The State of GANs for Natural Language Generation

Markus Roth
Swiss Joint Master of Science in Computer Science
University of Berne, Switzerland
Email: markus.roth1@students.unibe.ch

***Abstract*—Generative adversarial networks (GANs) - consisting of a generative and a discriminative network trained competitively - have been tremendously successful in the domain of computer vision. The generative network tries to generate synthetic data samples that are indistinguishable from a given set of real data samples. The discriminative network tries to correctly classify real data samples and synthetic data samples. This has lead to the unsupervised generation of synthetic images that are difficult to tell apart from photographs even for a human observer.**

GANs work by back-propagating the gradients from the discriminator to the generator in a gradient descent optimization process. After each synthetic sample is generated, the discriminator gives it a classification score. It then tells the generator the gradients of its weights respective to the target of classifying it as as real. The generator then updates its weights accordingly, leading to a slightly more convincing sample generated next time.

There is a large research interest in applying GANs to generating natural language. Two fundamental problems with this domain remain unsolved: Firstly, the choice of a word or letter by the generator is discrete, so back-propagation from the discriminator to the generator becomes impossible. Secondly, the space of natural language is sparse, so gradual improvement via gradient descent will not work.

This paper gives an overview of the current research trends, approaches, metrics and successes of trying to solve or circumvent these problems. It also includes the description of a prototype by the author that tries a novel approach in solving them.

I. Introduction

Conversational agents like Chatbots, voice-controlled smart home devices and mobile phones require the generation of convincing utterances of natural language. To create the feeling of real conversation, these generated utterances should have a high variation and mimic human-generated utterances as well as possible. Currently, this is an unsolved problem.

In the field of computer vision, the GAN approach to a similar problem has experienced very fast-paced success. See figure 1 for an example. The research community is eager to apply this success to the domain of natural language generation in the hopes of achieving similar success.

While this success has yet to be achieved, quite a few interesting approaches are being tried. This is in part possible due to rapid advancement in tooling and hardware as well as the availability of large data sets, making it fairly easy to experiment with GANs. Additionally, since the invention of GANs in 2014, numerous mathematical improvements on them have been proposed.

Structure: This paper will first detail the general structure and core mathematics behind a typical GAN in section II, then describe the problems of applying them to natural language generation in section III. Four approaches will then be described in detail to overcome these problems: SeqGAN in section VI, Gumbel-softmax in section VII, BGAN in section VIII and MaliGAN in section IX. My own prototype using word embeddings is described in section X, after which future work in section XI and conclusions in section XII conclude the paper.

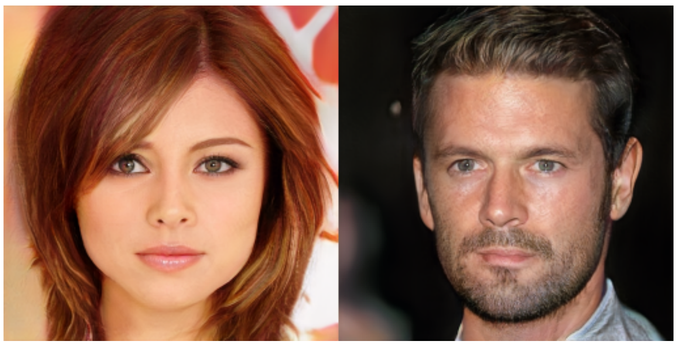


Fig. 1. Images generated by a GAN. The CelebA data set was used as a ground truth. For a human observer, it is not immediately obvious that these images are synthetic. Image taken from [35].

II. GENERATIVE ADVERSARIAL NETWORKS

Say we have some natural phenomenon that produces data samples $\boldsymbol{x}_1, ..., \boldsymbol{x}_n$ according to some true probability distribution $p_{data}(x)$. The goal of a generative model can be defined as learning a probability distribution $\hat{p}(x)$ so that the Kullback-Leibler divergence $D_{KL}(\hat{p}||p)$ between the learned probability distribution $\hat{p}$ and the true probability distribution p is minimal [1].

Formally, a generative model can be defined as a generator function G with parameters $\theta^{(G)}$ that can generate a synthetic data sample x by mapping a latent input variable $\boldsymbol{z}$ to the data sample space:

$$\boldsymbol{x} = G(\boldsymbol{z}; \theta^{(G)}) \tag{1}$$

Since p_{data} is unknown for real-world phenomenon, $D_{KL}(\hat{p}||p)$ cannot be empirically measured. To measure the quality G we thus need some way of differentiating between real and synthetic data samples.

The idea of a generative adversarial network (GAN) is to train a second, discriminative model that learns to classify data samples accordingly, and in turn to use this classification as a feedback to improve the generator function G.

The discriminator can be described as a scalar function D that takes a data sample x and a parameter $\theta^{(D)}$ and outputs the probability q that x is a real data sample rather than a synthetic data sample [2]:

$$q = D(\boldsymbol{x}; \theta^{(D)}) \tag{2}$$

D and G are trained in parallel. D is trained to maximize the probability of categorizing a given data sample x correctly, and G is trained to maximize $log(1 - D(G(x)))$, the chance of D classifying the output of G as real data. This leads to G and D competing in a minimax game with value function V(G, D):

$$\begin{aligned}\min_G \max_D V(D,G) = \\ &\mathbb{E}_{\boldsymbol{x}\sim p_{data}}(x)[log(D(\boldsymbol{x}))] \\ &+ \mathbb{E}_{\boldsymbol{z}\sim p_z}(z)[log(1 - D(G(\boldsymbol{z})))]\end{aligned} \tag{3}$$

Applying stochastic gradient descent in an alternating fashion to G and D to optimize $\theta^{(G)}$ and $\theta^{(D)}$ converges on the global optimum of $\hat{p} = p$ if $\max_D V(G,D)$ is convex in $\theta^{(G)}$. In the common case of using neural networks for G and D $\max_D V(G,D)$ is not convex, and thus gradient descent is not guaranteed to reach an equilibrium, but works well in practice [3]. An architecture overview of a GAN can be seen in figure 2.

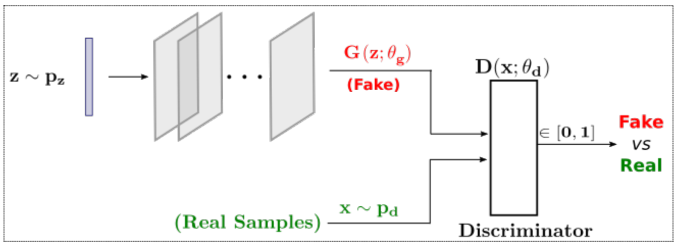


Fig. 2. Architecture of a generative adversarial network (GAN). A generator G samples from a latent distribution p_z to generate a fake sample p. The discriminator D takes fake and real samples and outputs the probability of the sample being real. G and D are trained together. Image taken from https://arnabgho.github.io/MADGAN.

III. SEQUENCES IN SPARSE SPACES

While GANs work very well for continuous, dense data sample spaces like images, there are two main reasons they are not immediately applicable to generating text: Differentiability and sparsity of the data sample space.

Sparsity is a problem for sequences of alphanumeric characters or word vectors representing natural language because slightly improving a sequence to be more likely to fool the discriminator is not meaningful. Changing a single letter in a word or word in a sentence to its somehow defined neighbor does not result in a high probability of generating a new sequence that is both syntactically and semantically correct as well as actually better at deceiving the discriminator. Moving from one valid sequence to another valid one with a higher chance of fooling the discriminator will often require a large change in parameters that cannot be achieved by gradient descent [2]. Recently, word embeddings [5] have become a popular way of addressing this issue.

Differentiability is a more fundamental problem. During training of a minibatch of m values sampled from the latent variable $\boldsymbol{z}$, $\theta^{(G)}$ gets updated with the gradient

$$\nabla_{\theta^{(G)}} \frac{1}{m} \sum_{i=1}^{m} log(1 - D(G(\boldsymbol{z}^{(i)}))). \tag{4}$$

Using this gradient to improve $\theta^{(G)}$ requires the composition of D and G to be fully differentiable. In the case of G generating a sequence of discrete values, this is not the case. In the simplest approach, after a fully differentiable network with continuous outputs a step function is appended to the architecture to select one of the candidate tokens to add to the output sequence. The gradient of such a step function will be 0 almost everywhere due to its discrete nature [4]. Without a differentiable function that is able to generate discrete tokens, GANs remain non-applicable to sequence generation.

IV. GANS GENERATING SEQUENCES

Various authors have tried to find ways to find differentiable functions for outputting discrete sequences so that GANs can be used for sequence generation. In the following sections a selection of these approaches are presented, selected by their applicability to be used in GANs to generate language.

The field of adversarial training applied to sequence generation is still young and immature. Many cited papers are only available as arXiv pre-prints and have yet to be peer-reviewed. None of the approaches analyzed in this paper are able to generate text that is indistinguishable from natural language even at a glance.

V. EVALUATION

Evaluating the quality of generated natural language is not trivial. Many papers use BLEU scores [11], a measure originally conceived to measure the quality of machine translations. It is comparatively simple: It compares the distribution on n-grams of tokens in generated text with their distribution in real text. It should be noted that BLEU does not include any way of measuring semantic content. Even with a perfect BLEU score, generated text will not necessarily make any sense to a reader.

VI. APPROACH I: SEQGAN

A. *Context*

The paper ”SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient” [7] was the first formalized approach at applying GANs to text. First pre-published to arXiv in September 2016, it was later accepted to AAAI 2017. The experiments were done in TensorFlow [6] with fully published source code[1].

[1]https://github.com/LantaoYu/SeqGAN

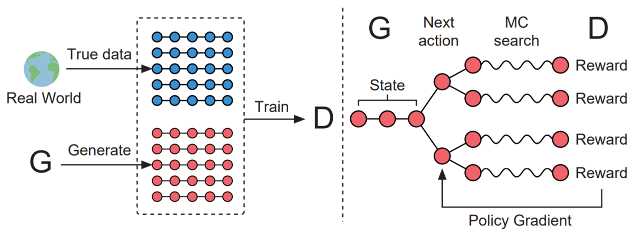


Fig. 3. Architecture of the SeqGAN model. The generator G decides on the next token of a partially completed sequence by sampling from a policy and using Monte Carlo search to update the policy gradient via discriminator response on the fully completed sequence. Image taken from [7].

B. *Contribution*

SeqGAN contributes two ideas: Solving the differentiability problem by using the REINFORCE algorithm, and updating intermediate state-action policies from the discriminator result on completed sequences by using a Monte Carlo search.

The paper restates a GAN as a reinforcement learning problem, where the generator G is a state-action policy $G_\theta(y_t|Y_{1:t-1})$ that generates the current token y_t based on a partially generated sequence $Y_{1:t-1}$. The objective of the generator is, starting from a start state s_0, to maximize its expected end reward

$$\begin{aligned} J(\theta) &= \mathbb{E}[R_T|s_0,\theta] \\ &= \sum_{y_1\in\mathcal{Y}} G_\theta(y_1|s_0) * Q_{D^{\theta(D)}}^{G^{\theta(G)}}(s_0,y_1) \end{aligned} \tag{5}$$

where R_T is the expected end reward for the completed sequence $Y_{1:T}$ and $\mathcal{Y}$ is the alphabet of all possible tokens. $Q_D^G(s,a)$ is the expected accumulative reward from discriminator D when starting from state s, taking action a and then continuing with policy G. An overview of the architecture can be seen in figure 3.

Intuitively, the generator in a SeqGAN can be seen as a probabilistic function that will give a probability distribution for every possible next token given a partially completed sequence. Since the resulting output is no longer a discrete token but a probability distribution, this function is fully differentiable and can thus be trained by the gradient output from the generator. This is the core idea behind the REINFORCE algorithm [8].

Thus the SeqGAN paper proposes approximating the cumulative award of the complete sequence $Q_D^G(s,a)$ using a N-time Monte Carlo search sampling over the unknown next $T-t$ tokens using the GAN generator $G^{\theta^{(G)}}$:

$$Y_{1:T}^1, ..., Y_{1:T}^N = MC^{G^{\theta(G)}}(T_{1:t};N) \tag{6}$$

In a N-time Monte Carlo search, the sequence is completed probabilistically N times according to the current generator and the resulting sequence is then fed into the discriminator to obtain the reward. The expected end reward for the completed sequence R_T then becomes

$$Q_{D^{\theta(D)}}^{G^{\theta(G)}}(Y_{1:t-1},y_t) = \frac{1}{N}\sum_{n=1}^{N} D^{\theta(D)} Y_{1:T}^n \tag{7}$$

where

$$Y_{1:T}^n \in MC^{G^{\theta(G)}}(Y_{1:t};N) \tag{8}$$

for all steps until the last step where $t=T$ and thus

$$Q_{D^{\theta(D)}}^{G^{\theta(G)}}(Y_{1:T-1},y_T) = D^{\theta(D)}(Y_{1:T}). \tag{9}$$

In summary, SeqGAN overcomes the gradient problem by updating the probabilistic policy directly using the policy gradient coming from the discriminator, and uses Monte Carlo unrolling to consider estimated rewards for multiple possible completed sequences at any particular generation step.

C. *Architecture*

SeqGAN uses a recurrent neural network, specifically a long short-term memory network (LSTM) [9] as a sequence and a convolutional neural network (CNN) [10] as a sequence discriminator. There is no in-depth discussion on network architecture details in the paper.

D. *Scenarios*

An interesting part of the SeqGAN paper is a novel evaluation metric. To measure the quality of the learned generator, they use randomly initialized target LSTM called the oracle that generates the real samples. Their generator then learns to be as close as possible to this oracle LSTM, and their differences can be precisely measured by comparing their probability distributions for generated sequences.

Additionally, the SeqGAN paper uses three real-world scenarios: Generating speeches of Barack Obama, Chinese Poems and folk tunes. For all three scenarios, evaluation uses the BLEU metric.

E. *Results*

SeqGAN is compared to four baseline methods: Random token generation, the generator LSTM trained in a non-adversarial fashion with maximum likelihood estimation, scheduled sampling [12] and using PG-BLEU rather than a trained discriminator in Monte Carlo search. They conclude that SeqGAN performs significantly better than the other methods.

The SeqGAN paper does not show any generated example sentences.

VII. Approach II: Gumbel-Softmax

A. *Context*

The paper ”GANs for Sequences of Discrete Elements with the Gumbel-softmax Distribution” [13] was uploaded to arXiv in November 2016. It was accepted as a workshop paper to NIPS 2016. No source code was published, and no information about the technology or frameworks used was included in the paper.

The paper presenting the underlying idea of the Gumbel-softmax estimator [14] was accepted as a poster to ICLR 2017.

B. Contribution

The GumbelGAN paper [13] is a pretty straightforward application of [14] to the area of GAN: Use of the smooth Gumbel-softmax estimator as the last layer in the generator network for token sampling as to enable back-propagation from the discriminator to the generator.

A natural way to model the last layer of a generative network is to have an output vector of the same size as the alphabet of possible tokens $\mathcal{Y}$, which activations can be interpreted as unnormalized probabilities of that token being generated next. This output vector is often normalized with the softmax function

$$[\text{softmax}(h)]_i = \frac{exp(h_i)}{\sum_{j=1}^{K} exp(h_j)}, \text{for i = 1,...,d} \tag{10}$$

such that the output of the softmax can be treated as a probability distribution over the alphabet. This categorical distribution can be sampled to get the next token in the output sequence. This sampling operation is not differentiable and cannot be passed when back-propagating the gradient from the discriminator.

The Gumbel-max trick [15] is that sampling from a categorical distribution with class probabilities π has an equal distribution as

$$z = \text{one hot}\left(\arg\max_i [g_i + \log \pi_i]\right) \tag{11}$$

where $g_i, ..., g_k$ are independent samples drawn from Gumbel(0, 1), which in turn can be sampled by drawing u $\sim$ Uniform(0, 1) and computing $g = log(log(u))$.

While Gumbel-max sampling is still not differentiable due to the hard choice in argmax, it has been proposed [14] to replace the hard argmax with a continuous and differentiable softmax to achieve tractability. Hereby we get the Gumbel-softmax operator to generate a k-dimensional sample vector ***y*** from a probability distribution π

$$y_i = \frac{\exp((\log(\pi_i) + g_i)/\tau}{\sum_{j=1}^{k} \exp((\log(\pi_j) + g_j)/\tau} \text{for i = 1,...,k.} \tag{12}$$

As the parameter τ approaches zero, Gumbel-softmax approaches a one-hot encoding. As τ increases, Gumbel-softmax approaches a fully random sampling where the probability of all tokes is equal (see figure 4). For this reason, τ is called the temperature of the distribution.

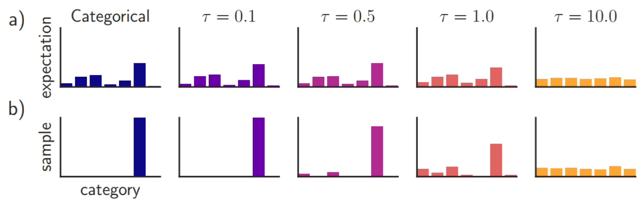


Fig. 4. Visualization of the effect of temperature on the Gumbel-softmax distribution. At low temperature, Gumbel-softmax is close to one-hot (accurate but high gradient variance), at high temperatures it becomes a random choice (low gradient variance but smooth). Image taken from [14].

The Gumbel-softmax distribution is smooth and has well-defined gradients for all values, so it can be used as a last layer for a generator network and be back-propagated through.

The main advantage of the Gumbel-softmax distribution is temperature annealing. Learning can be started with a high temperature, where samples are smooth but the variance in gradients is small, and then annealed over time to a temperature approaching zero, where samples are close to one-hot but gradients have a high variance [14].

In summary, the Gumbel-softmax estimator enables back-propagation from the discriminator to the generator by replacing the sampling hard choice with a differentiable operation. This Gumbel-softmax operation interpolates between a one-hot encoding and a fully random choice by use of a temperature variable, which in turn can be annealed in training to achieve stability.

C. Architecture

The GumbelGAN paper [13] uses LSTM recurrent networks [9] as both generator and discriminator. Both networks are optimized using ADAM [16] with a fixed learning rate of $1e-3$ and a minibatch-size of 200. Temperature was linearly annealed from $\tau = 5$ at the beginning of training to $\tau = 1$ at epoch 10'000, then kept constant until the end of training.

It is interesting to note that during training, the real data samples were transformed from one-hot encodings to distributions with p = 0.9 at the actual value and equal probability otherwise. Then the Gumbel-softmax estimator was used to transform the real value into a probabilistic vector with a similar distribution as the synthetic data samples to use during training.

D. Scenarios

In the GumbelGAN paper, a single experiment is performed. They authors create a synthetic language using a context-free grammar, then train their generator LSTM go generate sequences that are as similar as possible to the sequences generated by the context-free grammar.

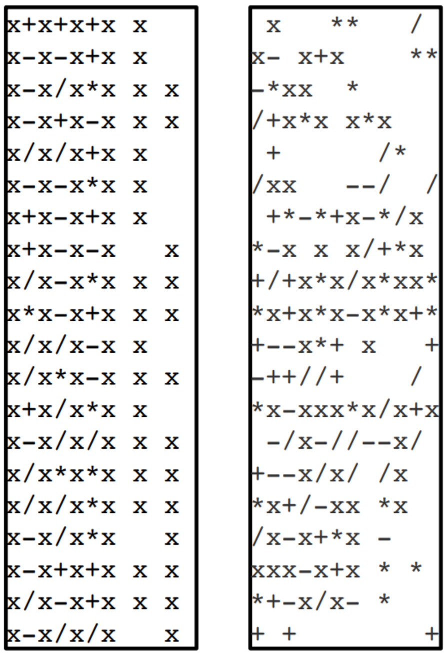
The language is very simple. It consists of the five characters "x", "+", "-", "*" and "/". These characters are combined by the context-free grammar for form simple mathematical operations with a single variable, for example "x/x+x-x" or "x/x+x-xx". Formally, the generation rules, separated by $||$, are as as follows:

$$S \to x||S + S||S - S||S \star S||S/S \tag{13}$$

Thus the learning target of the generator LSTM using Gumbel-softmax annealing is to learn the rules of the context-free grammar and generate sequences indistinguishable from the ones generated by the context-free grammar. No experiments with sequences more close to natural language are discussed.

E. Results

Even for the exceedingly simple approximation of natural language, the model does not learn the target context-free grammar. The authors admit that a recurrent neural network



```
x+x+x+x x
x-x-x+x x
x-x/x*x x x
x-x+x-x x x
x/x/x+x x
x-x-x*x x
x+x-x+x x
x+x-x-x   x
x/x-x*x x x
x*x-x+x x x
x/x/x-x x
x/x*x-x x x
x+x/x*x x
x-x/x/x x x
x/x*x*x x x
x/x/x*x x x
x-x/x*x   x
x-x+x+x x x
x/x-x+x x x
x-x/x/x   x
```

```
 x    **    /
x- x+x     **
-*xx   *
/+x*x x*x
 +       /*
/xx   --/  /
 +*-*+x-*/x
*-x x x/+*x
+/+x*x/x*xx*
*x+x*x-x*x+*
+--x*+ x    +
-++//+     /
*x-xxx*x/x+x
 -/x-//--x/
+--x/x/ /x
*x+/-xx *x
/x-x+*x -
xxx-x+x * *
*+-x/x- *
+ +         +
```

Fig. 5. A comparison between sampled sequences from an LSTM trained with maximum likelihood estimation (left) and the generator of the Humbel-softmax-GAN (right). Spaces are because input sequences were padded with spaces if shorter than 12 characters. Image taken from [14].

trained with maximum-likelihood estimation achieves a much better result for their scenario. Results can be seen in 5.

The paper does not include any comparison with other relevant generative methods such as SeqGAN [7] or scheduled sampling [12].

VIII. Approach III: BGAN

A. Context

The paper ”Boundary-Seeking Generative Adversarial Networks” [17] was uploaded to arXiv in February 2017. As of yet, it has not been accepted for publication. The experiments were done in Lasagne [18] built on Theano [19] with fully published source code[2].

B. Contribution

The idea behind the boundary-seeking GAN is a new loss function for the generator. Intuitively, rather than the generator trying to get the discriminator to assign the highest possible probability that a generated sample is real, it tries to generate samples where the discriminator is as uncertain as possible about whether the sample is real or generated. This is legitimized by arguing that for a perfect generator, the discriminator would be unable to discriminate between real and generated samples, so the ideal generated sample lies at the boundary between the discriminator classifying it as real or generated.

The idea behind BGAN is that a discriminator, which is a standard probabilistic binary classifier, is easier to train than a generator, which can be required to output complex data samples and has a continuously shifting objective function.

In addition to this new loss function, BGAN uses reinforcement learning in a similar fashion to SeqGAN. BGAN uses the discriminator output as importance weights, and can overcome the differentiability barrier of the discriminator output using importance sampling.

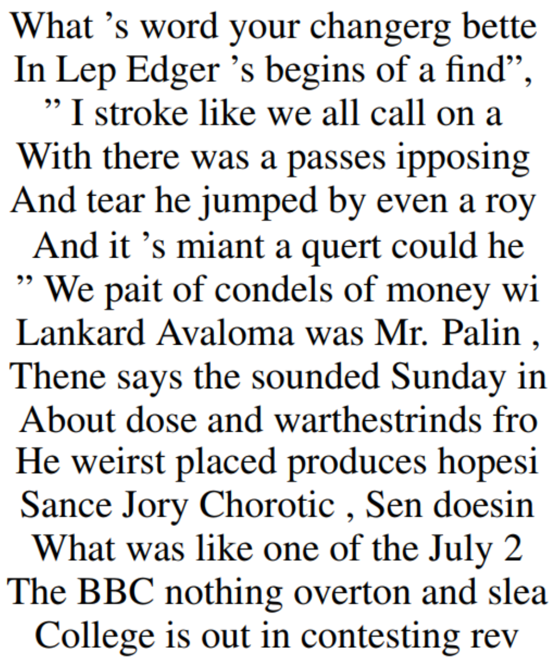
C. Architecture

Both generator and discriminator are deep convolutional networks following the improved WGAN paper [21]. Batch normalization [22] was used for the generator, but not the discriminator.

The model was optimized using ADAM [16] using exponential decay rates of $\beta_1 = 0.95$ and $\beta_2 = 0.5$. Three learning rates were compared: $1\mathrm{e}{-4}$, $1\mathrm{e}{-5}$, and $1\mathrm{e}{-6}$. Discriminator and Generator were trained in an alternating fashion with each network being trained the same amount per turn. M, the number of samples per generated sample used to get the normalized weights, was set to 20.

D. Scenarios

The authors use the billion-word dataset [20], while only sentences in the dataset of at least 32 characters were considered, and these sentences were truncated to 32 characters. Sequences of 32 characters were generated. Examples of generated sequences can be seen in 6.



What ’s word your changerg bette
In Lep Edger ’s begins of a find”,
” I stroke like we all call on a
With there was a passes ipposing
And tear he jumped by even a roy
And it ’s miant a quert could he
” We pait of condels of money wi
Lankard Avaloma was Mr. Palin ,
Thene says the sounded Sunday in
About dose and warthestrinds fro
He weirst placed produces hopesi
Sance Jory Chorotic , Sen doesin
What was like one of the July 2
The BBC nothing overton and slea
College is out in contesting rev

Fig. 6. Results of generating 32 character sequences using BGAN training from the 1 billion word dataset. Image taken from [17] and re-formatted.

E. Results

No quantitative evaluation of the generated sequences was attempted. The authors claim to have achieved the best qualitative results to date ”without any continuous relaxation”

[2]https://github.com/rdevon/BGAN

and being "trained from scratch without any pre-training and without any auxiliary supervised loss". They acknowledge that recurrent neural networks trained with maximum likelihood estimation achieve much better results.

It is interesting to note that the authors attempted to compare their BGAN with the Gumbel-softmax approach, but were unable to get Gumbel-softmax to converge in their scenario. Apart from that approach, no comparisons with other methods were attempted.

IX. Approach IV: MaliGAN

A. Context

The paper "Maximum-Likelihood Augmented Discrete Generative Adversarial Networks" [23] was uploaded to arXiv in February 2017. As of yet, it has not been accepted for publication. Source code for the experiments was not released, and no information about the used frameworks or programming languages is included in the paper.

B. Contribution

When training a generative model in a non-adversarial fashion, typically maximum likelihood estimation (MLE) is used as an objective function. MLE can be used to find the parameters $\theta^{(D)}$ of the generator that give the highest probability of the real data samples x to be generated. The fixed MLE objective function works well in training auto-regressive methods such as the standard LSTM [9], and leads to stable training.

The problem with MLE is that the model can only learn from real samples, it cannot learn from it's own generated output. This means that when a network generates a partial sequence during inference that does not exist in a real sample, it has no way of knowing how to continue the sequence.

GAN can solve this problem by getting feedback not from MLE, but from the discriminator D that can judge partial sequences that do not appear in a real data sample just as well as ones that do. However, there is a core problem in this approach: The training objective of the generator is the output of the discriminator, and the discriminator changes with the generator. This means the objective function of the generator is a moving target.

When a model is converging, this means that the objective function gets better with time and leads to much better results than a fixed objective function. In practice it has turned out that it is very hard to get GAN to converge in high-dimensional discrete data sample spaces such as natural language. If the generated sequence is far away from a real sequence, there is no good positive feedback, and the continuously shifting objective function for the generator does not help the generator network converge at all.

The MaliGAN paper proposed to ameliorate this situation by making the output of the discriminator network D closer to a MLE response, in essence by following the method proposed by Google Brain in [24]. It stabilizes training by introducing a delayed generator network G'. While the normal generator G is updated with the discriminator output continuously, G' is only trained irregularly. This delayed network acts as a stabilizer for the objective function, leading to more stable training.

Apart from this MLE-stabilized loss function, the network works similar to SeqGAN: It casts the problem of generating a discrete sequence as a reinforcement learning problem and uses the policy gradient method with the REINFORCE algorithm to update the parameters of the generator.

C. Architecture

For natural language generation, MaliGAN uses a single-layer gated recurrent unit (GRU) [26]. A as discriminator, a single-layer, bi-directional GRU is used. The generator is pre-trained on the real sentences using teacher-forcing.

D. Scenarios

The evaluation scenario closest to the natural language generation task uses the penn treebank dataset [25]. All sentences over 35 words are removed from the training set due to the large computational cost of Monte Carlo Tree search used in in MaliGAN. The best-performing model had 200 hidden neurons and 200 dimensions used for word embeddings.

They report the sentence-level perplexity, the averaged perplexity of all generated sentences in comparison to the real sentences from the dataset. They compare with a baseline model, a GRU with the same settings as their generator, trained with simple MLE rather than adversarially.

E. Results

TABLE I
MaliGAN performance on the Penn-Treebank dataset

	MLE	MaliGAN basic	MaliGAN full
Valid-perplexity	141.9	131.6	128.0
Test-perplexity	138.2	125.3	123.8

MaliGAN, as well as a less complex version trained without Monte-Carlo tree search called MaliGAN basic, were compared the the MLE-trained baseline model. The baseline model is identical to the generator except for the training objective.

Perplexity is a measure of comparing probability distributions, a lower perplexity value means the distributions are more similar. This means that even the simple MaliGAN basic model outperforms a simple recurrent approach trained with MLE, and a full model including Monte Carlo tree search as well as other variance-reducing measures again improves on this score. An overview of the results can be seen in table I.

The MaliGAN paper does not give examples of generated natural language sentences, nor does it attempt any comparison to other GAN-based text generation methods.

X. PROTOTYPE

A. Context

I was challenged to create a prototype of my own in the seminar "Chatbots and conversational agents". While there are some approaches at end to end deep learning systems that model complete conversations [27], and even some that use adversarial techniques [28] [29], I concluded that end to end text generation using generative adversarial networks is still in its infancy. Until the core problem of the differentiability of discrete sequence generation using GANs is solved, no practical efforts in actual dialogue generation are realistic. Thus, I decided to focus my effort on adversarial language generation in general, as a necessary first step in generating dialogue.

The project was implemented in PyTorch [32], I published the source code[3]. The source was heavily based on the PyTorch implementation of the advanced training for WGAN paper [21] by Marvin Cao[4] as well as the torchtext tutorial by Allen Nie[5].

B. Contribution

Typically, words are encoded as one-hot vectors across the entire language vocabulary. This makes the vectors discrete and the vector space extremely sparse - each dimension is only used by a single word. This is often called the ”curse of dimensionality”. Within a GAN, if a generator generates word sequences encoded in such a way, the choice is non-differentiable as discussed above. And even if it were differentiable, changing a single word vector slightly according to its gradient would not have any chance of making the generated sentence slightly more realistic.

An alternative way of encoding words are word embeddings (for example [31]), also known collectively under the term word2vec. Word embeddings map the one-hot encoded vectors to a new vector space with much fewer dimensions. In this embedding space, words that are semantically related have low distances to each other. One can then perform meaningful ”word arithmetic” within the embedded vector space, such as the famous ”king - man + woman = queen” or ”paris france + poland = warsaw” [32].

In my prototype, I map real word sequences into a word embedding space. The generator then generates synthetic sequences within the same space. The discriminator tries to discriminate between the two. Since the embedded vector space is comparatively low-dimensional and continuous, the differentiability problem does not arise.

Note that this also means that the discriminator does not operate on actual sentences. To generate synthetic output in the input space of natural language, the generator can generate synthetic sequences within the word embedding space. A nearest neighbor search within the embedding space is then used to generate sequences of tokens taken from the input data. It is important to state that this problem is only barely tractable, having the complexity of

$$O(d * N) \tag{14}$$

where d is the the dimensionality of the embedding space and N is the number of words in the vocabulary. Using such a sequence generation procedure during training would be prohibitively slow. However, since it is only necessary to generate sequences in the input space to evaluate performance periodically during training, this works well in a prototype. For a production system that needs to generate sequences as part of a productive workload, optimizations like a k-d tree might be used.

C. Architecture

GloVe [30] is an unsupervised learning method for mapping words into a much lower-dimensional vector space. I used the pre-trained GloVe embedding vectors ”glove.twitter.27B”, which is trained on 2 billions tweets consisting of 27 billion tokens. For performance reasons, I chose the embedding space with 25 dimensions, even though I expect higher dimensionality to improve results.

Both the generator and the discriminator are residual [32] convolutional networks using ReLU activations. They each consist of 10 residual layers each. Fully connected layers are used before and after the residual layers.

The model is trained as a Wasserstein GAN following the improved training practices described in [21]. The model was optimized using ADAM [16] using exponential decay rates of $\beta_1 = 0.5$ and $\beta_2 = 0.9$ with a learning rate of $1e - 4$, 10 discriminator iterations per generator iteration and a gradient penalty value λ of 10.

Training 200000 iterations took around 12 hours on a GeForce GTX 1080. Learning progress was visualized with TensorBoard[6].

D. Results

As a dataset, I chose a collection of customer support Tweets posted on Kaggle [7]. I filtered them to only use Tweets by AirAsia, ignoring conversational structure. This left me with 12829 real sequences. I made this choice because of the inherently related subject matter of all Tweets, as well as their short length. The tweets were tokenized using spaCy [8], stripped of most inter-punctuation and then clipped to a length of exactly 10 tokens.

TABLE II
EXAMPLE SEQUENCES GENERATED BY MY PROTOTYPE

sorry list us to to if there there will separate
we should hence have previous to not route when you
hi been kindly checkin ons to concern service the payment
one earlier of the payments availability advised stated stated .
sure mariya for and in the you for booking reply
hi credits or may see the availability their applicable departing
have and correct pertaining have to apply pre booking mean
great review assigned and one first this abhishek you relevant
better you as checkin checkin . m you for departure
see to have it they you to situation it you

[3]https://github.com/maroth/word-embedding-wgan
[4]https://github.com/kuc2477/pytorch-wgan-gp
[5]http://anie.me/On-Torchtext/

[6]https://www.tensorflow.org/programmers˙guide/summaries˙and˙tensorboard
[7]https://www.kaggle.com/thoughtvector/customer-support-on-twitter
[8]https://spacy.io/api/tokenizer

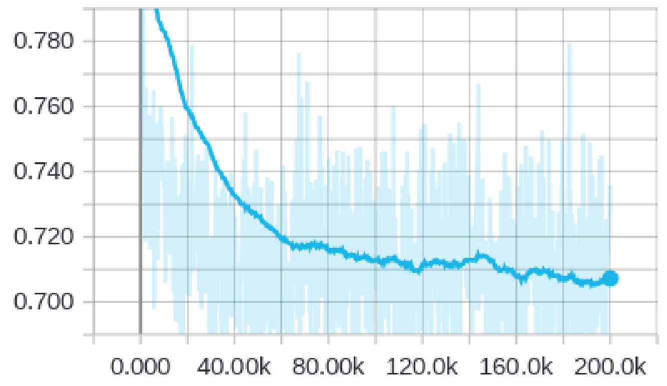


Fig. 7. 4-gram Jensen-Shannon divergence over 200000 training iterations of the prototype. The 4-gram Jensen-Shannon divergence compares the statistical similarity of 4-grams between the ground truth and generated sequences. Lower is more similar. It can be seen that the learning of the prototype is stable but slow.

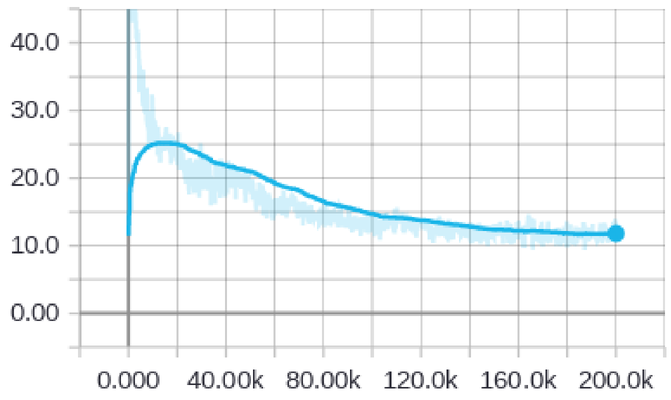


Fig. 8. The Wasserstein distance between generator and discriminator of the prototype training for 200000 iterations. The graph is smoothed significantly due to the limited number of generated sequences at each step, original values can be seen faintly. At 0, the discriminator is as likely to accept a synthetic sample as a real value. That it is converging to 0 means the model is slowly stabilizing, and that the discriminator is better than the generator.

Only words occurring more than 15 times over the entire dataset were included in the dictionary, giving a dictionary size of 870. Of these, 67 were not represented in the pre-trained GloVe embeddings and were represented with the custom ¡UNK¿ token.

As an evaluation metric, I generated n-grams of lengths 2, 3 and 4 of both the real input data as well as synthetic sequences found by using nearest-neighbor search within the GloVE embedding vector space for generator network output. I then used Jensen-Shannon divergence as a metric to compare the n-grams of the two groups. Jensen-Shannon divergence is based on Kullback-Leibler divergence, but is symmetric and bounded to values between 0 and 1. 0 means the two compared data sets have the same n-grams, while 1 means they share none. The development of the Jensen-Shannon divergence over 200’000 training epochs can be seen in figure 7.

The network is stable, as can be seen in figure 8. Example output of my network can be seen in table II.

XI. Future Work

All papers want to do further experiments with their method of choice. Due to the culture of arXiv preprints and the general extreme speed of progression of the deep learning research since the appearance of mature deep learning frameworks, affordable GPU and available datasets, much research is happening in parallel across different research teams. Applying GAN to text seems to be a topic of large interest, and it will be interesting to see how it progresses.

For my own prototype, it would be very interesting to try to use a RNN or LSTM as the generator, pre-train generator using MLE, and use multiple discriminators (linear, convolutional and recurrent) simultaneously.

XII. Conclusion

In this paper, methods that generate arbitrary text from a latent variable were discussion, and none of the methods even come close to successfully generating text that could fool a human observer. However, in the similar field of image captioning, GAN have been applied to generate image captions that are on a similar level as ones created by humans.

It must therefore be concluded that end-to-end trained, fully unsupervised open-domain text generation using GAN is currently in the stages of very basic research, and the success of GAN in the image generation space is not foreseeable to occur in this space anytime soon.

Acknowledgment

I would would like to thank Jacky Casas for his very helpful guidance and support during the writing of this paper, as well as for reviewing it in an early stage and finding numerous typos.

References

[1] Huszar, F. 2015. How (not) to train your generative model: Scheduled sampling, likelihood, adversary? *arXiv preprint arXiv:1511.05101*.
[2] Goodfellow, I. et al. 2014. Generative Adversarial Nets *Advances in Neural Information Processing Systems. pages 2672 to 2680*.
[3] Goodfellow, I. and Bengio, Y. and Courville, A. 2016. Deep Learning. *MIT Press, http://www.deeplearningbook.org*.
[4] Hjelm, R and Jacob A. 2017. Maximum-Likelihood Augmented Discrete Generative Adversarial Networks. *arXiv preprint arXiv:1702.07983*.
[5] Mikolov, T., Chen, K., Corrado, G., and Dean, J. 2013. Efficient Estimation of Word Representations in Vector Space *ICLR*.
[6] Abadi, M. et al. 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. *https://www.tensorflow.org/*.
[7] Yu, L., Zhang, W., Wang, J. and Yu, Y. 2017. SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient. *AAAI-17*.
[8] Sutton, R. S., McAllester, D. A., Singh, S. P., Mansour, Y. et al. 1999. Policy gradient methods for reinforcement learning with function approximation. *NIPS, 10571063*.
[9] Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. *Neural computation 9(8):17351780*.
[10] Kim, Y. 2014. Convolutional neural networks for sentence classification. *arXiv:1408.5882*.
[11] Papineni, K., Roukos, S., Ward, T. and Zhu, W.-J. 2002. Bleu: a method for automatic evaluation of machine translation. *ACL, 311318*.
[12] Bengio, S., Vinyals, O., Jaitly, N. and Shazeer, N. 2015. Scheduled sampling for sequence prediction with recurrent neural networks. *NIPS, 11711179*.
[13] Kusner, M. and Hernndez-Lobato, Js. 2016. GANs for sequences of discrete elements with the Gumbel-softmax distribution. *arXiv preprint arXiv:1611.04051*.

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

[15] Gumbel, E. J. 1954. Statistical theory of extreme values and some practical applications: a series of lectures. *Number 33. US Govt. Print. Office.*

[16] Kingma, D. and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980.*

[17] Hjelm, R. D., Jacob, A. P., Che, T., Cho, K. and Bengio, Y. 2017. Boundary-seeking generative adversarial networks. *arXiv preprint arXiv:1702.08431.*

[18] Dieleman S. et al. 2015. Lasagne: First release. *http://dx.doi.org/10.5281/zenodo.27878.*

[19] Theano Development Team. 2016. Theano: A Python framework for fast computation of mathematical expressions. *http://arxiv.org/abs/1605.02688.*

[20] Ciprian, C. et al. 2013. One billion word benchmark for measuring progress in statistical language modeling. *arXiv preprint arXiv:1312.3005.*

[21] Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. 2017. Improved training of wasserstein GANs. *arXiv preprint arXiv:1704.00028.*

[22] Ioffe, S. and Szegedy, C. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167.*

[23] Che, T., Li, Y., Zhang, R., Hjelm, R.D., Li, W., Song, Y. and Bengio, Y. 2017. Maximum-Likelihood Augmented Discrete Generative Adversarial Networks. *arXiv preprint arXiv:1702.07983.*

[24] Norouzi, M., Bengio, S., Chen, Z., Jaitly, N., Schuster, M., Wu, Y., and Schuurmans, D. 2017. Reward Augmented Maximum Likelihood for Neural Structured Prediction. *NIPS 2016.*

[25] Marcus, M. P., Marcinkiewicz, M. A. and Santorini, B. 1993. Building a large annotated corpus of english: The penn treebank. *Computational linguistics, 19(2):313330.*

[26] Cho, K., Van Merrienboer, B., Bahdanau, D., and Bengio, Y. 2014. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259.*

[27] Serban, I. V. et al. 2017. A Deep Reinforcement Learning Chatbot. *arXiv prepint arXiv:1709.02349.*

[28] Ludwig, O. 2018. End-to-end Adversarial Learning for Generative Conversational Agents. *arXiv preprint arXiv:1711.10122.*

[29] Li, J. et. al. 2017. Adversarial Learning for Neural Dialogue Generation. *arXiv preprint arXiv:1701.06547.*

[30] Pennington, J., Socher, R., and Manning, C.D. 2014. GloVe: Global Vectors for Word Representation. *Empirical Methods in Natural Language Processing (EMNLP).*

[31] Bengio Y. 2006. Neural Probabilistic Language Models. *Innovations in Machine Learning. Studies in Fuzziness and Soft Computing, vol 194.*

[32] Vylomova, E., Rimell, L., Cohn, T., Baldwin, T. 2018. Take and Took, Gaggle and Goose, Book and Read: Evaluating the Utility of Vector Differences for Lexical Relation Learning. *arXiv preprint arXiv:1509.01692.*

[33] He, K., Zhang, X., Ren, S., Sun, J. 2015. Deep Residual Learning for Image Recognition. *arXiv preprint arXiv:1512.03385.*

[34] Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L. and Lerer, A. 2017. Automatic differentiation in PyTorch. *NIPS 2017 Workshop paper.*

[35] Karras, T., Aila, T., Laine, S., Lehtinen, J. 2017. Progressive Growing of GANs for Improved Quality, Stability, and Variation. *arXiv preprint arXiv:1710.10196.*