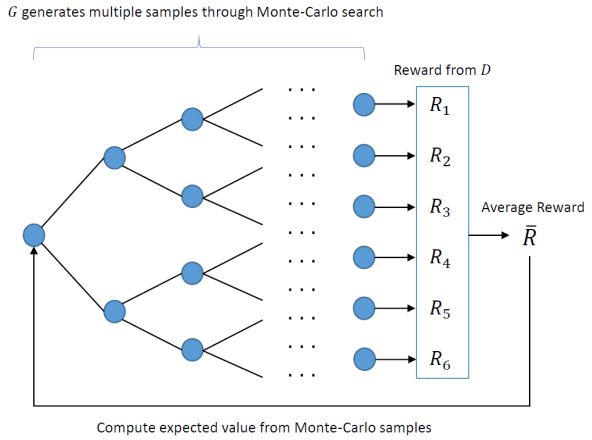

Figure 4: The Monte-Carlo Search Process.

4 EXPERIMENTS

In this section, we conduct experiments to show the performance of the proposed QE-CGAN framework in generating reasonable query expansions (or keywords) for sponsored search ads selection.

4.1 Dataset

We used the user click logs sampled from a commercial search advertising system as the training data. A total of about 24 million clicked query keyword pairs are sampled from the click logs from May 2016 to April 2017. We did token normalization (e.g., converting to lower-cased tokens, dropping special characters, etc.) and represented each query (and keyword) as a sequence of uni-gram terms. This yields a dictionary size of around 822,990 for queries and 280,140 for keywords. The average length of queries and keywords are 3.76 and 3.06 tokens, respectively. We summarize the statistics of the training set in Table 1.

4.2 QE-CGAN Implementation Details

We used PyTorch [23] for the QE-CGAN implementation and the experiments were run on a single processor machine equipped with an 8-core Intel Xeon E5-2670 clocked at 2.60Ghz. The machine has 32GB of RAM, and has a NVIDIA K80 GPU.

For the generator, we used a two-layer bi-directional GRU, with a hidden size of 500. We also applied a dropout rate of 0.1 as regularization. During the decoding phase, we connected our network to a V-dimensional softmax layer, where V is the overall vocabulary size for queries and keywords. The softmax layer calculates the probability of each word in the vocabulary to be the output. As a large V would be computationally infeasible, we trimmed our vocabulary set. We substituted tokens that appeared less than 50 times with the conventional <UNK> token, and obtained a final dictionary size of 60,792.

For the discriminator, to keep a better balance with the generator, the discriminator has a simpler model. The discriminator is a single layer one-directional GRU. The hidden size of the discriminator is 150, and the output is the cosine similarity of the 150-dimensional vector. The optimizer we used is Adam [17] with the learning rate 10^{-3}.

Although GANs are well-known for its difficulties in training, pre-training has shown some benefits for the training process [26]. Therefore, we first pre-trained our sequence-to-sequence generator model for 2 epochs, by the Adam optimizer [17] with learning rate of 10^{-3}.

In addition, the pre-training also helped us produce the initial word embedding for the follow-up GAN training. Furthermore, after the pre-training process, we would freeze the word embedding weights. Note that this is especially crucial for the GAN training. The reason is that, during the GAN training, there should not be any communication between the generator and the discriminator, other than the output of generator. If a side channel exists (e.g., the word embedding), the generator could simply set all the embedding values to zero, and it would be impossible for the discriminator to distinguish between the true and false samples.

4.3 Baseline Method

For the baseline method, we trained a classic sequence to sequence translation model with attention mechanism [2], denoted by *S2S*, using the above clicked query keyword dataset.

Specifically, we suppose the query q contains a sequence of tokens $q_1, q_2, \cdots, q_T$, and the keyword k contains a sequence of tokens $k_1, k_2, \cdots, k_{T'}$. In our implementation, we used bi-directional GRU [7] as the base RNN unit.

In one direction, the encoder takes the input q sequentially, from $q_1, q_2, \cdots, q_T$, resulting in the *forward hidden states* $(\overrightarrow{h_1}, \cdots, \overrightarrow{h_T})$; in the other direction, it takes the input with the reverse order $q_T, \cdots, q_1$, resulting in the *backward hidden states* $(\overleftarrow{h_T}, \cdots, \overleftarrow{h_1})$. The hidden states for the same i are concatenated together, such that for each i, we have $h_i = (\overrightarrow{h_i}, \overleftarrow{h_i})$. This hidden state contains the information focusing on q_i.

The decoder calculates the following probability of a word,

$$p(k_i|q_i, k_1, k_2, \cdots, k_{i-1}) = g(k_{i-1}, s_i, c_i), \tag{21}$$

where s_i is the decoder RNN hidden state computed by,

$$s_i = f(k_{i-1}, s_{i-1}, c_i), \tag{22}$$

where c_i is the attention vector for the sequence $(h_1, \cdots, h_T)$.

Table 2: Comparison with human labeling results.

	Excellent	Good	Fair	Bad	Not Sure
S2S	179	473	93	137	29
QE-CGAN	195	486	67	124	39

The attention vector is computed through the weighted sum of the encoder's hidden state,

$$c_i = \sum_{j=1}^{T} \alpha_{ij} h_j, \tag{23}$$

by finding the weight α_{ij} with a softmax probability

$$\alpha_{ij} = \frac{e^{b_{ij}}}{\sum_{k=1}^{T} e^{b_{ik}}}, \tag{24}$$

where b_{ij} is the score indicating how much attention should be focused around. In practice, b_{ij} can be computed in various methods. In our implementation, we used the most common method, i.e.,

$$b_{ij} = a(s_{i-1}, h_j), \tag{25}$$

where the function $a(\cdot, \cdot)$ is parameterized by a feed forward neural network jointly trained with the whole model.

4.4 Evaluation Results

In search advertising, the queries are assigned with deciles. Usually, we sort all the queries in the descending order by their frequencies in a certain period of time (e.g., 30 days) and assign them to 10 deciles. The accumulated query frequency of each decile is 10% of the total query frequency. Therefore, decile 1 contains the most popular queries, and decile 10 contains the rarest queries; the number of unique queries in each decile increases from decile 1 to decile 10. In the task of ads selection, most head and torso queries (decile 1 to decile 5) can be solved by exact match, broad match, and phrase match; however, most tail queries (decile 6 to decile 10) have to be solved by smart match. As the proposed QE-CGAN model is mainly targeted at smart match for rare queries, we will evaluate its performance on the tail queries deciles.

Specifically, we randomly sampled 1,000 queries from decile 6 to decile 10, and used these queries as the input to the S2S and QE-CGAN models to generate output keywords. The generated query keyword pairs were sent to professional human judges to tag five grade labels: *excellent*, *good*, *fair*, *bad*, and *not sure*. Each pair was labeled by three different human judges, and the final label was obtained by voting. If a pair got three different labels, we would drop it. As a result, we finally got 911 queries with the labeled query keyword pairs for both models.

Table 2 shows the labeling statistics of the S2S model and the QE-CGAN model. From the results, we can see that the numbers of *excellent* and *good* query keyword pairs of QE-CGAN are larger than those of S2S, while the number of *bad* pairs of QE-CGAN is smaller than that of S2S. Therefore, QE-CGAN can produce better keywords than S2S for smart match on rare queries.

Table 3 and 4 shows the decile level comparison of the S2S model and the QE-CGAN model. From these results, we can see that QE-CGAN works much better than S2S especially in very tail deciles.

Table 3: Human labeling results for *S2S* in deciles.

	Excellent	Good	Fair	Bad	Not Sure
Decile 6	65	126	22	33	12
Decile 7	55	130	33	39	8
Decile 8	23	100	18	34	5
Decile 9	13	32	5	15	0
Decile 10	23	85	15	16	4

Table 4: Human labeling results for *QE-CGAN* in deciles.

	Excellent	Good	Fair	Bad	Not Sure
Decile 6	56	109	21	59	13
Decile 7	63	145	21	27	9
Decile 8	33	106	11	21	9
Decile 9	11	43	3	5	3
Decile 10	32	83	11	12	5

4.5 Case Study

Table 5 shows the generated keywords by the two models for some given queries, in which we randomly sampled five comparison cases for each decile from decile 6 to decile 10. From these showcases, we have the following observations:

- Generally speaking, the generated keywords by the two models both look reasonable. However, if we study them carefully, we can find that majority of keywords by QE-CGAN are better than those by S2S. For example, for query *add hp printer computer*, the generated keyword *install my hp printer* by QE-CGAN is better than *hp printer drivers* by S2S, as the former produces the action verb *install*, reflecting *add* in the query. For query *acrobat windows 10*, QE-CGAN outputs *adobe acrobat standard download*, in which there is a safe extra word *standard*, but S2S outputs *adobe acrobat 8.1 update download*, in which the word *update* might misunderstand the query intent. For query *car rentals puerto rico san juan airport*, QE-CGAN drops *puerto rico* as San Juan is the capital of Puerto Rico, which does not hurt the query intent; however, S2S drops *airport*, which changes the query intent to a much broader range. These cases show that the adversarial game may help keep the query intent when generating the query expansion.
- QE-CGAN can generate some extra beneficial words that often well match the query intent, such as *discount* in *orlando theme parks discount tickets*, *free* in *free online cv templates*, and *used* in *mercedes benz e class used*. These cases show that GE-CGAN can learn some good extension patterns to enlarge the coverage in the keyword set.
- In some cases, S2S would misunderstand the query and output keywords that are quite far away from the original query intent. For example, for query *dog car ramps large dogs*, S2S generates *dog wheelchair* which is quite different with *dog ramp*; for query *walmart mini hot dogs buns*, S2S produces *walmart bedroom sets*, which is quite irrelevant to the original query intent. Meanwhile, QE-CGAN outputs good keywords for both of the above cases.

Table 5: Keywords generated by *S2S* and *QE-CGAN*, together with the human labels.

Input Query	Output Keywords by S2S	Label	Output Keywords by QE-CGAN	Label
Decile 6				
add hp printer computer	hp printer drivers	Fair	install my hp printer	Excellent
best diet drinks	best weight loss tip	Fair	greens drink superfood	Good
gap women	gap womens clothing	Excellent	gap gap gap	Good
spirit airlines promo codes 2017	swiss international air lines 10 off	Fair	extremely cheap airfare	Good
jazz music online	jazz music festival tickets	Fair	jazz music	Excellent
Decile 7				
acrobat windows 10	adobe acrobat 8.1 update download	Fair	adobe acrobat standard download	Good
2017 minivans	minivan	Good	minivans minivans	Good
theme park tickets orlando	orlando theme parks hotel	Fair	orlando theme parks discount tickets	Excellent
dog car ramps large dogs	dog wheelchair	Fair	dog ramp	Good
housing interest rates today	low income apartments	Fair	mortgage rates	Good
Decile 8				
condos rent st augustine beach florida	st augustine condo rentals	Good	saint augustine beach condo rental	Excellent
heart disease kids	heart disease children	Excellent	heart disease early	Good
online cv template	cv templates	Good	free online cv templates	Excellent
handyman hub	handyman list	Fair	handyman service	Good
legal aid divorce forms	legal zoom com	Fair	divorce forms	Good
Decile 9				
car rentals puerto rico san juan airport	san juan puerto rico car rentals	Excellent	san juan airport car rental	Excellent
tile paneling bathrooms	bathroom wall tile	Good	bathroom tile	Good
under armor gear phone cases	shad top case	Bad	under armor cases	Good
weakness legs elderly	leg weakness legs	Good	causes legs legs	Good
out town movers	move helper	Excellent	2 men truck	Good
Decile 10				
free songs listen online	listen unlimited free music	Good	listen free music	Excellent
2012 mercedes e class sale	mercedes benz e class	Good	mercedes benz e class used	Good
top rated convertible cars	convertible top	Bad	best car convertible	Excellent
dvd player repair la crosse wi	dvd drive not reading	Fair	dvd player repair	Good
walmart mini hot dogs buns	walmart bedroom sets	Bad	mini buns buns	Good

- In some other cases, QE-CGAN can produce some relevant keywords that have the same meanings with the input query but write differently. For example, for query *housing interest rates today*, it produces *mortgage rates*; for query *handyman hub*, it produces *handyman service*. These cases show that GE-CGAN can produce more diverse keywords to extend the relevant ads coverage.
- There is an interesting observation that QE-CGAN would sometimes output some duplicated tokens like *gap gap gap*, *minivans minivans*, *causes legs legs*, and *mini buns buns*. Although these duplicated tokens are good according to the query, they do not bring extra benefit. These cases can be avoided by placing some constraints in the sequence generation process. For example, we can put an additional condition in Eq.(6) to exclude the same token with k_{t-1} in the generation of k_t.

To sum up, we can see that QE-CGAN performs better than S2S on generating relevant keywords conditioned on the input queries. The original queries from the users are often very ad-hoc, but the rewritten results by QE-CGAN are usually more concise and meaningful so that they should come up with better coverage in the keyword set as well as the ads corpus.

5 CONCLUSIONS AND FUTURE WORK

We presented the conditional generative adversarial networks framework for query expansion in search ads selection. In this framework, the generator is a sequence-to-sequence model that takes a noise vector as input conditioned on a user query and output an advertiser bid keyword; the discriminator is a parallelized recurrent neural network that judges the valid query keyword pair from in invalid pairs. We leveraged the policy gradient strategy and the REINFORCE algorithm in the adversarial model training. Experimental results have shown that the generated keywords are much more relevant to the given query compared with the baseline model.

For the future work, we would like to test the QE-CGAN model in the online ads selection system, to check its revenue contribution. We would also like to explore more choices of the generator and the discriminator, to seek a better CGAN design that can bring even

better query expansion results. In addition, we would like to test the QE-CGAN model on other tasks like the query rewriting in recommender systems.

REFERENCES

[1] Philip Bachman and Doina Precup. 2015. Data Generation as Sequential Decision Making. *CoRR* abs/1506.03504 (2015). arXiv:1506.03504 http://arxiv.org/abs/1506.03504

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural Machine Translation by Jointly Learning to Align and Translate. *CoRR* abs/1409.0473 (2014). arXiv:1409.0473 http://arxiv.org/abs/1409.0473

[3] Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. 2015. Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks. *CoRR* abs/1506.03099 (2015). arXiv:1506.03099 http://arxiv.org/abs/1506.03099

[4] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2016. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606* (2016).

[5] Andrei Broder, Peter Ciccolo, Evgeniy Gabrilovich, Vanja Josifovski, Donald Metzler, Lance Riedel, and Jeffrey Yuan. 2009. Online expansion of rare queries for sponsored search. In *Proceedings of the 18th international conference on World wide web*. ACM, 511–520.

[6] Andrei Z Broder, Peter Ciccolo, Marcus Fontoura, Evgeniy Gabrilovich, Vanja Josifovski, and Lance Riedel. 2008. Search advertising using web relevance feedback. In *Proceedings of the 17th ACM conference on Information and knowledge management*. ACM, 1013–1022.

[7] KyungHyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. *CoRR* abs/1409.1259 (2014). arXiv:1409.1259 http://arxiv.org/abs/1409.1259

[8] Yejin Choi, Marcus Fontoura, Evgeniy Gabrilovich, Vanja Josifovski, Mauricio Mediano, and Bo Pang. 2010. Using landing pages for sponsored search ad selection. In *Proceedings of the 19th international conference on World wide web*. ACM, 251–260.

[9] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *Journal of Machine Learning Research* 12, Aug (2011), 2493–2537.

[10] Ariel Fuxman, Panayiotis Tsaparas, Kannan Achan, and Rakesh Agrawal. 2008. Using the wisdom of the crowds for keyword generation. In *Proceedings of the 17th international conference on World Wide Web*. ACM, 61–70.

[11] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in neural information processing systems*. 2672–2680.

[12] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and Harnessing Adversarial Examples. *CoRR* abs/1412.6572 (2014). arXiv:1412.6572 http://arxiv.org/abs/1412.6572

[13] Thore Graepel, Joaquin Quinonero Candela, Thomas Borchert, and Ralf Herbrich. 2010. Web-scale bayesian click-through rate prediction for sponsored search advertising in microsoft's bing search engine. Omnipress.

[14] Ishaan Gulrajani, Faruk Ahmed, Martín Arjovsky, Vincent Dumoulin, and Aaron C. Courville. 2017. Improved Training of Wasserstein GANs. *CoRR* abs/1704.00028 (2017). arXiv:1704.00028 http://arxiv.org/abs/1704.00028

[15] Dustin Hillard, Stefan Schroedl, Eren Manavoglu, Hema Raghavan, and Chirs Leggetter. 2010. Improving ad relevance in sponsored search. In *Proceedings of the third ACM international conference on Web search and data mining*. ACM, 361–370.

[16] Eric Jang, Shixiang Gu, and Ben Poole. 2017. Categorical Reparameterization with Gumbel-Softmax. https://arxiv.org/abs/1611.01144

[17] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *CoRR* abs/1412.6980 (2014). arXiv:1412.6980 http://arxiv.org/abs/1412.6980

[18] Matt J. Kusner and José Miguel Hernández-Lobato. 2016. GANS for Sequences of Discrete Elements with the Gumbel-softmax Distribution. *CoRR* abs/1611.04051 (2016). arXiv:1611.04051 http://arxiv.org/abs/1611.04051

[19] H Brendan McMahan, Gary Holt, David Sculley, Michael Young, Dietmar Ebner, Julian Grady, Lan Nie, Todd Phillips, Eugene Davydov, Daniel Golovin, et al. 2013. Ad click prediction: a view from the trenches. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1222–1230.

[20] Tomáš Mikolov. 2012. Statistical language models based on neural networks. *Presentation at Google, Mountain View, 2nd April* (2012).

[21] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[22] Mehdi Mirza and Simon Osindero. 2014. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784* (2014).

[23] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. (2017).

[24] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 1532–1543.

[25] Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. 2015. Sequence Level Training with Recurrent Neural Networks. *CoRR* abs/1511.06732 (2015). arXiv:1511.06732 http://arxiv.org/abs/1511.06732

[26] Tim Salimans, Ian J. Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. 2016. Improved Techniques for Training GANs. *CoRR* abs/1606.03498 (2016). arXiv:1606.03498 http://arxiv.org/abs/1606.03498

[27] Shiqi Shen, Yong Cheng, Zhongjun He, Wei He, Hua Wu, Maosong Sun, and Yang Liu. 2015. Minimum Risk Training for Neural Machine Translation. *CoRR* abs/1512.02433 (2015). arXiv:1512.02433 http://arxiv.org/abs/1512.02433

[28] Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Grégoire Mesnil. 2014. A Latent Semantic Model with Convolutional-Pooling Structure for Information Retrieval. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management (CIKM '14)*. ACM, New York, NY, USA, 101–110. https://doi.org/10.1145/2661829.2661935

[29] Amanda Spink, Dietmar Wolfram, Major BJ Jansen, and Tefko Saracevic. 2001. Searching the web: The public and their queries. *Journal of the Association for Information Science and Technology* 52, 3 (2001), 226–234.

[30] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*. 3104–3112.

[31] Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. 1999. Policy Gradient Methods for Reinforcement Learning with Function Approximation. In *Proceedings of the 12th International Conference on Neural Information Processing Systems (NIPS'99)*. MIT Press, Cambridge, MA, USA, 1057–1063. http://dl.acm.org/citation.cfm?id=3009657.3009806

[32] Zhaopeng Tu, Yang Liu, Lifeng Shang, Xiaohua Liu, and Hang Li. 2016. Neural Machine Translation with Reconstruction. *CoRR* abs/1611.01874 (2016). arXiv:1611.01874 http://arxiv.org/abs/1611.01874

[33] Sam Wiseman and Alexander M. Rush. 2016. Sequence-to-Sequence Learning as Beam-Search Optimization. *CoRR* abs/1606.02960 (2016). arXiv:1606.02960 http://arxiv.org/abs/1606.02960

[34] Lijun Wu, Yingce Xia, Li Zhao, Fei Tian, Tao Qin, Jianhuang Lai, and Tie-Yan Liu. 2017. Adversarial Neural Machine Translation. *CoRR* abs/1704.06933 (2017). arXiv:1704.06933 http://arxiv.org/abs/1704.06933

[35] Zhuoran Xu, Xiangzhi Wang, and Yong Yu. 2011. Rare Query Expansion via Wikipedia for Sponsored Search. In *Knowledge Engineering and Management*. Springer, 521–530.

[36] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. 2017. SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient.. In *AAAI*. 2852–2858.

[37] Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaolei Huang, Xiaogang Wang, and Dimitris N. Metaxas. 2016. StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks. *CoRR* abs/1612.03242 (2016). arXiv:1612.03242 http://arxiv.org/abs/1612.03242