Neural Language Generation Using Generative Adversarial Network (GAN)

Nobel Jacob Varghese
7002401
noja00001@stud.uni-saarland.de

Abstract

The aim of this work is to understand the current methods that are prevalent to generate natural language using neural network architectures, more specifically using generative adversarial networks (GANs). Furthermore, to explore how successfully GANs can be incorporated into language generation tasks and to check if they have achieved the same level of success as in the computer vision domain. This work discusses the most popular methods, the common evaluation criteria, and a few seminal papers which guided research for using GANs for language generation tasks.

1 Introduction

Neural language generation is the process of generating text which meets specific requirements. There exist various architectures to generate text like RNNs (Recurrent neural networks), Transformers etc. It is a common fallacy that text generation tasks are only limited to generating an output when provided with input or a constraint (eg- generate text which invokes happiness, fear etc) . Various tasks associated with language generation include 1) text summarization 2) question generator 3) distractor generator etc 4) image to text conversion etc .

Traditional methods utilized feature extraction techniques with a classifier at the end of the network. In traditional methods, only the classifier was learned to improve the classification scores. There existed a large number of pitfalls for this traditional approach. Features were often handcrafted and fixed, the manually engineered features might be too general or specific. To overcome the issues associated with them, the idea of deep learning was introduced which introduced the concept of joint training of the feature extraction and the classifier (end to end training) . The basic idea of all deep neural networks remains the same. Do a forward pass and generate output for the given input, when there is an error, propagate errors backward to update weights (back propagation). Deep Neural Networks (DNN's) provided better self-learning capabilities and eliminated the need for manual feature engineering.

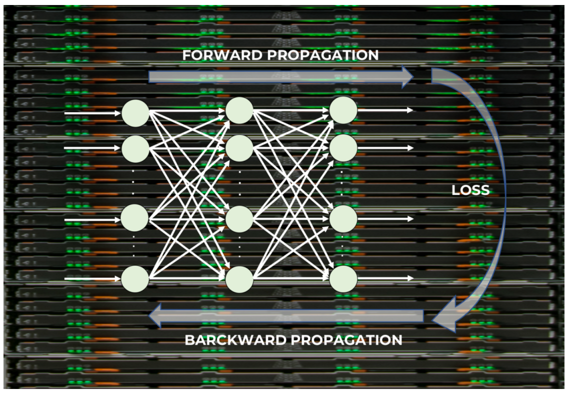


Figure 1: General DNN training procedure[1]

Deep Neural Networks (DNN) are commonly used for computer vision, fraud detection, natural language processing, data refining etc. Adversarial generative models (GANs) which is a subset of the deep neural networks have shown good success in image generation tasks, and this success of adversarial networks in computer vision domain has motivated research for other using them and adapting them to other domains.

Using Generative adversarial models to produce text/language is an interesting dimension to explore because of two core reasons : 1)They use an unsupervised mechanism where no annotated data is required 2)The Training and generation process itself. Generative adversarial networks have a training and generation mechanism wherein they deploy a combination of neural networks, namely the generator and the discriminator which proceeds by mocking a two-player game scenario, which is quite different from the standard deep neural network procedure.

2 Current approaches in neural language generation

This section discusses the state-of-the-art and most commonly deployed methods that are used for neural language generation.

2.1 Statistical language models vs Neural language models

*A **statistical language model** is a probability over sequences of words. Given such a sequence, say of length m, it assigns a probability $P(w_1, \ldots, w_m)$ to the whole sequence*[2]. Language modes are used for the prediction of the next word based on context, assigning probabilities to different available choices, and generating text. The core idea of language modes is that they need a mechanism to incorporate what it has already seen(memory), as the task in hand depends on the context. One of the major challenges of language models is the issue of unseen words as the model would assign zero probability to them. To conquer these issues a divide and conquer mechanism is usually enforced where a complex long sentence is broken down into smaller distribution of words, also known as n-grams. The Famous Markov assumption states that for an n-gram the ”relevant ” history for a current word is only n-1. Markov chains can be employed for language generation. Markov chains have a finite set of states, and there exists a transition between these states, which is based on some probability calculation depending on the configuration to meet the task at hand. They fail to capture long-term dependency at a high level of precision since the next transition is dependent only on the current state, this is the reason they are often referred to as ”memory-less”. Since they are not able to capture long dependencies, they are not able to produce coherent content which can absorb the entire context[3].

***Neural language models** (or continuous space language models) use continuous representations or embeddings of words to make their predictions. These models make use of Neural networks[4].* The use of neural language models for various language tasks and text generation has produced results that have outperformed the traditional methods and a strongly growing field of research. From RNNs, Long Short-Term Memory (LSTM) , and finally the state-of-the-art Transformers, neural based methods have revolutionized text generation process.

2.2 Embeddings

The techniques which are used to represent words in a mathematical notation are referred to as word embeddings[5] .As a neural-based system can only process numbers, the idea is to provide the input in a format that the neural system can understand. The most popular word embedding/word vectors are one-hot encoding, frequency/count-based, prediction-based, static embedding (word2vec, Glove), and contextualized embeddings (Transformers) .

One-hot encoding is one of the simplest encoding techniques and follows the following steps: 1) All the elements of vocabulary are ordered alphabetically. 2) Vectors are constructed which have the size of the vocabulary. 3) The corresponding index(based on alphabetical ordering) of the vector gets a ’one’(1) and the rest have a value ’zero’(0).The advantage of this method is that each word gets a unique representation. However, when the size of the vocabulary is very large, the size of the vector gets difficult to manage to cause increasing computational complexity, furthermore there is no notion of similarity in this case.

An example of frequency/count-based embeddings is TF-IDF.frequency/count bases embeddings are developed on the idea that words (representations)that are similar semantically should appear closer in the vector space.TF-IDF for a word in a document is calculated by multiplying two different terms - term frequency and inverse document frequency.

Term Frequency is defined as - (number of times a word appears in a document)/(number of words in the document).

Inverse Document Frequency is defined as the log((number of documents)/(number of documents containing the word))

To calculate the similarity of the tf-idf vectors different methods like Euclidean distance and cosine similarity are employed. If d1 and d2 are tf-idf vectors, the cosine similarity is given as-

$$cos\theta = \frac{d1.d2}{|d1|.|d2|}$$

The numerator is the formula is the dot product of the two vectors while the denominator is the product of Euclidean norms of each of the vectors.

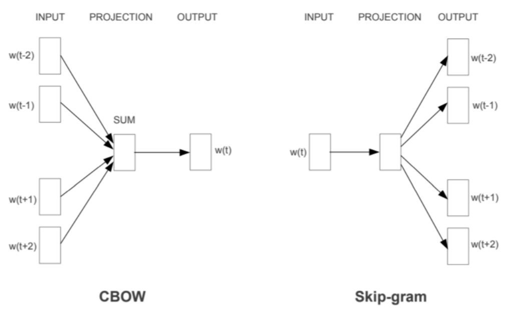


Figure 2: CBOW and Skip-gram[6]

The most common embedding techniques are **1) CBOW** and **2) SKip-gram** [7]. Under the CBOW procedure, a prediction based on the context (words around it or neighbors) is employed. Also to account for how much context should be incorporated while making the prediction, the window size is specified. Skip-gram can be seen as the inverse of CBOW, here the task is to predict the context when provided by the target word[8]. Since this is a supervised learning task, the data that is required for the network to train is obtained by sliding a window over the data set which is freely available. The simple sliding process can create a supervised data set with the least expense. The size of the sliding window is a parameter that can be modified.

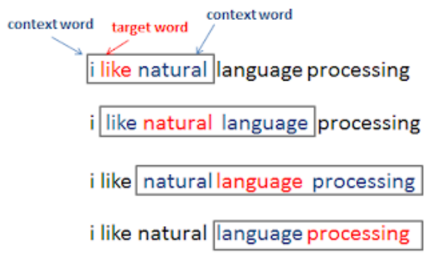

context word
target word
context word
i like natural language processing
i like natural language processing
i like natural language processing
i like natural language processing

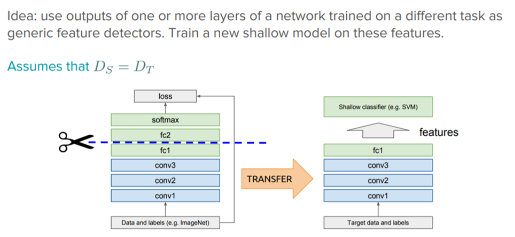
Figure 3: Sliding window for generating data[9]

Other popular embedding techniques are **Word2Vec** - It takes text as input, uses a neural network to learn embeddings, and groups words with similar meanings in the vector space. **GloVe**[10] - A global co-occurrence matrix is used in this case which gives slightly better performance over the prior. **FastText**[11] - Provides an improvement over Word2Vec, by changing the format in which the data is given to the neural network (using n-grams)[12] .

In a latent space, with good learned word embeddings, the words that have similar meanings appear close together to the extent that analogical reasoning can also be performed at this level.

2.3 Transfer Learning

Deep or more complex learning tasks usually require a large amount of data to train the network. This data is not easily available, also data collection is a tedious and expensive task. Transfer learning can be employed for such situations. The idea of transfer learning is to use a model which was trained for one task to be used for another related task. For deeper networks usually, the last few layers (classifiers) are re-initialized and trained again to adapt to the current task at hand while the other initial layers (convolution or pooling layers) are untouched.


Idea: use outputs of one or more layers of a network trained on a different task as generic feature detectors. Train a new shallow model on these features.

Assumes that $D_S = D_T$

Figure 4: Transfer Learning modifying the classifier layers[13]

Transfer learning should be used with caution as they are not always applicable/might not give the expected results.The table below provides a few guidelines on how to use transfer learning.

	highly analogous dataset	highly contrasting dataset
insufficient data	use linear classifier on top layer	perform linear linear classifier from different stages
suffient data	finetune a few layers	fine tune large number of layers

Table 1: Guide to use transfer learning [14]

2.4 Transformers

Transformers[15] are the current state-of-the-art technique used for language tasks ranging from translation, language generation, etc. Due to inherent sequentially, RNNs and other techniques are slow to train. Transformers are easier to parallelize and train. Furthermore, they are used for sequence-to-sequence applications, ie transforming a sequence of symbols to another sequence of symbols. The length of the data fed into the encoder

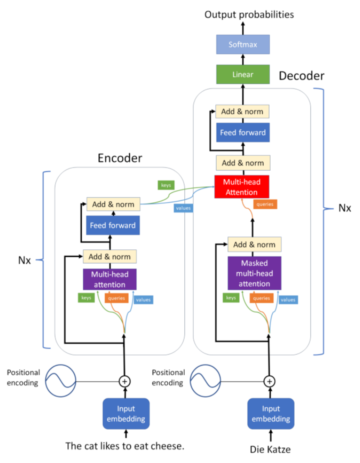


Figure 5: Transformer Architecture[16]

might be different from the length you receive back at the decoder.

BERT (Bidirectional Encoder Representations from Transformers) is based on the transformer encoder and usually consists of 12 layers stacked on top of each other. BERT encodes the input text such that each word is represented as a contextualized embedding, BERTS are also considered omnidirectional since their attention is pointing in all directions. **GPT (generative pre-trained transformer)** is built based on the decoder block and outputs one token at a time. The output becomes the input for the model in the next step, this technique is called "auto-regression".

The success of transformers can be attributed to its various attention mechanisms -**1) Self-attention 2) Cross-attention(encoder-decoder attention)** and **3) Masked-attention** . Self-attention in the encoder allows any word in the input to look at all the other words in the input including itself and compute attention scores, this can also be visualized as normalizing into a probability distribution - where the representation of one word is the sum of other weighted attentions of other words including itself.Cross attention - allows the decoder to gaze into the encoder. In this technique, the decoder looks at every word in the encoder and calculates how important are they for the current task at hand. Masked (Multi Headed) Attention- The decoder would generate the output from left to right (word by word), step by step in an auto-regressive way. Masked attention would allow the decoder to look back at everything the decoder has already produced while preventing it by looking into the future, this is the reason it is called "masked", the generated words are again fed back for the next step[17].

Positional Encoding - Self-attention treats the whole input string as a bag of words (BOW) or multi-bag of words. Because of this, it loses the idea of sequentiality. To counter this positional embedding is used to let the transformer know the order of words. The positional encoding uses sine and cosine waves- each word is associated with 2 positional values which arrive from the sine and cosine wave, unique with respect to the position of sine and cosine wave which is run through the input string. Combination of theses waves help identify relative and absolute positioning of words in a string[18].

3 Generative models and GANs

3.1 Supervised vs Unsupervised Learning

To understand the increasing significance of GANS, it is essential to understand the difference between **supervised** and **unsupervised** learning tasks.

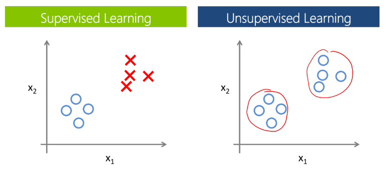


Figure 6: Supervised vs Unsupervised learning [19]

In supervised learning, task data is available/provided in format **(x,y)** - where x refers to the data point and y refers to the corresponding labels or class.The objective of supervised learning tasks is to discover a function that can map the data(x) to its corresponding label(y). Example- object detection, image captioning, translation tasks, etc.

On the contrary, an unsupervised learning task has only the data (x) and contains no labels. The objective here is to find the obscure properties of data. Examples of unsupervised learning tasks are clustering, dimensionality reduction etc. While annotated data of the format (x,y) is expensive to obtain, the growth of the internet has provided ample access to data with no labels and is very easily available/easy to obtain.

3.2 Generative Models

The objective of Generative models are to generate data from the same distribution, when provided with training data using unsupervised learning.Assume that training data has the distribution D_{train}, the objective of a generative model is to generate samples D_{model} similar to D_{train}[20]. Maximize the objective such that -

$$Dmodel = Dtrain$$

Generative models are further classified into **Explicit** and **Implicit**.In explicit density estimation, D_{model} is explicitly designed and solved, whereas in implicit density estimation it learns a model which can generate data without explicitly designing it. GANS fall under the umbrella of implicit density estimation where an explicit density function is not defined and learns the distribution of the training data by mocking a two-player game.

3.3 Training GANS

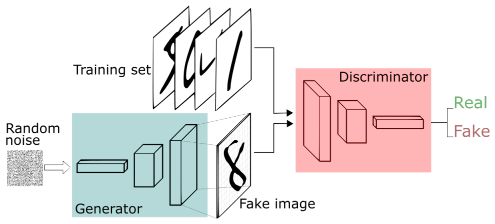


Figure 7: Two-player GAN network[21]

The objective here is to generate samples from the model. Furthermore, if the generated samples look similar to the training distribution, then we can conclude the model is a good estimator.The generated samples should be similar and the model should be fixed to produce more of the same.Since it is not possible to divulge how likely the generated sample is, using log-likelihood is out of scope, since they need an estimate on likelihood and to back-propagate the maximize the likelihood.

Figure 8: Images produced by CoGAN [22]

The GAN consists of two neural networks namely the **generator** and the **discriminator**.The generator tries to learn the underlying distribution and try to fool the discriminator network by generating outputs (fakes) such that they cannot be distinguished by the discriminator networks.The discriminator network tries to correctly categorize fake (generated by the generator network) and originals.

The training procedure of GANs mimics a min-max game. The parameters of the generator are theta(G) and the parameters of the discriminator are theta(D). The generator is trying to minimize this function while the discriminator is trying to maximize this function, this function is looking for a saddle point equilibrium.

One sample(Gaussian Distributed data) goes through the generator and data is generated. This data is run through the discriminator which tries to distinguish if it is real or fake(generated). The generator wants to confuse the discriminator and d(G(theta(Z))) would go down and 1-(G(theta(Z)) will go up, while the discriminator wants to correctly classify real data as real and fake data as fake.Discriminator wants to maximize the function such that d(theta(x)) is close to 1 for real data and d(g(z)) is close to zero for fake data.Generator theta(g) wants to minimize the objective such that d(g(z)) is close to 1.

The general process of training GANs consist of the following steps : 1) Problem definition 2) Picking the correct GAN architecture 3) Training the discriminator on the real data 4) Training the generator network 5) Again training the discriminator with fake data 6) Using the outputs of the Discriminator to better train the discriminator[23].

$$\min_{\theta_g}\max_{\theta_d}\left[\mathbb{E}_{x\sim p_{data}}\log \underbrace{D_{\theta_d}(x)}_{\text{Discriminator output for real data x}} + \mathbb{E}_{z\sim p(z)}\log(1-\underbrace{D_{\theta_d}(G_{\theta_g}(z))}_{\text{Discriminator output for generated fake data G(z)}})\right]$$

Figure 9: GAN network loss function [24]

GANs also suffers from a variety of issues that are sometimes not easy to deal with. Mode collapse is a common issue with GANs where the generator is not able to produce data with variousness, or can only produce a small subset of outcomes. Another issue is the Non-convergence and instability, where the parameters move around and never converge to global minima. Vanishing Gradient is a common problem associated with GANs, where the discriminator network has high accuracy in prediction leaving the generator gradient to move towards zero, causing almost zero learning. With the two different networks involved, training GANs are generally more complicated.

3.4 GANS for language tasks

GAN's were initially designed to perform tasks that had data in a continuous format. If the data was in a continuous format, it would be possible for the GANs to twitch the output by a little bit to provide better output in the next run. This is one of the reasons why GANs have not achieved much success in the Language domain as compared to the computer vision domain where the pixel updates can be carried out with much ease. In a latent space learned word embeddings are not continuous but discrete.

On a discussion forum Ian Goodfellow, the original inventor of GAN quotes the following - *"You can make slight changes to the synthetic data only if it is based on continuous numbers. If it is based on discrete numbers, there is no way to make a slight change. For example, if you output an image with a pixel value of 1.0, you can change that pixel value to 1.0001 on the next step. If you output the word "penguin", you can't change that to "penguin + .001" on the next step, because there is no such word as "penguin + .001". You have to go all the way from "penguin" to "ostrich". Since all NLP is based on discrete values like words, characters, or bytes, no one really knows how to apply GANs to NLP yet."*[25]. This shows that GANs cannot be directly used for language tasks and requires quite a bit of modification to adapt themselves for language tasks. Many techniques like Reinforcement learning, Gumbel-softmax, policy gradient, professor-forcing, and maximum likelihood have been used for this purpose.

The earliest solutions for this problem were suggested in the paper MaliGAN(Maximum-Likelihood Augmented Discrete Generative Adversarial Networks) [26], which proposed a new normalized likelihood maximum likelihood approximation and provided better stable training than the standard GANs.To tackle the problem of dealing with discrete data that was generated, a method called **Gumbel-soft-max**[27] can be employed. In a normal scenario, it is not possible to do back propagation for discrete data. Gumbel soft-max technique allows to approximate categorical distribution and updates weights through error propagation.

Assume we have a categorical variable z with corresponding class probability values. To pick samples from a categorical distribution like z, we use the Gumbel-Max trick. The arg-max here is difficult to differentiate, hence to counter this we use the softmax, which is a softened version of the arg-max. So we get a softened z, denoted as y[28].

$$Z = \text{onehot}\left(\text{argmax}_i \left\{G_i + \log\left(\pi_i\right)\right\}\right)$$

$$y_i = \frac{\exp\left(\left(\log\left(\pi_i\right) + g_i\right)/\tau\right)}{\sum_{j=1}^{k} \exp\left(\left(\log\left(\pi_j\right) + g_j\right)/\tau\right)}$$

Reinforcement learning based training was one of the better solutions which provided high-quality solutions. Every reinforcement learning-based problem has an Agent, which is the key component that performs the actions and tries to achieve the end state. In the case of text generation, the end state would be generating the EOS(end of sentence), the end of the sentence token. Each state is the number of tokens generated so far, actions are the next tokens to be generated to reach the goal state. Out of all the available actions that are applicable at a given state, the optimal action is picked by a policy function. A good text generation model should have a good policy that provides the agent highest performance.

3.5 Evaluation of text generated by GANS

Like other language tasks, the data generated by GANS are evaluated using metrics like BLEU[29], ROUGE, perplexity and Distinct n-gram (Dist-n) .Bilingual Evaluation Understudy (BLEU) compares the generated text to a given reference text, this technique combines the concept of both precision and recall to produce a score. Even though it is one of the most commonly used evaluation metrics, it suffers from quite a few issues like giving very low scores to perfect paraphrases even though they convey the same meaning, in most cases, they do not inform the user what happened correctly and what went wrong. Recall-Oriented Understudy for Gisting Evaluation (ROGUE-n) returns a value that compares the number of matching n-grams between the generated text and reference text. ROGUE-1 would compare uni-gram overlap, ROGUE-2 for bi-gram overlap, etc. Perplexity is another common method, which has been traditionally employed to evaluate language models,and can be interpreted in layman terms as "lower the perplexity, better the model". Dist-n is another metrics that measures the ratio of distinct n-grams to the total number of n-grams, generated by the model.

3.6 Discussion about papers on GANS for text generation

We would now discuss some original papers which have produced quite remarkable results and paved the way for research in this direction.

SeqGAN - Sequence Generative Adversarial Nets with Policy Gradient[30] - This paper was published in 2017 before the GPT and BERT models were introduced. The issue of exposure bias caused by using log-likelihood while training an RNN is addressed in this paper. Exposure bias can occur when the model tries to predict the next tokens, based on the previous history which was never seen during training, which causes a cascading effect as the model moves to longer sequences. In Reinforcement learning, the core idea is to take the next action so as to maximize the reward. *Train a θ-parameterized generative model G_θ to produce a sequence $Y_{1:T} = (y_1, \ldots, y_t, \ldots, y_T), y_t \in \mathcal{Y}$, where $\mathcal{Y}$ is the vocabulary of candidate tokens. We interpret this problem based on reinforcement learning. In timestep t, the state s is the current produced tokens $(y_1, \ldots, y_{t-1})$ and the action a is the next token y_t to select. Thus the policy model $G_\theta(y_t \mid Y_{1:t-1})$ is stochastic, whereas the state transition is deterministic after an action has been chosen*[31] . The discriminator like in other GAN was trained by data produced by the discriminator and real data. The crux of the model was how the generator was trained- the generator had Reinformcent learning-based training solution where the reward was awarded by the discriminator and distributed to sub-action states using Monte Carlo Search. An RNN was used as generator and a CNN was used as discriminator. To generate real data(to train the discriminator), an LSTM was employed. Negative log-likelyhood (NLL) scores was compared against 4 different baseline models (a random token generator, Maximum Likelihood estimation (MLE) trained LSTM, scheduled sampling and Policy Gradient with BLEU), in which the SeqGan outperformed the other baselines.

LeakGAN (Long Text Generation via Adversarial Training with Leaked Information) [32] -One of the issues that LeakGAN tries to address is that only after the entire text has been generated by the generator, the discriminator can provide feedback on which direction to proceed and lacks an intermediate stage that might provide better results by guiding the generator. A CNN is used as a discriminator for extracting the features and classification(as real or fake). Both the MANAGER and WORKER module introduced in the paper are LSTM networks. The MANAGER "leaks" a latent vector to the WORKER by extracting features of the words that are already generated, which can guide the WORKER for the next word generation. The WORKER module is an LSTM network that contains information on the words that are produced until now, both the information(from the MANAGER and WORKER) mode is now incorporated while producing the new output. The MANAGER module is was also termed a spy, which is quite interesting and gives an easy understanding of the underlying process. Quoting from the paper - *"As such, the MANAGER module can be also viewed as a spy that leaks information from the discriminator to better guide the generator"*[33] .

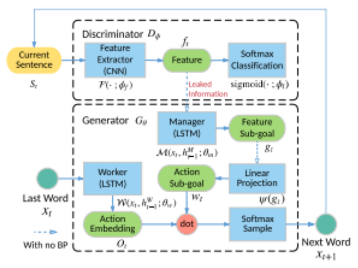
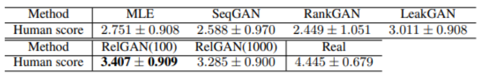


Figure 10: LeakGAN architecture[34]

During Training the authors utilized 3 techniques to deal with common pitfalls and errors which usually occur while dealing with GANS and Reinforcement Learning. Bootstrapped Rescaled Activation helped in stabilizing activation's and solving the vanishing gradient problem. Interleaved Training by toggling between adversarial and supervised training after every 15 epochs helped to an extent to prevent the model from mode collapse.Temperature Control- Boltzmann temperature constant alpha(α) helped balance between exploration and exploitation during reinforcement learning. During the evaluation, the LeakGAN showed significant improvement in performance as compared to the previous discussed SeqGAN in BLEU scores and NLL metrics evaluation.

RelGAN -(RELATIONAL GENERATIVE ADVERSARIAL NETWORKS FOR TEXT GENERATION)[35] : This paper moves away from the use of Reinforcement learning and introduces throws light in a new direction, this paper also has a controllable, adjustable parameter that helps to balance between quality and diversity. The 3 main components of this paper are - 1) Relational Memory-based Generator 2) Gumbel-Softmax Relaxation 3) Multiple Representations in Discriminator. Relational Memory-based Generator : The relational memory takes into account a specific set of memory slots and allows for the exchange between them using the self-attention mechanism. This technique uses self-attention(from original Attention Is All You Need [36] paper) while updating the memory while combining with new observations. Relational memory has more expressing power and is better for incorporating long-distance dependencies as compared to LSTMs. Gumbel-Softmax Relaxation technique allows to approximate categorical distribution and updates weights through error propagation. Multiple Representations in Discriminator has a framework in which Discriminator tries to embed multiple representations, the advantage of using multiple representations are that they provide more diverse and informative guidance for the generator updates[37] .

The experiments were conducted on synthetic data sets and real-world datasets. In the Synthetic data set setting negative log-likely-hood (NLL-oracle) metric was employed to measure quality and negative log-likely-hood (NLL-gen) to incorporate diversity, the corresponding formulars are given here. $\mathrm{NLL}_{\text{gen}} = -E_{r_{1:T}\sim P_r} \log P_\theta\left(r_1, \cdots, r_T\right)$, $\mathrm{NLL}_{\text{oracle}} = -E_{y_{1:T}\sim P_\theta} \log P_r\left(y_1, \cdots, y_T\right)$. RelGAN outperformed the previously discussed SeqGAN and LeakGAN by providing the lowest NLL-oracle score, as the better score is the least. In a real-setting COCO image caption dataset and EMNLP2017 WMT News dataset was used, and RelGAN was able to outperform the other prominent models like SeqGAN and LeakGAN. From the evaluation part, it was clear that REAGAN in terms of BLEU scores where able to perform better than the other previous models which were considered and were able to generate high-quality sentences on EMNLP2017 WMT News dataset[38] . For the Human evaluation study Amazon mechanical Turks were employed which provided high scores for the sentences generated by RelGAN. The paper also mentions that they had randomly sampled 100 sentences from each of the model and real-dataset, only 10 people were asked to give a rating score on a scale of 1-5. However, the authors in the paper have not mentioned which aspect of evaluation was considered for this scoring.



Method	MLE	SeqGAN	RankGAN	LeakGAN
Human score	2.751 ± 0.908	2.588 ± 0.970	2.449 ± 1.051	3.011 ± 0.908

Method	RelGAN(100)	RelGAN(1000)	Real
Human score	**3.407 ± 0.909**	3.285 ± 0.900	4.445 ± 0.679

Figure 11: Human evaluvation on RELGAN[39]

DPGAN- The DPGAN (Diversity-Promoting Generative Adversarial Network for Generating Informative and Diversified Text) [40] gives a low-reward score for repetitive text which is generated by the model and a high reward for more diverse texts. The discriminator that is proposed by the authors can better understand diverse and distinct texts from repetitive ones by eliminating the saturation problem that is associated with other discriminators[41] .

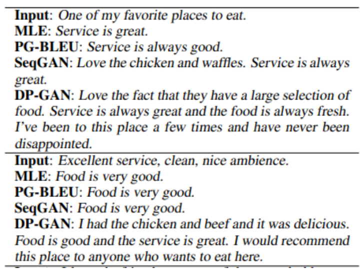
The generator G is a sequence to sequence model, based on the LSTM encoder which generates sentences is coupled with the discriminator based on unidirectional LSTM which tries to correctly classify the real and generated(fake) text. The output of the discriminator is a cross-entropy score which is provided to the generator to guide its output.

The generator G is a sequence to sequence model, based on the LSTM encoder which generates sentences is coupled with the discriminator based on unidirectional LSTM which tries to correctly classify the real and generated(fake) text. The output of the discriminator is a cross-entropy score which is provided to the generator to guide its output. The cross- entropy-based is given by $R\left(y_{t,k}\right) = -\log D_\phi\left(y_{t,k} \mid y_{t,<k}\right)$ where y is the t-th sentence and k-th word and is used for adversarial reinforcement learning for the generator G.

The reward function has two parts, reward is split into word level $\left(R\left(y_{t,k} \mid y_{t,<k}\right) = -\log D_\phi\left(y_{t,k} \mid y_{t,<k}\right)\right)$ and sentence level rewards $\left(R\left(y_t\right) = -\frac{1}{K}\sum_{k=1}^{K} \log D_\phi\left(y_{t,k} \mid y_{t,<k}\right)\right)$. A sentence-level reward is calculated by averaging over all the words of the sentence. A total reward is calculated by using the word-level and sentence level reward, the core idea is to penalize repetitive text generation and provide a low score for text that are of repetitive nature.

The model was tested on 3 data-sets- Yelp, Amazon Review Generation Dataset , and OpenSubtitles Dialogue data-set. The proposed model was applied for review generation and dialogue generation tasks and compared to MLE, PG-BLUE, and SeqGAN tasks.The image below shows the performance of

DP-GAN on the Yelp dataset, and it can be seen the DP-GAN generates more meaningful sentences than the others.



Input: *One of my favorite places to eat.*
MLE: *Service is great.*
PG-BLEU: *Service is always good.*
SeqGAN: *Love the chicken and waffles. Service is always great.*
DP-GAN: *Love the fact that they have a large selection of food. Service is always great and the food is always fresh. I've been to this place a few times and have never been disappointed.*

Input: *Excellent service, clean, nice ambience.*
MLE: *Food is very good.*
PG-BLEU: *Food is very good.*
SeqGAN: *Food is very good.*
DP-GAN: *I had the chicken and beef and it was delicious. Food is good and the service is great. I would recommend this place to anyone who wants to eat here.*

Figure 12: DP-GAN on Yelp data-set[42]

Since the core idea of the paper was to generate more diverse tokens/words, the dist-n metric which is the ratio of the number of distinct n-grams to the total number of n-grams was measured. 3 different setting within the DP-GAN was used DP-GAN(S) with sentence-level reward, DP-GAN(W) with word-level reward, and DP-GAN(SW) with combined reward. In almost all cases the DP-GAN(SW) surpassed all the baselines.

4 Conclusion

Even though there have been several papers published regarding the usage of GANS for text generation, it has not seen much success in generating coherent and controllable text in comparison to Transfomer based and LSTM based architectures. The core issue with GANS for text generation is that they were initially designed for tasks with continuous data and not for discrete data, almost all the papers have used the Gumbel soft-max technique or Reinforcement learning to deal with this issue and have been successful to some extent. Recent research is trying to incorporate large pre-trained language models like GPT-2 with GANs to generate high-quality text which is a new learning direction that is evolving at a good pace.

5 References

[1]https://towardsdatascience.com/learning-process-of-a-deep-neural-network-5a9768d7a651

[2]https://en.wikipedia.org/wiki/Language_model

[3]https://analyticsindiamag.com/hands-on-guide-to-markov-chain-for-text-generation/

[4]https://en.wikipedia.org/wiki/Language_model

[5]https://towardsdatascience.com/word-embedding-techniques-word2vec-and-tf-idf-explained-c5d02e34d08

[6,7,8]Efficient Estimation of Word Representations in Vector Space Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean (https://arxiv.org/abs/1301.3781)

[9]https://thinkinfi.com/word2vec-skip-gram-explained/

[10]GloVe: Global Vectors for Word Representation Jeffrey Pennington, Richard Socher, Christopher Manning (https://aclanthology.org/D14-1162.pdf)

[11]Enriching Word Vectors with Subword Information Piotr Bojanowski, Edouard Grave, Armand Joulin, Tomas Mikolov (https://arxiv.org/abs/1607.04606)

[12]https://towardsdatascience.com/the-three-main-branches-of-word-embeddings-7b90fa36dfb9#:~:text=Word2Vec%20takes%20texts%20as%20training,that%20two%20words%20appear%20together.

[13]https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a

[14]https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a

[15,16]Attention Is All You Need Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin (https://arxiv.org/abs/1706.03762)

[17,18]https://jalammar.github.io/illustrated-transformer/

[19]https://www.coursera.org/learn/machine-learning

[20]Generative Adversarial Networks Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio (https://arxiv.org/abs/1406.2661)

[21]https://towardsdatascience.com/generative-adversarial-networks-explained-34472718707a

[22]Coupled Generative Adversarial Networks Ming-Yu Liu, Oncel Tuzel (https://arxiv.or

g/abs/1606.07536)
[23]https://www.analyticsvidhya.com/blog/2021/10/an-end-to-end-introduction-to-generative-adversarial-networksgans/
[24]https://towardsdatascience.com/generative-adversarial-networks-explained-34472718707a
[25]https://www.reddit.com/r/MachineLearning/comments/40ldq6/generative_adversarial_networks_for_text/
[26]Maximum-Likelihood Augmented Discrete Generative Adversarial Networks Tong Che, Yanran Li, Ruixiang Zhang, R Devon Hjelm, Wenjie Li, Yangqiu Song, Yoshua Bengio https://arxiv.org/abs/1702.07983
[27]Categorical Reparameterization with Gumbel-Softmax Eric Jang, Shixiang Gu, Ben Poole https://arxiv.org/abs/1611.01144
[28]https://www.zzzdavid.tech/GumbelSoftmax/
[29]ROUGE: A Package for Automatic Evaluation of Summaries Chin-Yew Lin https://aclanthology.org/W04-1013//
[30,31]SeqGAN:Sequence Generative Adversarial Nets with Policy Gradient Lantao Yu, Weinan Zhang, Jun Wang, Yong Yu (https://arxiv.org/abs/1609.05473)
[32,33,34]Long Text Generation via Adversarial Training with Leaked Information Jiaxian Guo, Sidi Lu, Han Cai, Weinan Zhang, Yong Yu, Jun Wang (https://arxiv.org/pdf/1709.08624.pdf)
[35]RelGAN: Relational Generative Adversarial Networks for Text Generation Weili Nie, Nina Narodytska, Ankit Patel(https://openreview.net/pdf?id=rJedV3R5tm)
[36]Attention Is All You Need Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin (https://arxiv.org/abs/1706.03762)
[38,39]RelGAN: Relational Generative Adversarial Networks for Text Generation Weili Nie, Nina Narodytska, Ankit Patel(https://openreview.net/pdf?id=rJedV3R5tm)
[40,41,42]DP-GAN: Diversity-Promoting Generative Adversarial Network for Generating Informative and Diversified Text Jingjing Xu, Xuancheng Ren, Junyang Lin, Xu Sun(https://arxiv.org/abs/1802.01345)