Final Report

Boya Song bs3065 | Xi Chen xc2403 | Weimeng Luo wl2621

1. Introduction

Generating synthetic sequential text with good quality is an important research problem in unsupervised learning. Recurrent neural networks (RNNs) [7] along with its variant long short-term memory (LSTM) [9] cells have shown an excellent ability in generating sequential text. In the previous works, the most common way to train the RNN is an approach named *teacher forcing* via maximum likelihood estimation (MLE). These methods are designed to optimize perplexity, but it may lead to a problem called *exposure bias* where the model is only exposed to the training data distribution, rather than its prediction distribution. Methods like *professor forcing* [11] and *Scheduled Sampling*[3] have been proposed to address the problem, by not directly specifying a cost function to encourage high quality samples.

Generative adversarial networks (GANs) [6] are growing very fast in recent years and succeeded in generating images. In GANs, there is a generator and a discriminator. The generator generates faked samples to fool the discriminator, while the discriminator separates the real samples from the dataset and faked samples from the generator.

However, there are three problems in applying GANs to text generation task. First, since text sequence consists of discrete tokens, it is non-differentiable and makes it hard to propagate gradients from the discriminator to the generator. Second, the discriminator only generates one probability to a sentence about whether it is real or not. Since a sentence consists of multiple tokens, the generator may not be able to get sufficient from the discriminator and improves its performance. Third, GANs also suffer from mode dropping. Language has a great variety of modes, such as phrases, idioms, slangs. It is hard for GANs to learn all of them.

To address the first problem, we use Reinforcement Learning to train the generator. The rewards are defined based on the discriminator output and Actor-Critic algorithm is used for policy gradients. To solve the second and third problem, we train the model on a text fill-in-the-blank task, which is to fill in the deleted or modified portions of text while keeping some portions of the original texts from the training corpus. Furthermore, the discriminator provides rewards for each generated token instead of the whole sentence, which gives the generator more signals to learn.

In this work, we reproduce the work by Fedus et. al [4][1]. A model called MaskGAN is proposed to adopt GANs for text generation, and use *reinforcement learning* (RL) to train the generator while using the traditional MLE method and stochastic gradient decent to train the discriminator.

We trained MaskGAN using the Penn Treebank (PTB) corpus and obtained some similar results as shown in the paper[4]. In order to improve upon MaskGAN, we further propose another preprocessing method and call the model trained with this preprocessing method PadGAN. We compare the performance of four models: MaskMLE, MaskGAN, PadMLE, PadGAN, where MLE models are trained via MLE. The evaluation results showed that most of the generated text sequences are semantically and syntactically correct. Also, GAN models generates texts with higher quality and Pad models improve the quality of generated text over Mask Models.

2. Related work

GANs are very useful in generating real-valued data, but it has limitations when dealing with discrete tokens due to their non-differentiable nature. Researchers often try to avoid this issue by working in a continuous space, or reformulate the problem by solving it with reinforcement learning method.

SeqGAN[20] solves this problem by directly performing gradient policy update. This work is employing a policy gradient and Monte Carlo search based on the expected reward from the discriminative model D_ϕ, to the generative model G_θ. Besides that, both the generator and discriminator are pre-trained using MLE on the fake and real data before policy gradient.

Lamb [11] proposed *professor forcing*, as an alternative way of training RNNs. *professor forcing* seeks to match the behavior of generator and the teacher-forcing behavior as much as possible.

Li et al.[12] applied GANs to dialogue generation. They formulate the task as a reinforcement learning problem, as the goal of the generator which generates utterances, is to fool the discriminator into believing it is a real one. To do

[1] Github code repo: https://github.com/boya-song/models/tree/master/research/maskgan

this, they treat the output of the discriminator as a reward to the generator, making it more probable to generate more human-like utterances.

Using efficient gradient approximators to replace the non-differentiable sampling techniques (Jang et al., 2017 [10]) has not shown encouraging results, while unbiased and low variance gradient estimation proposed by Tucker et al. [19] could have some promising result.

WGAN-GP (Gulrajani et al., 2017 [8]) generates text in a one-shot manner to avoid dealing with backpropagation of discrete tokens. Rajeswar et al. (2017)[15] proposes a method in which the discriminator directly operates on the output of the generator.

RNNs often suffers from the problem of overfitting, with dropout proved to fail when applied to recurrent layers. Gal and Ghahramani [5] drop the same neural network units at each time step for both inputs, outputs and recurrent layers.

Reinforcement learning methods have been proved successful in natural language processing. MIXER (Ranzato et al., 2015 [16])uses a hybrid method of REINFORCE and cross entropy successfully optimized BLEU score directly. Bahdanau [1] employed an actor-critic methods where the rewards are task-specific scores like BLEU, rather than the rewards provided by the discriminator.

This work employs the reinforcement learning with an actor-critic methods to train the generator, while using the pre-train methods same with SeqGAN [20]. Also, during training, we use the variational recurrent dropout[5] to train both the generator and discriminator.

3. Model: MaskGAN

Our text generation problem can be defined as follows. Given a corpus S, we denote a real sequence sampled from S as $\boldsymbol{x} = \{x_1, x_2, \cdots, x_T\}$ with length T. We use $\boldsymbol{y} = \{y_1, y_2, \cdots, y_T\}$ to represent fake sequence generated from the generator. The input to discriminator, which is either real sequence $\boldsymbol{x}$ or fake sequence $\boldsymbol{y}$, is denoted by $\boldsymbol{z} = \{z_1, z_2, \cdots, z_T\}$. We also use $\boldsymbol{m} = \{m_1, m_2, \cdots, m_T\}$ to denote the mask vector. Note that for each $m_t \in \boldsymbol{m}$, it is either 1 or 0, meaning either the token is given or not. We use GAN to generate text sequence with good quality, which is demonstrated next.

3.1. The Generator

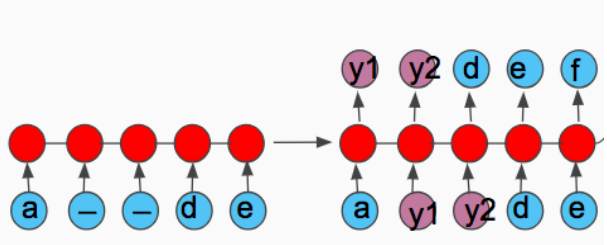

Figure 1. seq2seq generator architecture. The real sequence consists of a, b, c, d, e and b, c are substituted by blanks. Then in the decoder, the network generates new tokens to fill in the blanks, i.e., y_1 and y_2.

The generator is modeled via a seq2seq[18] network, which consists of an encoder and a decoder, as shown in Fig. 3.1. The encoder compresses the information of the input into a context vector and feed into decoder. Both encoder and decoder are long short-term networks (LSTM). The attention mechanism suggested by [13] is used, which can be substituted by any other attention mechanisms, e.g., [2]. The mask is generated deterministically or stochastically. Then the input to the encoder can be calculated as $\boldsymbol{m}(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{m}$, which means, at each time t, x_t is replaced by blank if m_t is 0. Then the decoder generates a new sequence by filling the blanks. The generator samples next token from the probability distribution

$$G_\theta(y_t) = P(y_t | x_1, \cdots, x_{t-1}, \boldsymbol{m}(\boldsymbol{x})) \tag{1}$$

3.2. The Discriminator

The discriminator is modeled via a similar seq2seq network. Given sequence $\boldsymbol{z}$ and original mask $\boldsymbol{m}$, the discriminator assigns a probability to each token, to denote whether it is real, i.e.,

$$D_\phi(z_t) = P(z_t \text{is real} | z_{0:T}, \boldsymbol{m}(\boldsymbol{x})). \tag{2}$$

Notice that the discriminator not just assigns a probability to the entire sequence. Given the probability for each token, we obtain more information about the sequence.

3.3. Actor-Critic

Since a sequence is made of discrete tokens, GAN[6] has difficulty to propagate the gradients from the discriminator to the generator, which makes it hard to train. Reinforcement learning is used to solve this issue. Instead of directly trying to minimize the loss, the generator use reinforcement learning to optimize its parameters. For the generator, current state is the generated sequence so far and the action is the token to generate next. And the reward is defined as the logarithm of the discriminator

$$r_t = \log D_\phi(y_t). \tag{3}$$

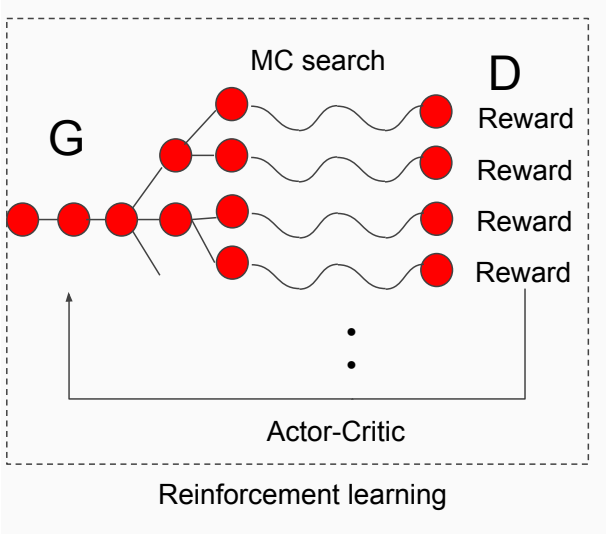

Figure 2. Actor-Critic architecture. The generator uses Monte Carlo (MC) search to estimate the rewards given by discriminator and then update its parameters.

We use Advantage Actor-Critic algorithm to update the generator, as shown in Fig. 3.3. The actor decides the policy, i.e., which token to generate, and the critic defined by the Q-function, tells the actor how good this token was and how it should adjust:

$$R(t) = \sum_{s=t}^{T} \gamma^{s-t} r_s, \tag{4}$$

where γ is the discount factor. To reduce the variance, we choose the state value function as our baseline and thus the advantage function is defined as:

$$A_t = R_t - V^G(x_{1:t}) \tag{5}$$

3.4. Optimization

The generator aims to maximize the cumulative reward $R = \sum_{t=1}^{T} r_t$. For a token y_t, the gradient is

$$\nabla_\theta \mathbb{E}_G[R_t] = (R_t - V^G(x_{1:t})) \nabla_\theta \log G_\theta(y_t).$$

Thus, for the full sequence, the gradient of the generator is

$$\nabla_\theta \mathbb{E}_G[R] = \mathbb{E}_{\boldsymbol{y}}(\sum_{t=1}^{T} R_t - V^G(x_{1:t})) \nabla_\theta \log G_\theta(y_t). \tag{6}$$

For the discriminator, given n samples, it tries to maximize the standard objective function

$$\min_\theta \max_\phi [\mathbb{E}_{\boldsymbol{x}} \log D_\phi(\boldsymbol{x}) + \mathbb{E}_{\boldsymbol{y}} \log(1 - D_\phi(G_\theta(\boldsymbol{x})))], \tag{7}$$

whose gradient is given by

$$\nabla_\phi \frac{1}{n} \sum_{i=1}^{n} \log D_\phi(\boldsymbol{x^{(i)}}) + \log(1 - D_\phi(\boldsymbol{y^{(i)}})). \tag{8}$$

3.5. MaskGAN Algorithm

Similar to [20], the whole MaskGAN algorithm is as shown in 1. We first pretrain the generator and discriminator by minimizing cross entropy loss (i.e., maximum likelihood estimation). In this way, GAN's training will start in a good state and will be speed up. Then we train the generator and discriminator iteratively. The generator is trained via Advantage Actor-Critic algorithm mentioned above and the discriminator is trained to maximize Eq. (7).

Algorithm 1 MaskGAN

```
Initialize G_θ, D_φ randomly.
Pretrain G_θ, D_φ using MLE.
repeat
    for i=0 to g do
        for t=1 to T do
            Generate a token y_t
            Compute the critic by Eq. (4)
        end for
        Update θ according to Eq. (6)
    end for
    for i=0 to d do
        Sample n fake samples {y^(1), ··· , y^(n)} from the generator G.
        Sample n real samples {x^(1), ··· , x^(n)} from corpus S.
        Feed discriminator with the real samples and fake samples. Update φ according to Eq. (8)
    end for
until MaskGAN converges
```

4. PadGAN

The original MaskGAN in the paper[4] uses a fixed number of tokens per sequence. However, the number of the tokens in a sentence has a great variety so that a sequence always consists of multiple pieces of different sentences. These sentences are not necessarily relate to each other and gives the network weird history information. This makes the generator hard to generate sentences of good quality.

To solve this problem, we preprocess the training data in the corpus in a different way. We fit one sentence into one sequence. Since we still need to have a fixed number of tokens per sequence, we pad the sentence using a special symbol if the number of the tokens it contains are less than the fixed length. We truncate the sentence if the number of tokens it contains are more than the fixed length.

MaskGAN	greatest play even in europe <eos> congress N million of silver N N <eos> put these copies of <unk>
PadGAN	consumers may want to move their telephones a little closer to the tv set <eos>

Table 1. Examples of training sequences from MaskGAN and PadGAN.

We call the model adopting this new preprocessing strategy PadGAN. Table 4 shows the difference between MaskGAN and PadGAN. The end of sentence is denoted by <eos>. Notice that the training sequence from MaskGAN consists of three sentence pieces: the end of the previous sentence, another whole sentence, and the beginning of the third sentence, while in PadGAN, a training sequence consists of a whole sentence.

5. Experiments

We use the The Penn Treebank (PTB)[14] as our corpus S. It contains a vocabulary of 10000 words. There are 930000 words in the training set, 74000 words in the validation set and 82000 words in the test set. For our experiments, we train on the training partition. 'n't' and "s' are treated as a word and 'N' stands for an arbitrary number. Other words outside of the vocabulary are marked as <unk>.

Following the steps described in the paper[6], we first pretrain a MLE model using the parameter of Gal & Ghahramani(2016)[5] until we get a perplexity about 78 for MaskMLE and PadMLE. Then, we train the MaskGAN model and PadGAN model with a mask rate of 0.5 until we get a perplexity of 55. However the second run start at perplexity of 101 and converged at around 90. So we generated the sentences with the converged model and below are the results.

The following conditional sample shown in Table 5 are generated with a mask rate of 0.5. A continuous mask is generated and the part of sentence with the mask is blanked out. The part of sentences without the underline is from the original sample and the part with underline is generated by the generator.

By masking all tokens in the input, we generate unconditional samples as a standard language model, shown in Table 5.

6. Evaluation

Our goal is to improve the performance of language model. Thus, in this section, we focus on the evaluation on the unconditional samples, which is equivalent to language model. The evaluation can be regarded as the hardest part in this project, since perplexity which is the standard evaluation measure for language model is not very suitable for the reason we elaborated later. Thus, we have to resort

Original	los angeles financier marvin davis who put united in play with a $ N billion bid two months ago last night <unk> both a ray of hope and an extra element of uncertainty by saying he remains interested in acquiring ual
MaskMLE	los angeles financier marvin davis who put united in play with a $ N billion offer for the year and gold kept his comfort said he was pacific very extra element of uncertainty by ignore the number of programs in <unk>
MaskMLE	los angeles financier marvin davis who put united in play with a $ N billion bid for the ashland group and an operating loss of $ N million or extra element of uncertainty by some analysts
PadMLE	los angeles financier marvin davis who worked discussing himself for technology recently switched investors in retailing as a form with night <unk> both a ray ...
PadGAN	los angeles financier marvin davis who wrote the movie to turning a <unk> depend on a stake in the two night <unk> both a ray ...

Table 2. Conditional samples from PTB for four models

Original	london has concluded that <unk> president <unk> was n't responsible for the execution of six british <unk> in world war ii although he probably was aware of the <unk>
MaskMLE	london and newport officials ruled that fees was completed by two-thirds must suffer
MaskGAN	london 's <unk> <unk> <unk> <unk> computer co. a washington haven in pittsburghhas agreed to <unk> a <unk> trip between the united states
PadMLE	london had gone part by buying by ford 's directors plus a most difficult option that is still probably involved in four games
PadGAN	london officials on hollywood dealings and issue june N

Table 3. Unconditional samples from PTB for four models

to other measures for evaluation[2].

6.1. Perplexity

Perplexity is the most widely used intrinsic evaluation measure for language model. It is developed from the field of information theory[17] and can be defined as

$$PP(W) = P(w_1 w_2 \cdots w_N)^{-\frac{1}{N}}, \tag{9}$$

where W stands for a sentence made of words $w_1, w_2, \cdots, w_N$. The intuition of perplexity is that a good language model should assign a sentence W in test set with high probability. Thus, the language model is better if the perplexity is lower. The results over perplexity is shown in Table 6.1. As we can see, the state-of-models of

model	perplexity
MaskMLE	101.309
MaskGAN	133.302
PadMLE	25.612*
PadGAN	63.485*

Table 4. Perplexity Result

GANs does not achieve good perplexity as models trained via maximizing likelihood estimation (MLE). Perplexity is a bad approximation unless the validation corpus is very similar to training corpus, where the sentences and n-grams are very similar. However, GAN models intends to capture the variety of samples the model can produce. Thus, it is reasonable to see GAN models does not improve perplexity over MLEs.

6.2. Mode Collapse

As mentioned in Section 1, GANs always suffer mode collapse and reduces the diversity of the language. We investigate the performance of mode collapse in this section.

Mode collapse are measured by calculating certain n-gram statistics directly in text generation task. To be more specific, we measure it by calculating the percentage of unique n-grams which are not occurred in the validation corpus, shown in Table 6.2. If the percentage is high, the model generates a more variety of sentences. Notice that

model	2-grams	3-grams	4-grams
MaskMLE	0.524	0.802	0.928
MaskGAN	0.422	0.768	0.909
PadMLE	0.701	0.870	0.952
PadGAN	0.546	0.834	0.943

Table 5. N-gram statistics

[2]Notice that in the original paper[4], the evaluation are mostly done over another corpus thus we could not do a direct comparison with their evaluation results. However, our evaluation results demonstrate similar patterns

mode collapse of GANs are supported by the decreased percentage from MLE models to GANs. But we can also see that PadGAN shows improvement over MaskGAN. It consistently generates more 2-grams, 3-grams, and 4-grams. Though mode collapse happens, the samples produced by all the models are unique.

6.3. Human Evaluation

As mentioned in section 6.1, perplexity may not be a good evaluation measure for GANs. To better understand the models and evaluate the quality of generated texts, similar to Fedus et. al [4], we also performed a human evaluation. The evaluation procedure can be described as follows. We randomly selected 25 generated sentences from each model, i.e., MaskMLE, MaskGAN, PadMLE, and PadGAN, which makes it total 100 sentences. We also randomly permuted 100 sentences such that there is no way to know which sentence comes from which model. We asked three educated people to score all of the sentences. Each sentence is scored from 1 to 5 for grammar, topic, and overall quality, where 1 stands for poor and 5 stands for great. Then for each sentence, we calculate the average score over three people's scores to reduce bias. And finally for each model, we calculate the average scores for all sentences from that model.

rating	syntax	context	overall
MaskMLE	3.04	2.95	2.88
MaskGAN	3.76	3.67	3.53
PadMLE	3.23	3.17	3.03
PadGAN	3.89	3.64	3.56

Table 6. Blind heads-up human evaluation between four models trained on PTB. 25 new snippets (each 30 words long) from each model are unconditionally sampled and randomized. Three ratings were obtained and averaged for each model.

As shown in Table 6.3, GANs obtains much higher scores than the corresponding MLE model, which shows GANs can generate texts with higher quality. Also, notice that PadMLE achieves higher scores than MaskMLE over syntax, context and overall, while PadGAN achives higher scores than MaskGAN on syntax and overall. Though PadGAN does not improve the score over context, it still can be regarded as having a comparable performance. Since we only evaluated against 25 sentences per model, there may still exist some biases. But in general, we can conclude that GANs has a better performance than MLEs while Pad models are better than Mask Models.

7. Conclusion and Future Work

In this work, we aim to generate texts with good quality. First, we successfully reproduced MaskGAN[4], proposed by Fedus. et al, proved MaskGAN does show advantage in

generating human-readable text, and the proposed contiguous in-filling task has successfully reduced mode collapse and stabilized the training stability. To further improve the performance of MaskGAN, we proposed padGAN, using a different preprocessing pipeline, which fit one sentence into one sequence by padding the sentence using a special symbol. We conducted experiments by using The Penn Treebank (PTB) corpus and evaluated the result of MaskGAN and PadGAN using perplexity, mode collapse, and human evaluation. The evaluation results showed PadGAN improves the overall quality of generated text.

MaskGAN shows good performance in language model. In the future, we want to try this model in other Natural language process (NLP) fields as well, which utilizes language model, e.g., question and answering, intelligent customer service. We can also apply this method in other tasks like image captioning. Given a question, the model aims to generate a good answer. The setting of the model is actually very similar to the setting in MaskGAN and we think MaskGAN can generate good answers.

References

[1] D. Bahdanau, P. Brakel, K. Xu, A. Goyal, R. Lowe, J. Pineau, A. Courville, and Y. Bengio. An Actor-Critic Algorithm for Sequence Prediction. (2015):1–17, 2016.

[2] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014.

[3] S. Bengio, O. Vinyals, N. Jaitly, and N. Shazeer. Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks. pages 1–9, 2015.

[4] W. Fedus, I. Goodfellow, and A. M. Dai. MaskGAN: Better text generation via filling in the _. In *International Conference on Learning Representations*, 2018.

[5] Y. Gal and Z. Ghahramani. A theoretically grounded application of dropout in recurrent neural networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 1019–1027. Curran Associates, Inc., 2016.

[6] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

[7] A. Graves. Supervised Sequence Labeling with Recurrent Neural Networks. *arXiv preprint arXiv:1308.0850*, 12(1):126–140, 2013.

[8] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville. Improved training of wasserstein gans. In *Advances in Neural Information Processing Systems*, pages 5769–5779, 2017.

[9] S. Hochreiter and J. Urgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997.

[10] E. Jang, S. Gu, and B. Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[11] A. Lamb, A. Goyal, Y. Zhang, S. Zhang, A. Courville, and Y. Bengio. Professor Forcing: A New Algorithm for Training Recurrent Networks. (Nips):1–9, 2016.

[12] J. Li, W. Monroe, T. Shi, A. Ritter, and D. Jurafsky. Adversarial learning for neural dialogue generation. *CoRR*, abs/1701.06547, 2017.

[13] M. Luong, H. Pham, and C. D. Manning. Effective approaches to attention-based neural machine translation. *CoRR*, abs/1508.04025, 2015.

[14] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini. Building a large annotated corpus of english: the penn treebank. *Comput. Linguist.*, 19(2):313–330, June 1993.

[15] S. Rajeswar, S. Subramanian, F. Dutil, C. Pal, and A. Courville. Adversarial generation of natural language. *arXiv preprint arXiv:1705.10929*, 2017.

[16] M. Ranzato, S. Chopra, M. Auli, and W. Zaremba. Sequence level training with recurrent neural networks. *arXiv preprint arXiv:1511.06732*, 2015.

[17] C. Shannon. A mathematical theory of communication. *The Bell System Technical Journal 1948*, pages 379–423, 623–656, 2017.

[18] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215, 2014.

[19] G. Tucker, A. Mnih, C. J. Maddison, J. Lawson, and J. Sohl-Dickstein. Rebar: Low-variance, unbiased gradient estimates for discrete latent variable models. In *Advances in Neural Information Processing Systems*, pages 2624–2633, 2017.

[20] L. Yu, W. Zhang, J. Wang, and Y. Yu. Seqgan: Sequence generative adversarial nets with policy gradient. *CoRR*, abs/1609.05473, 2016.