Generating Opinionated Movie Reviews using GANs

Thomas Nortmann
982524

Lara McDonald
978124

March 26, 2022

Abstract

Generative Adversarial Networks have been immensely successful in generating artificial images almost indistinguishable from real-world examples. This success begs the question, of whether the architecture can also be applied to other domains, such as text. After discussing the challenges of adapting GANs to the textual domain and outlining existing approaches, we implement a modification of the LaTextGAN architecture. The model is trained on the IMDB Reviews Dataset and tasked with generating realistic movie reviews. Additionally, the model takes a sentiment parameter (positive/negative) and should generate opinionated reviews reflecting this sentiment. The final model is only partially successful. It is capable of generating novel sentences resembling movie reviews, however with multiple issues and no clear sentiment value.

1 Introduction

Generative Adversarial Networks (GANs) are a well-established tool for the creation of artificially generated images. Having experienced great success in this domain, it is interesting to explore the possibilities of adapting the architecture to other domains, such as text-based tasks.

The basic idea behind the GAN architecture is to let two neural networks play a min-max game against each other, or rather a re-imagined version of the Turing Test. Player One, the Generator, is tasked with creating samples based on a random noise input, while Player Two, the Discriminator, is tasked with deciding whether its input sample belongs to the set of real training samples or is simply an artificial sample created by Generator. The goal of the model is for the Generator to be able to generate images that are indistinguishable from real samples.

The idea behind our Final Project was to apply GANs, which have been very successful in generating images indistinguishable from actual photographs, to a textual domain. Specifically, we aimed at constructing a model which can generate plausible, opinionated movie reviews. When given a sentiment parameter (positive/negative) the GAN should be able to generate a review that aligns with the sentiment. To achieve this, the basic GAN structure needs to be adjusted to allow for input constructed from this discrete, textual domain, rather than the continuous spaces it was originally intended for. Additionally, vanilla GANs do not accept an external condition for sample generation, such as the sentiment parameter. Thus, a second adaptation to the architecture is necessary for the model to achieve its task.

We chose to implement a variation on LaTextGAN, an architecture proposed by D. Donahue and A. Rumhinsky in 2019 [4]. LaTextGAN circumvents the problem of having discrete outputs by utilizing a Variational Autoencoder to encode the input samples before GAN training. We also adapted the LaTextGAN architecture to allow for the additional sentiment parameter.

Unfortunately, our final implementation did not produce the anticipated results. While the model can generate output that matches the topic of movie reviews and contains grammatically correct sub-sentences, the generated sentences neither conform to the sentiment parameter nor are they coherent. After a few promising starting words, the sentence seemingly enters an endless loop of the same sub-sentences. Additionally, there seems to be little to no variety in the produced sentences.

This paper aims to document our research and development process. We begin by further detailing the dataset, discussing its benefits in regards to the task and the performed pre-processing steps. Afterward, we give an overview of relevant architectures and later of related approaches. We then describe and explain our architecture and training choices. Lastly, we discuss our results.

2 The Dataset

The model is trained on the *Large Movie Review Dataset* [6], also known as the *IMDB Reviews Dataset*, which is publicly available in TensorFlow Datasets. The dataset contains 100,000 user-generated movie reviews from the online movie database IMDB. Half of the reviews are unlabeled, while the other 50,000 are labeled according to the primary sentiment of the review, e.g. positive or negative. The sentiment classes were assigned according to the mandatory rating an IMDB user needs to leave alongside their review. Movies are rated on a 10-star scale, with 1 star equating to strong discontent and 10 stars to strong satisfaction.

The IMDB Reviews Dataset is advantageous in multiple ways for this project. Firstly, the labeled examples are balanced, i.e. there are 25,000 reviews classified as being positive and an equal amount of examples classified as negative. Since our GAN aims to generate reviews of both sentiments, an unbalanced dataset could skew results. Furthermore, the dataset only contains reviews with strong sentiment polarity. To be included in the negative sentiment class, a review's rating must be less or equal to 4 stars, while only reviews with a rating of 7 stars or higher are included in the positive sentiment class. The model can thus learn to generate sentences with a clear sentiment. [6]

#	label	text
...	...	...
5	1	This is a film which should be seen by anybody interested in. [...]
6	0	Okay, you have: <br/ ><br/ >Penelope Keith [...]
...	...	...

Table 1: Structure of IMDB Reviews Dataset [1]

2.1 Preprocessing Steps

Each sample in the dataset, as it is available in TensorFlow, is a *FeaturesDict* containing a *label* and a *text*. The *label* is an integer of value 0 (negative sentiment) or 1 (positive sentiment), while the *text* is the corresponding movie review in plain text. To use the data samples as input for our model, several preprocessing steps need to be performed on the plain text component of the tensors.

First, the text is cleaned by removing special characters. The resulting text is now tokenized with the BertTokenizer from TensorFlow Text. To each tokenized sentence, a [START] and [END] token is added to the beginning and end of the sentence, respectively. This is a necessary step, as it enables the Generator to learn how to begin a review and more crucially when to end it.

3 Background

3.1 Generative Adversarial Networks

Basic GAN Architecture

The first GAN architecture was proposed in 2014, in the paper "Generative Adversarial Nets" by Goodfellow et al. Its architecture fulfills the basic principle of the min-max game, as described above. Two neural networks, the Generator $G(z)$, and the Discriminator $D(x)$ work against one another: the Generator, aims to generate samples resembling real-world samples closely enough to fool the Discriminator, who tries to distinguish between the fake, generated samples and the real-world data. The underlying principle of this architecture is that the Generator approximates a transformation function, which maps a random input distribution z to the distribution underlying real data samples. On the other hand, the Discriminator approximates the distance between the generated samples and the real underlying distribution. Ultimately, the training procedure results in a minimax game between the Discriminator and the Generator. [5] [12]

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

Figure 1: The objective function of GAN training. [5]

For a GAN to converge, it is necessary to reach, or at least come close to reaching, Nash equilibrium between the Generator and Discriminator. In game theory, the Nash equilibrium describes the state where both players' strategies are optimal, considering the other player's strategy. However, GANs are typically optimized using Stochastic Gradient Descent (SGD). As SGD isn't designed to find this equilibrium between Generator and Discriminator, basic GANs can suffer from failure to converge [9]. If the Discriminator outperforms the Generator by too much, vanishing gradients may occur, impeding convergence. If the roles are reversed and the Generator consistently fools the Discriminator, training may continue after the point of optimality, worsening results. Another common issue with GANs is mode collapse, which occurs when the Generator generates a successful example and cannot deviate from this, thus repeatedly generating only the same example. [2]

Wasserstein GANs

There are various changes to the standard GAN architecture that address its problems with slow convergence, or even lack thereof, and mode collapse. A widely accepted improvement is the use of Wasserstein Distance ("*Earth Mover's Distance*"), instead of the Jensen-Shannon Divergence used in a vanilla GAN, as a loss function. In a standard GAN, the JS Divergence measures the distance between

the real sample distribution and that of the fake samples generated by the Generator. During Discriminator training, this distance should then be minimized. In Wasserstein GANs (WGANs) the Discriminator is transformed to what the authors call *The Critic*. Instead of discriminating between real and fake samples, the Critic merely scores its input sample on how real it is. [2]

The advantage of WGANs lies in the continuously differentiable Earth Mover's Distance, which allows for the Critic to be trained to optimality without risking vanishing gradients, as is the case in the vanilla architecture. As the antagonist to the Generator is defined differently, there is no need to reach Nash equilibrium, thus stabilizing training. The issue of mode collapse is alleviated, as well. [2]

Conditional GANs

After GANs proved to be an effective tool for artificial image generation, the desire to directly modify the image class arose. Conditional GANs (CGANs) allow for this modification by training both Discriminator and Generator on an additional input parameter y. This parameter is simply concatenated with the input sample to either component while training occurs as usual. [7]

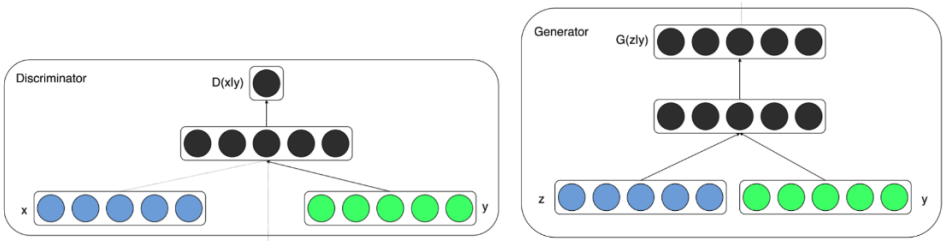

Figure 2: The CGANs Generator and Discriminator Components. [7]

3.2 Variational Autoencoders

The Variational Autoencoder (VAE) is an adaptation of the basic Autoencoder, allowing for more advanced sample generation. The architecture of autoencoders consists of an encoder component and a decoder component, both neural networks. The goal of the autoencoder during training is for its output to correspond to its input, while not simply learning the identity function. To achieve this, the data is passed through the encoder to a bottleneck, the embedding layer, only for this to be transformed to the original input by the decoder. At the embedding layer, the input data is now compressed while still capturing the essential characteristics of the input. [12]

VAEs build upon this architecture by additionally learning the underlying distribution of the embeddings. Using the reparameterization trick, i.e. assuming an underlying Gaussian distribution, noisy samples can now be generated from the embedding, allowing for variations on the original inputs. [8]

4 Text Generation with GANs

When applying the GAN architecture to the textual domain, the main hurdle to overcome is how to handle the value space of text-based data [11]. Compared to the continuous image space GANs were originally intended for, textual symbols are discrete and thus non-differentiable. This poses a major problem during training, as gradient backpropagation is no longer meaningfully possible [3]. There are multiple proposed workarounds to this issue, such as pre-training the model with Maximum Likelihood Estimation [10]. Other approaches make use of Reinforcement Learning Strategies or Word Embeddings.

4.1 SeqGAN

A well-known example of a GAN architecture with Reinforcement Learning Properties is the SeqGAN model proposed in 2017. The generative task in SeqGAN is posed as a reinforcement problem, with the generator aiming to maximize the expected reward of a generated sentence given by the discriminator. The more plausible the sentence, the higher the reward [13]. However, the generator does not produce a full sequence immediately. Instead, it samples the next token and then enacts a Monte Carlo Search over the rolled-out policy gradient to calculate the expected reward of a sequence. [12]

4.2 GAN2Vec

A different solution to the task is the GAN2Vec architecture, proposed in 2019. The objective of the generator is not to create sentences in a human-readable discrete format, but rather to generate Word2Vec embeddings. This overcomes the hurdle of discrete spaces, as the resulting vectors are no longer one-hot vectors but continuous embedding vectors. The paper also proposes a modification to GAN2Vec which allows for an additional conditional parameter to be given to the model, such that it is possible to generate text accordingly. [3].

4.3 LaTextGAN

The LaTextGAN architecture was proposed in 2019 to adapt GANs to discrete spaces without the need to use Reinforcement Learning techniques. Unlike vanilla

GANs, LaTextGAN consists of a WGAN and an additional VAE component, which itself is further comprised of two sub-components: an Encoder and a Decoder. [4]

The Generator and Discriminator components in LATextGAN are realized using a ResNet architecture, as ResNets can provide a more stable, deeper network. The Variational Autoencoder also utilizes LSTM cells to process the input sample at each time step and to model the long-term dependencies between words. At the embedding layer (colored orange in Figure 2) of the VAE, the sample is encoded as a low-dimensional *real* vector in the latent space. This representation is key to circumventing the problem of using GANs on discrete domains and enables backpropagation. [4]

Due to the additional VAE component, training has to occur in two distinct stages. Stage 1 focuses solely on the VAE, while the GAN is fixed. During Stage 2, the trained Encoder and Decoder are used to transform the discrete textual representation into real vectors and vice versa. The Generator no longer is tasked with generating novel sentences, but rather real-valued latent representations. To accommodate this modality change, the Discriminator is trained on the latent encoding of the training samples, to score the plausibility that the generated encoding is a real sample. The Decoder can later be used to transform the latent vector back into natural language. [4]

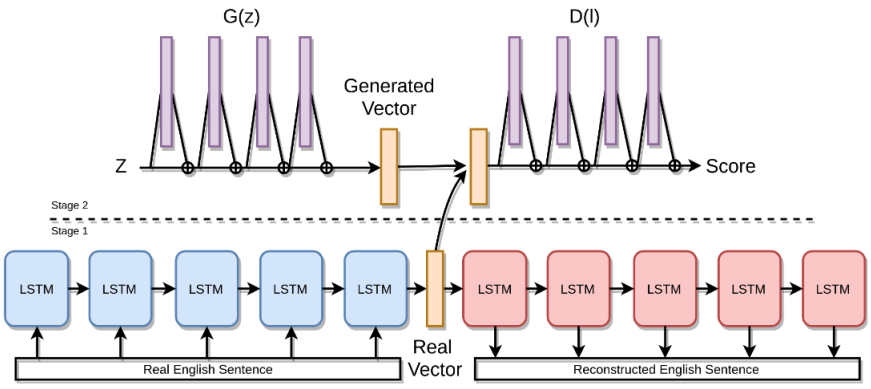

Figure 3: The basic LaTextGAN architecture [4]

5 Model

To achieve this specific task utilizing GANs, a few considerations regarding the model's architecture need to be made. Primarily, the architecture must overcome the problem of backpropagation in discrete spaces, such as movie reviews. In addi-

tion, the model must be able to generate reviews under the intended sentiment, i.e. whether the review should be positive or negative. To overcome the first concern, we decided to focus on the usage of word embeddings, rather than using reinforcement policies. Regarding the sentiment parameter, there was the option to implement GAN2Vec's conditional variation or to modify the LaTextGAN architecture to comply with the idea of Conditional GANs. We settled on the latter, as it allowed us to partially implement an established architecture, while also being able to extend the existing approach to fit the task at hand.

Underlying our generative model is the LaTextGAN architecture, however, adapted to the specific task definition, i.e. generating plausible, opinionated movie reviews, conditioned on the given sentiment parameter (positive/negative).
Note: Architecture and hyperparameter specifications marked as *s.b.a.* ("suggested by authors") are direct replications of the architecture proposed in [4].

5.1 VAE Component

The Variational Autoencoder is tasked with encoding the tokenized sentences into a real-valued latent representation, such that the GAN does not need to be able to deal with discrete values. After GAN training, it is then needed to decode a given representation into a human interpretable representation, i.e. from a vector back into sentences.

Encoder

The Encoder consists of an Embedding Layer, which transforms integer word-tokens into float-vectors of size 100. This promotes a causal connection between similar embedding vectors, a property that could not be captured by the alphabetically sorted integer tokens. Following the Embedding Layer is an LSTM Layer with 100 LSTM cells (*s.b.a.*), as was proposed in [4], which sequentially processes the sequence of embedded words. LSTMs are the RNN cells of choice for this architecture, as they are capable of modeling long-term, grammatical dependencies necessary for coherent sentences. The final encoding is produced by a Dense Layer. The layer receives a concatenation of the LSTM output and the LSTMs final cell state and outputs a vector the size of the Decoder's hidden state: the real-valued latent representation of the tokenized input sentence.

Decoder

The Decoder's task is to map a given latent encoding to a tokenized representation of a sentence, essentially reversing the encoding. Corresponding to the Encoder architecture, the Decoder consists of an Embedding Layer, followed by an LSTM Layer and a Dense Layer. The Embedding layer is constructed in the same way as in the Encoder and is also responsible for embedding the tokenized samples

in a float vector space of 100. The LSTM layer comprises 600 LSTM cells in the Decoder (*s.b.a.*) and acts as the Language Model of the system. Its hidden state is initialized with the latent representation, i.e. the encoding, and the layer receives the embedding generated in the previous layer as its input, producing an output sequence by predicting the next most probable word for the current input. Regularization is applied with a Dropout Layer with a rate of 0.5 (*s.b.a.*) to ensure strong connections throughout the network. Lastly, the sequence passes through a Dense Layer with softmax activation which maps the LSTM cell's hidden state to a vector containing the probability for each possible next token.

As the Decoder should be able to generate a full, tokenized movie review, it has an additional function to generate the final movie review sequence. To begin the sequence, the function receives the [START] token, which is then passed to the Decoder's Embedding and LSTM layers. By calculating the $argmax$ over the final Dense Layer's output vector, it receives the most probable next token which is appended to the sequence. The function then loops over the sequence input, choosing the next most probable word to append to the sequence, until either the sequence length or the [END] token is reached.

5.2 GAN Component

As a result of adding the Variational Autoencoder to the architecture, the GAN must not be modified to accommodate discrete spaces. However, the aim of the GAN has now changed. If before the Generator aimed to construct (tokenized) movie reviews and the Discriminator aimed to rate the realness of a given input movie review, they must now generate and score real-valued latent representations, i.e. the encoding of such a review.

Generator

The Generator is realized with a Dense Layer, followed by 40 Res Blocks (*s.b.a.*) and finally a second Dense Layer. The first Dense layer has no activation because it simply reshapes the random noise input to match the input shape of the following Res Blocks. Each Res Block is comprised of a Dense Layer with ReLU activation, followed by Dropout to alleviate mode collapse, and another Dense Layer without an activation function (*s.b.a.*). To avoid vanishing gradients, the input to the Block and the output of the latter Dense Layer are added together. The final layer of the Generator resizes the output of the last Res Block to match the state size of the Decoder, using sigmoid activation to ensure a viable output range. This output is the generated artificial latent encoding, which can now be passed either to the Discriminator for scoring, or to the Decoder for translation into tokens.

Discriminator

The GAN's Discriminator component roughly follows the architecture already described for the Generator: a Dense Layer followed by 40 Res Blocks and another Dense Layer. All layers must implement weight clipping to ensure the best possible performance of the Wasserstein loss. The main difference between the two networks is a consequence of our task definition. The Discriminator must not be able to generate an encoding, but must rather rate its input encoding according to its plausibility as stemming from a real sample. Furthermore, the Discriminator must predict whether the given encoding is of positive or negative sentiment. Thus, it must have two outputs in its final Dense Layer, each scoring one of the two tasks.

5.3 Training Procedure

Our model's training procedure conforms to that of regular LaTextGAN models and is split into two separate training stages. During the first stage, solely the VAE is trained using Adam as an optimizer *s.b.a.* with a learning rate $\alpha = 0.001$ and using Sparse Categorical Crossentropy as a loss function. The training samples are first passed to the Encoder component. The Encoder's output is utilized as the initial hidden state of the Decoder, with the target being the same as the input sample. The VAE is trained for 5 epochs.

Once the VAE has been fully trained, its Encoder can be utilized to train the GAN in the second stage of training. The Generator and Discriminator trained separately from one another, with the Generator receiving as input a random noise vector, sampled from the Gaussian Distribution. The generated artificial encoding is then passed to the Discriminator, which scores the plausibility of the encoding, as well as predicts the sentiment. In a second step, the Discriminator is passed a real training example and outputs a second score and prediction. For both components, the losses are separately calculated by adding up the loss of the plausibility score and the Mean Squared Error of the sentiment target. We chose RMSProp with learning rate of $alpha = 0.0005$ as an optimizer, instead of the Adam optimizer used in [4]. This is due to Adam sometimes destabilizing WGAN training, while RMSProp, which does not utilize momentum, does not have the same effect [2]. Following the training procedure suggested in [4], the Discriminator is trained multiple times per epoch, while the Generator is simply trained once.

6 Results

After successful training results can be generated by calling the VAE Decoder's *generate_text* function. The model is only able to generate a singular output, which varies neither with different sentiment parameters nor with different inputs

to the Generator. The generated sample displays the following properties:

i. The first word in the text is the [START] token.

ii. Besides the [START] token, all words are lower-case and the text does not contain any special characters.

iii. The text exhibits topic words, such as "movie" and "dvd", as well as sentiment expressions, such as "not disappointed" and "fan".

iv. The sub-segments are grammatical, and, if one were to manually determine the end of a sentence, the full text would also be grammatical.

v. No [END] token was generated.

vi. After a few starting words, the text begins to repeat a sub-segment, until the maximum sentence length is reached.

> "[START] i saw this movie on dvd and i was expecting to see it i was expecting to see the movie i was expecting to see the movie i was not disappointed i had to say that i was a fan of the movie i was not disappointed i had to say that i was a fan of the movie i was not disappointed i had ...] "

Figure 4: Example review generated by the model using variable lengths.

> "[START] i have seen this movie and i have to say that i was a fan of the movie i was a little bit of the movie i was a bit of the movie i was a bit of the movie i was a bit of the movie i was a bit of the movie i was a bit of the movie [...]"

Figure 5: Example review after repeated training.

6.1 Discussion

Properties i. and ii. directly result from the chosen pre-processing steps. The [START] token was prepended to each training sample to denote the beginning of a review and is later passed as a starting input to the *generate_text* function. Due to all special characters having been removed from the training samples, the model cannot generate such.

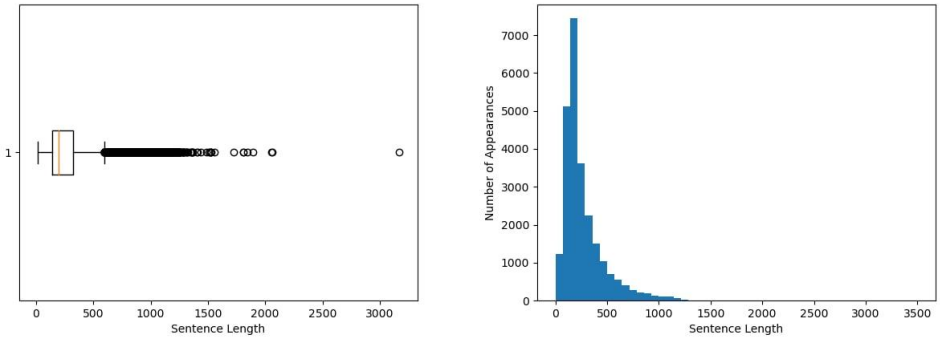

Figure 6: Variance (left) and distribution (right) of sentence lengths in the dataset.

When examining the generated review (properties v. and vi.), it seems as though the model has not learned how to end a review. The failure to produce the [END] token might be attributed to several things. Since review lengths included in the dataset exhibit a mean of $\mu = 267.3$, the generated sample may simply be too short to reach a learned [END] token. However, after increasing the maximum sentence length to 2000 words, the model still did not generate an [END]. This might suggest that the model has run into an endless repetition of the aforementioned sub-segments. However, as the sentence lengths show a variance of $\sigma^2 = 41421.8$, and a standard deviation of $\sigma = 203.5$, it is likely that the high variability of sentence lengths impedes the modeling of the end of a review. To test the latter hypothesis, we trimmed each training review to a maximum length of 40, ensuring uniformity and giving more weight to the [END] token. After re-training both VAE and GAN on the dataset with truncated input lengths, a [END] token was generated after 34 tokens. Consequently, the architecture is capable of learning when to [END] a text, as long as the length of a review is an explicit feature of the data. When having to model the ending of a text from implicit, contextual information, which is the case for movie reviews, where the length depends strongly on contextual factors and the will of the author, the architecture might need to be modified. Thus it may have been more suited to the task to choose a dataset with a limited text length, such as tweets or single-sentence reviews, to actively increase the weight of the [END] token or to limit the maximum length during pre-processing.

Even when truncating the lengths of the samples, however, the generated output runs into repetitions of sub-sentences. While grammatical and topic-aligned, these sub-sentences and especially their repetition causes the review to be incoherent and

"[START] i have seen this movie i was a lot of the movie i was a lot of the movie i was a lot of the movie i was a lot of the movie [END]"

Figure 7: Example review generated after training with truncated lengths.

reveals what could be an implementation error, as the sentences in [4] did not exhibit these cycles. The subject words that constitute these segments are expected to be among the most common for this domain ("i, "movie", "fan", "disappointed"). As the generated sentences do not deviate from each other at all, and the VAE still exhibits a loss of roughly 4 after training, we suspect both issues to stem from the VAE being too weak or overfitting to the most common words, i.e. those most likely to correct if predicted.

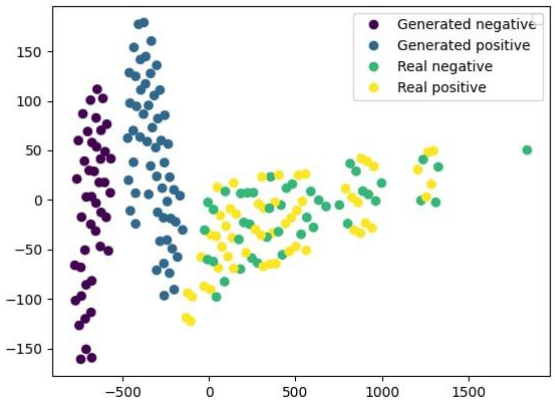

Figure 8: t-SNE visualisation of the artificial and real latent spaces.

Lastly, the Discriminator's loss regarding sentiment classification explicates that the model was unable to learn the difference between positive and negative sentiments. Even if it produced different generations, it is unlikely that they would strongly align with the desired sentiment. Perhaps this is due to the summation of Wasserstein loss and classification loss, where the latter receives less weight. However, without resolving the aforementioned problems, this hypothesis can't be tested.

7 Conclusion

Despite many undesired properties of the generated sentences, the results are somewhat successful. The topic and sentiment words generated ("movie", "dvd", "disappointed") capture the essence of both the film domain and the polarity of a review. When reducing the artificial review to its first few words it would be plausible to expect this to be a genuine review, written by a user on IMDB. As the text uses words found in most of the sample reviews and thus seems rather generic, i.e. not including specificities of a movie, such as names, it shows that the model can generalize. A further accomplishment is the model's ability to generate complete, grammatically correct sub-sentences, thus modeling those underlying relations at least to an extent.

References

[1] Tensorflow datasets imdb reviews documentation. `https://www.tensorflow.org/datasets/catalog/imdb_reviews`.

[2] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR, 2017.

[3] Akshay Budhkar, Krishnapriya Vishnubhotla, Safwan Hossain, and Frank Rudzicz. Generative adversarial networks for text using word2vec intermediaries. *arXiv preprint arXiv:1904.02293*, 2019.

[4] David Donahue and Anna Rumshisky. Adversarial text generation without reinforcement learning. *arXiv preprint arXiv:1810.06640*, 2018.

[5] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[6] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

[7] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[8] Achraf Oussidi and Azeddine Elhassouny. Deep generative models: Survey. In *2018 International Conference on Intelligent Systems and Computer Vision (ISCV)*, pages 1–8. IEEE, 2018.

[9] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. *Advances in neural information processing systems*, 29, 2016.

[10] Lifeng Shang, Zhengdong Lu, and Hang Li. Neural responding machine for short-text conversation. *arXiv preprint arXiv:1503.02364*, 2015.

[11] Guy Tevet, Gavriel Habib, Vered Shwartz, and Jonathan Berant. Evaluating text gans as language models. *arXiv preprint arXiv:1810.12686*, 2018.

[12] Zesen Wang. Generative adversarial networks in text generation, 2019.

[13] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. Seqgan: Sequence generative adversarial nets with policy gradient. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017.