Generative Adversarial Networks and Word Embeddings for Natural Language Generation

Robert D. Schultz Jr
The Graduate Center, City University of New York

GENERATIVE ADVERSARIAL NETWORKS AND WORD EMBEDDINGS FOR NATURAL LANGUAGE GENERATION

by

ROBERT D SCHULTZ

A master's thesis submitted to the Graduate Faculty in Linguistics in partial fulfillment of the requirements for the degree of Master of Arts, The City University of New York

2019

Generative Adversarial Networks and Word Embeddings for Natural Language Generation
by

Robert D. Schultz

This manuscript has been read and accepted for the Graduate Faculty in Linguistics in satisfaction of the thesis requirement for the degree of Master of Arts.

Date	William Sakas
	Thesis Advisor

Date	Gita Martohardjono
	Executive Officer

THE CITY UNIVERSITY OF NEW YORK

ABSTRACT

Generative Adversarial Networks and Word Embeddings for Natural Language Generation

by

Robert Schultz

Advisor: William Sakas

We explore using image generation techniques to generate natural language. Generative Adversarial Networks (GANs), normally used for image generation, were used for this task. To avoid using discrete data such as one-hot encoded vectors, with dimensions corresponding to vocabulary size, we instead use word embeddings as training data. The main motivation for this is the fact that a sentence translated into a sequence of word embeddings (a "word matrix") is an analogue to a matrix of pixel values in an image. These word matrices can then be used to train a generative adversarial model. The output of the model's generator are word matrices which can then be translated back into sentences using closest cosine similarity. Four models were designed and trained including two Deep Convolutional Generative Adversarial Networks (DCGAN) using this method. Mode collapse was a common problem encountered, along with generally ungrammatical outputs. However, by using Wasserstein GANs with gradient penalty (WGAN-GP) we were able to successfully train models with no mode collapse, whose generator outputs were reasonably well-formed. Model generators' outputs were evaluated by well-formedness using a pretrained BERT language model, and by uniqueness using an inter-sample BLEU score. Both WGAN-GP models trained performed well in these two metrics.

All models were constructed and trained using PyTorch, a machine learning library for Python. All code used in the experiments can be found at https://github.com/robert-d-schultz/gan-word-embedding.

ACKNOWLEDGMENTS

I’d like to thank my parents for financial and morale support over the last two and half years.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. Introduction

Natural language generation is a natural language processing task in which data is transformed into human-readable language. Natural language generation is used for automated question answering, generating summaries of large documents, and generating readable content. These tasks have real world applications in journalism, customer service, and data analysis.

Much has been done with neural networks and natural language generation, but it hasn't been until recently that Generative Adversarial Networks (GANs) have been applied to the task. GANs are an architecture of neural network with a discriminator and generator network. By having these two sub-neural networks compete against each other, the model is able to learn the distribution of training data and generate new examples. GANs generally do not perform well on discrete data such as the words of natural language. A way around this is using word embeddings. Word embeddings are the result of embedding words into a vector space. Using word embeddings with GANs have the advantage of reduced dimensionality over using one-hot encoded vectors the size of the training data's vocabulary. Word embeddings also have semantic information encoded within them, which should result in better trained and more generalized models.

In this thesis we outline a technique of applying word embeddings to generative adversarial networks. First, we will define generative adversarial networks and word embeddings in detail. Then review the literature involving natural language generation and neural networks. We also look to the well-explored task of image generation using deep convolutional generative adversarial networks for inspiration. Images are simply grids of numbers representing individual pixel values, by translating natural language into word vectors, we are left with a similar result. We then explain the methodology used for the new experiments, including the dataset used, the model architectures, and any model hyperparameters. We outline four unconditional text generation experiments: two using conventional deep convolutional GANs, one experiment using a 2d Wasserstein GAN, and one using a 1d Wasserstein GAN. For each experiment we present training statistics and output examples. We discuss the performance of each model, including its evaluation with a pretrained English-language model.

2. Background

2.1. Generative Adversarial Networks

Generative Adversarial Networks (GANs) were first introduced by Goodfellow et al. in 2014. GANs are composed of two neural networks called the generator and discriminator. The discriminator is fed both real and fake training samples and tries to tell the difference between the two classes. The generator tries to fool the discriminator by generating those fake training samples. If done properly, the result of training the model is a generator that generates fake samples that are indistinguishable from real ones and a discriminator that has resorted to guessing between the two classes.

GAN models are generally trained as follows: First the generator is trained. It generates fake data, which is put through the discriminator. Loss is then calculated, and backpropagation is applied to the generator. This can be thought of as the discriminator giving feedback to improve the generator. The next step is training the discriminator. Batches of fake data created from the generator, as well as real samples from the training set, are fed through the discriminator and the discriminator classifies them. Backpropagation is then applied to the discriminator. Training of the full model follows a loop of these two steps: train the generator and then train the discriminator for one or more batches.

2.2. Deep Convolutional GAN

Radford (2015) outlines deep convolutional generative adversarial networks (DC-GAN) which is one of the most popular architectures for image generation. The discriminator and generator of a DC-GAN are comprised of transposed convolutional layers. Generators under a DC-GAN architecture have their channels progressively reduced and their X and Y dimensions progressively expanded from layer to layer. The discriminator has the opposite effect, X and Y dimensions are reduced and channels are expanded. The output of the generator and the input of the discriminator correspond to images, with 1 or 3 channels, depending on if the images are black and white or in color. Radford recommends replacing pooling layers with strided and fractional strided convolutions. Their model uses batch normalization in both the generator and discriminator and does not use any fully connected linear layers. For non-linearities, they use ReLU activation in the generator, with hyperbolic tangent on the output, and LeakyReLU in the discriminator. LeakyReLU is defined as:

$$LeakyRelu(x) = \begin{cases} x, & x \geq 0 \\ neg\_slope * x, & otherwise \end{cases}$$

Where *neg_slope* is a negative slope hyperparameter. Radford used a negative slope of 0.2 in their models.

2.3. Wasserstein GAN

Wasserstein Generative Adversarial Networks (WGAN) are a variation of GAN (Arjovsky, 2017). WGAN have a different cost function that uses Wasserstein distance. Wasserstein distance is also known as the earth mover’s distance and can be thought of as the minimum cost of turning one pile of dirt into another. The dual form of the Wasserstein metric is formally defined as:

$$W(\mathbb{P}_r, \mathbb{P}_\theta) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x \sim \mathbb{P}_r}[f(x)] - \mathbb{E}_{x \sim \mathbb{P}_\theta}[f(x)]$$

Where $\mathbb{P}_r$ is a fixed distribution, $\mathbb{P}_\theta$ is a distribution, $\sup$ is the least upper bound, $\mathbb{E}$ is the expected value, and f is 1-Lipschitz function such that $|f(x_1) - f(x_2)| \leq |x_1 - x_2|$.

The cost function (value function) of the discriminator is derived from this metric and can be expressed as:

$$g_w = \nabla_w \frac{1}{m} \sum_{i=1}^{m} \left[f\left(x^i\right) - f(G(z^i)\right]$$

Where g_w is the gradient, m is the batch size, and f is a 1-Lipschitz function. WGANs clip the weights of the discriminator so that they are always between -*c* and *c*, where *c* is a hyperparameter of the model. This is because f must be a 1-Lipschitz function.

Because of this new value function, WGANs have a much smoother gradient and can learn even if the generator is underperforming the discriminator. Another massive benefit is that there is virtually no mode collapse.

A drawback of have weight clipping is it makes the model sensitive to the *c* hyperparameter's value. Wasserstein GAN with gradient penalty (WGAN-GP) (Gulrajani, 2017) is a further evolution of WGAN that seeks to address this problem. WGAN-GP uses a gradient penalty instead of weight clipping, so there is no hyperparameter *c* to tune. The gradient penalty function gives a penalty to the model is the norm of the gradient is different from 1, its target value. This enforces the 1-Lipschitz requirement $|f(x_1) - f(x_2)| \leq |x_1 - x_2|$ of the original WGAN. With this penalty included, the new objective function is defined as:

$$L = \mathbb{E}_{\tilde{x} \sim \mathbb{P}_g}[D(\tilde{x})] - \mathbb{E}_{x \sim \mathbb{P}_r}[D(x)] + \lambda\, \mathbb{E}_{\hat{x} \sim \mathbb{P}_{\hat{x}}}[(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1)^2]$$

Where $\tilde{x} \sim \mathbb{P}_{\tilde{x}}$ represents random samples sampled uniformly along straight paths between points sampled from the training data distribution $\mathbb{P}_r$ and the generator distribution $\mathbb{P}_g$. λ is the penalty coefficient. A λ of 10 was found by Gulrajani to work well across many architectures, datasets, and tasks.

2.4. Word Embedding

Word embedding is a technique in which words are mapped into a vector space. Word embeddings are also referred to as word vectors. There are several models with their own corresponding training algorithms that can be used to produce word embeddings. These include Word2vec (Mikolov, 2013), fastText (Joulin, 2016), and GloVe (Pennington et al., 2014). Pretrained word embeddings, trained on billions of words, are available for all three of these algorithms. The GloVe word embedding algorithm in particular uses a model:

$$J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij})^2$$

Where w_i^T are word vectors, $\tilde{w}_j$ are context word vectors, b_i and $\tilde{b}_j$ are biases, V is the vocabulary size, and f is a weighting function. The weighting function Pennington et al. chose to use is:

$$f(x) = \begin{cases} (x/x_{max})^{\alpha}, & x < x_{max} \\ 1, & otherwise \end{cases}$$

Where x_{max} is a cutoff chosen to be 100 and α is a parameter chosen empirically as 0.75. Using this GloVe model, a vector for each word in the vocabulary is learned from large amounts of training data.

Correctly trained word embeddings encode semantic relations. Word embeddings are usually evaluated on word analogy tasks that check that linear transformations applied to vectors results in analogies such as man is to king as woman is to queen. This property of word embeddings might make it easier for neural networks to see patterns in sentences that using one-hot encodings could never achieve. This might lead to better generalization of models trained using this technique, where the models avoid overfitting.

3. Literature Review

Notable attempts at applying GANs to natural language include *Adversarial Generation of Natural Language* (Rajeswar et al, 2017). They used WGAN and WGAN-GPs with convolution and long short-term memory (LSTM) to generate word-level and character-level language. For the word-level generation they used sequences of probability distributions over the entire vocabulary for generated data and sequences of one-hot vectors for real data. Because of this, the vocabulary size was constrained to the 30,000 most frequent words for all experiments they conducted. For evaluation of their word-level generation models, they used a probabilistic context-free grammar created from the Penn Treekbank corpus to check for grammaticality of the generator outputs. Their convolutional WGAN-GP model achieved a 98.59% accuracy under this evaluation metric.

Xu et al. (2018) attempt to increase variation in natural language-producing GAN by introducing a diversity-promoting generative adversarial network (DP-GAN). These DP-GAN models reward new, fluent text while not rewarding repeated text. The reward function in this architecture has two parts: sentence-level and word-level. The sentence-level reward function is given as $R(y_t) = -\frac{1}{K}\sum_{k=1}^{K} \log D_{\varphi}\left(y_t | y_{t,<k}\right)$ where y_t is a sentence of K words and D_{φ} is a binary classifier that decides how likely y_t is from the training data. The word-level rward function is given as $R\left(y_{t,k} | y_{t,<k}\right) = -\log D_{\varphi}\left(y_t | y_{t,<k}\right)$. They use DP-GANs in review generation and dialogue generation tasks. There final models were human-evaluated and compared with other architecture such as SeqGAN (Yu, 2017) and a sequence-to sequence

model PG-BLEU (Bahdanau, 2016). As judged by human evaluators, DP-GANs outperformed competing models in all task in relevance and diversity.

Kim (2014) shows that convolutional neural networks can achieve good results in classifying sequences of word embeddings. Classification tasks performed include classifying by sentiment, subjectivity, question type, and opinion polarity. They used models with convolutional layers to achieve good performance in these tasks. The convolutional layers of discriminator in a DC-GAN model that was trained on word vectors would perform similarly. The filters of the convolutional layers would look for patterns in the word vectors to correctly classify the sentence. However, the classification task would instead be classifying sequences of word embeddings by well-formedness. Kim shows that concatenated word vectors have local features that convolutional can "pick up" on. GANs trained to generate images use local features, like eyes, noses, and mouths, in their convolutional layers. This leads me to believe that applying GANs to word embeddings should result in well-trained models.

4. Data

We used pretrained 50-dimensional GloVe vectors for the experiments. These were the smallest pretrained vectors out of the three discussed in the background section. Another advantage is that the pretrained GloVe vector vocabulary included stops words such as "is" and "the", allowing for grammatical outputs that would not have been possible with the Word2vec ones. The pretrained Facebook fastText vectors were ruled out because their vocabulary is segmented into pseudo-morphemes. Any outputs of a GAN's generator would have to be first translated into these pseudo-morphemes, using minimum cosine distance, and then recombined into English words. This recombination step would have been overly complicated.

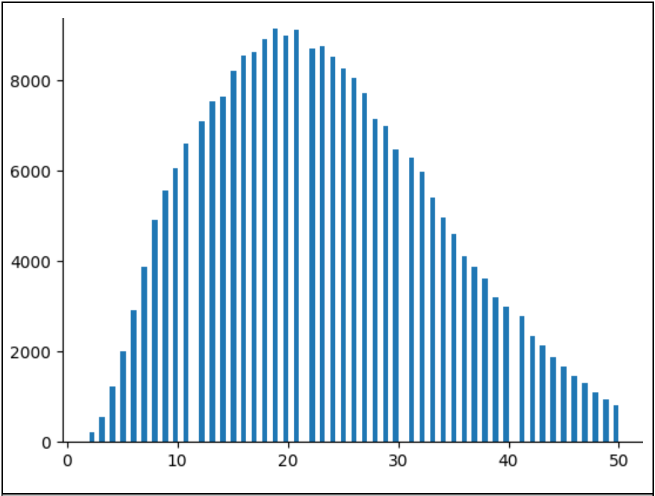

Figure 1: Histogram of Training Data Token Length

For training data, a subset of News Crawl 2009 was used. This corpus is comprised of English-language sentences from newswire data. Each sentence was first tokenized using the Punkt Sentence Tokenizer (Kiss, 2006) and was case folded but not lemmatized. Sentences that had more tokens than the maximum sequence length of 50 were rejected. Sentences were then translated into word vectors and then concatenated into arrays. If a token was not in the GloVe word vector vocabulary, then the entire sentence was rejected. If a sentence had less than the maximum sequence length, then it was padded with 0-vectors. The result was a set of arrays each with 2500 floats arranged into a 50x50 matrix.

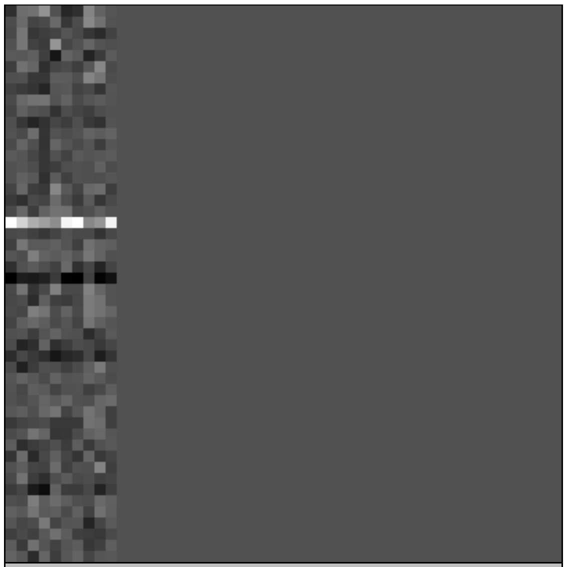

Figure 2: Word Matrix as a Greyscale Image

There are 250,000 samples in the training set. The average sentence (token) length is 23.23 with a standard deviation of 10.33. Figure 1 shows the distribution of sentence lengths.

These “word matrices” that make up the training set can be thought of as greyscale images. This was an important conceptual motivation for starting this project. Figure 2 shows the sentence “The quick brown fox jumps over the lazy dog.” translated into GloVe vectors, and then concatenated. Pixel values were scaled to fit a 0 to 255 range for this image. The uniform right side of the image represents the 0-vectors that pad the matrix to its final 50 by 50 dimensions.

5. Methodology

All models were trained on the dataset discussed above. For each model a sample of outputs was taken from their generator. The outputs were translated back into English-language tokens and an evaluation was conducted.

5.1. Word Embedding to Word

To convert from a GAN's generator's output back into word tokens, we use closest cosine similarity. Each column of the generator's output is taken as a word vector and is replaced with the token whose word vector in the GloVe dictionary has the smallest cosine distance.
Cosine distance measures the cosine of the angle between two non-zero vectors and is defined for two vectors A and B as:

$$\frac{A \cdot B}{||A||||B||}$$

Columns of the generator's output that had sufficiently small magnitude were simply replaced with a "[pad]" token, which has been removed from the examples given below. This small magnitude was decided on experimentally, with a value of 0.25 giving the best results.

5.2. Evaluation

All models' generators were evaluated on a pre-trained Bidirectional Encoder Representations from Transformers (BERT) language model (Devlin, 2018). A sample of 100 outputs was generated from each model's generator for evaluation. Each generated sequence of word vectors was converted into word tokens by the method described in the previous section. Pad tokens were removed for this step. Perplexity for each sentence was calculated by the BERT model. For each model, perplexity was averaged over all sentences in the sample. BERT's average on well-formed English-language sentences is around 30. An average score close to this means the generator is producing well-formed outputs, while a higher score corresponds to the generator producing ungrammatical sentences or even gibberish.

With this evaluation system it would be possible for a generator produce the exact same grammatical sentence and fool this metric. This might happen if the model experiences mode collapse. To prevent this, uniqueness was also measured for the outputs using an inter-sample BLEU score. BLEU score was computed between each sentence in the 100 sentence samples. The scores were then averaged. An average BLEU score close to 0 shows high uniqueness in a model's generator outputs.

6. Experiments

Four main experiments were conducted:

- Train and evaluate a model that has a simple DC-GAN architecture, very similar to what image-producing GANs use. The X dimension corresponds to different tokens in the sentence, while the Y dimension corresponds to the length of the word vectors being used, 50 for GloVe. 2D convolution is used in this model.
- A variation on the above model, changing around the architecture to mimic the syntactic structure of language. This DC-GAN model was much deeper than the first one.
- Train and evaluate a WGAN-GP but use 1D convolution. Channels correspond to word vector length.
- Train and evaluate model that has a 2D WGAN-GP architecture, again emulating image-producing GANs.

6.1. DCGAN Experiment 1

6.1.1. Model Specifics

A general DC-GAN architecture was adopted for this first experiment. This model's architecture was designed to emulate image-generating GANs.

The generator was comprised of 4 main transposed convolutional layers. Batch normalization layers were added between each, with an epsilon of 1×10^{-5} and a momentum of 0.1. Batch normalization is calculated as:

$$y = \frac{x - mean(x)}{\sqrt{Var(x) + \epsilon}} * \gamma + \beta$$

Where gamma and beta are learnable parameters. LeakyReLU with a negative slop parameter of 0.01 was used as an activation function on the output of each batch normalization layer.

The discriminator was made up of 3 convolutional layers, again with batch normalization and LeakyReLU. Hyperbolic tangent was used on the input of the discriminator. The final value was then put through the Sigmoid function described as:

$$Sigmoid(x) = \frac{1}{1 + e^{-x}}$$

With 4 convolutional layers used in the generator and only 3 in the discriminator, the discriminator was notably weaker. One notable difference from a generic image-based DC-GAN is the lack of a non-linearity function on the generator's output. Instead it has been moved to the input of the discriminator. This is because GloVe embeddings are not constrained like image pixels are.

The model was trained for 4 epochs over the 250,000 training samples. Each batch contained 100 training samples. The latent vector Z was 100 dimensional, and random samplings of Z space was done using normal distributions. We used a learning rate of 0.0002 and Adam optimizer (Kingma, 2014) with a beta1 of 0.5, beta2 of 0.999, and an epsilon value of 1×10^{-8}. Binary cross entropy (BCE) loss, which measures BCE between the outputs and label, was used as a criterion. BCE loss is described as:

$$l(x, y) = \{l_1, .. l_N\}, \quad l_n = -w_n[y_n * \log x_n + (1 - y_n) * \log(1 - x_n)]$$

Where N is the batch size.

6.1.2. Results and Generator Output

Generator and discriminator losses are displayed in Figure 3. See Table 1 for examples of generated sentences. This model experienced mode collapse, so the examples are from a training iteration before that happened. Padding has been removed from the ends of each sentence.

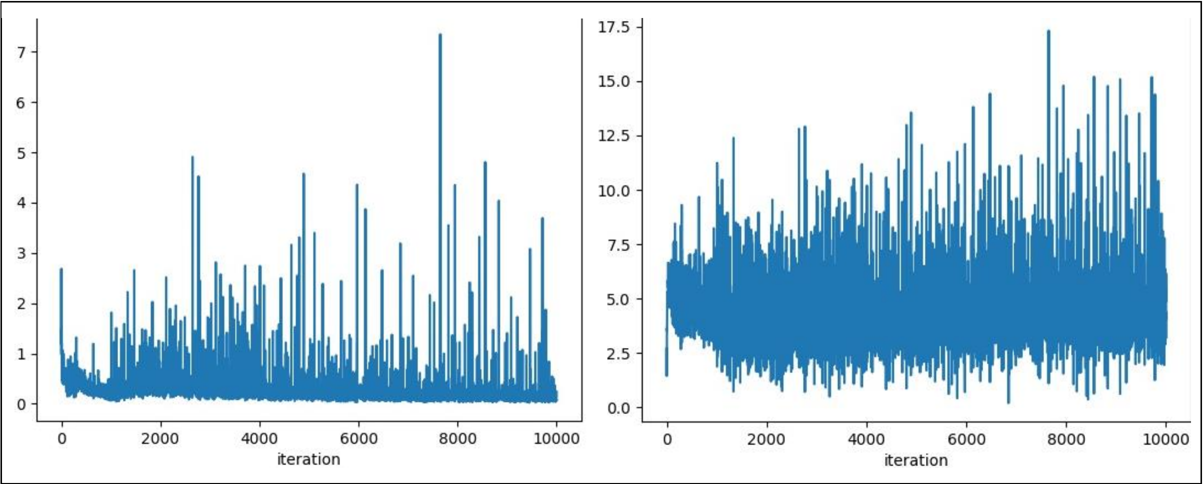

Figure 3: Discriminator (Left) and Generator (Right) Losses of DCGAN Experiment 1

A sample of 100 generator outputs were evaluated by a pretrained BERT language model. This resulted in an average perplexity score of 23,032. As a measure of uniqueness, the inter-sample BLEU score was also computed as 0.0565. A summary of this evaluation can be seen in Table 5.

Table 1: Example Generator Output of DCGAN Experiment 1

1	liked did shot just ` against curse compare when mr. bidwill knows quit washington signed prosecutor theirs big jersey to because letting 's disappearing although precious lockout ; mdn
2	admired did shot alone wise face knowing thought apparently mr. successor berg retire came united 's looked like be scheme trend investment leaves it because to it chair
3	lincoln 2000 pieces oldest supposedly once . tunya
4	lincoln madonna pieces oldest supposedly once .
5	liked did shot just luck against knowing someone apparently mr. successor knows decides move united head never like considered to because failing green look be give split .

6.1.3. Discussion

Judging from the outputs, this model did not train very well. Figure 3 shows a very low discriminator cost with a relatively high generator cost. This points to the discriminator being too “good”, spotting the fake samples the generator produces so easily that it is not able to give good feedback to help improve the generator. Indeed, this model suffered from partial mode collapse. As can be seen in Table 1, the phrases “liked did shot just” and “lincoln” were very common in sample generator outputs. The inter-sample BLEU score appears promising, but when compared to the later experiments, it is relatively poor. The extremely high average perplexity of the evaluation samples shows that the outputs are not well-formed. Perplexity of grammatical English sentences for the BERT model is around 30.

6.2. DCGAN Experiment 2

6.2.1. Model Specifics

For the second experiment, a DCGAN with a different architecture was trained. In the first DCGAN experiment, the generator’s dimensions slowly and uniformly expand through transposed 2d convolution. In this experiment the generator was setup so that the initial z-space vector expands rapidly in the word-vector dimension and then slowly expands in the sequence length dimension. The last few layers of the generator expand by only 1, mimicking a binary tree-like structure. This model the principle of compositionality of language, where sentences are composed of constituent parts. Binary trees are commonly used semantic and syntactic analysis in linguistics.

The generator was made up of 11 transposed convolutional layers. The first layer expanded the initial latent vector in the word vector dimension up to 1x50 dimensions. Subsequent layers used 50-dimensional kernels to slowly expand in the sequence-length dimension. The last 5 layers expanded the sequence length dimension by 1, mimicking a binary tree. Two-dimensional batch normalization as well as LeakyReLU were used for generator layers.

The discriminator was made up of 5 convolutional layers, again with batch normalization and LeakyReLU. Hyperbolic tangent was used on the input of the discriminator. The final value was then put through the Sigmoid function. The system of convolution was not like the generator, and instead expanded slowly in both dimensions uniformly, like in the first experiment.

The model was trained for only a single epoch, over the 250,000 training samples. Each batch contained 100 training samples. The latent vector Z was 100 dimensional, and random samplings of Z space was done using normal distributions. And once again we used a learning rate of 0.0002 and Adam optimizer with a beta1 of 0.5, beta2 of 0.999, and an epsilon value of 1×10^{-8}. Binary cross entropy (BCE) loss was used as well.

6.2.2. Results and Generator Output

Generator and discriminator costs are displayed in Figure 3. See Table 2 for examples of generated sentences. This model experienced mode collapse, so the examples are from training iteration 2000, just before that happened.

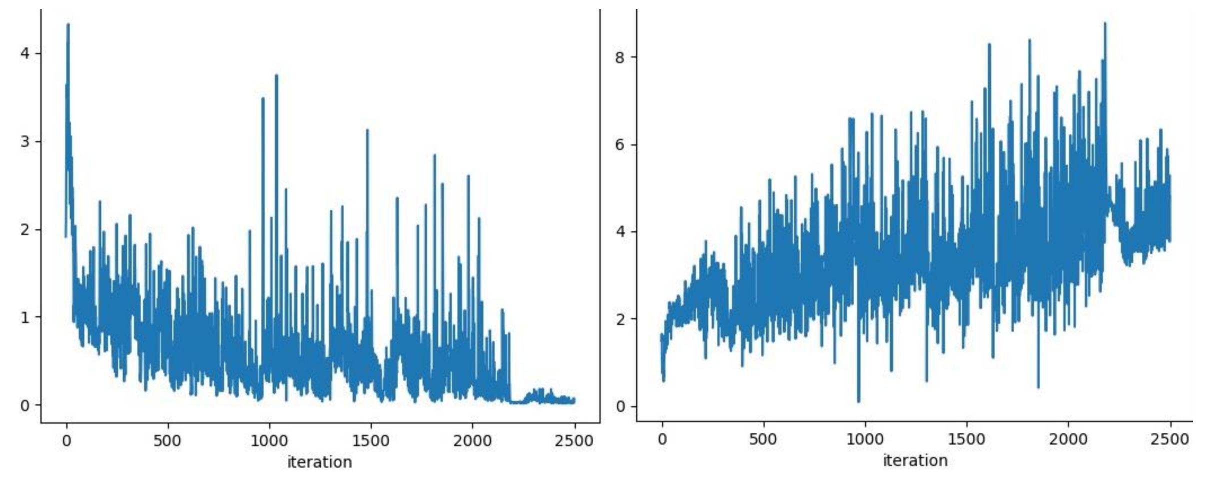

Figure 3: Discriminator (Left) and Generator (Right) Losses of DCGAN Experiment 2

A sample of 100 generator outputs from iteration 2000 were evaluated by a pretrained BERT language model. This resulted in a score of 13,110. As a measure of uniqueness, the inter-sample BLEU score was also computed as 0.7207. A summary of this evaluation can be seen in Table 5.

Table 2: Example Generator Output of DCGAN Experiment 2

1	thought fact never never once nowhere but still but as non . went arw . but who an comparison on copenhagen badly but .
2	thought fact never never once nowhere but still turned as inclusion to held arw . still once although pre . closing area now but .
3	thought still never never once gone even . turned same non on opened 180 . jeanne but . stoesz .
4	same has him why still gone way everywhere but .
5	survey is gave stealth demographics time its nisoor still .

6.2.3. Discussion

This model did not train properly. As can be seen in Figure 3 the model had a promising start to training with generator loss being relatively low compared to the first experiment. However, by iteration 2200 the model experienced complete mode collapse, just like the DC-GAN model of Experiment 1. At this iteration the discriminator loss dropped to a consistent near-zero value and the generator loss experienced a drop as well. The samples in Table 2 are from iteration 2000, just before the mode collapse. The first three show signs of it with "thought fact never never once nowhere but still" appearing twice. The inter-sample BLEU score of 0.7207 calculated as this iteration shows that the generator was repeating itself quite a bit. The average perplexity of 13,110 shows that the samples were very ungrammatical, although less so than that of Experiment 1.

6.3. WGAN-GP Experiment 1

6.3.1. Model Specifics

For this experiment, in the hope of avoiding mode collapse like the DC-GAN ones, a WGAN architecture was used. In particular, the WGAN with gradient clipping architecture proposed by Gulrajani (2017). Residual blocks were also used in this network. Residual blocks are a component of residual neural networks, proposed by He et al., 2016. As we have seen from the last experiment, deep neural networks are difficult to train properly. Residual neural networks allow for these deeper networks to be trained with their residual blocks, where a skip connection or "shortcut" allows later blocks to reuse an early block's activation and avoid the vanishing gradient problem.

The generator was built from a linear layer that expands the latent vector to 512 dimensions. The next five layers are "Residual block" layers with dimensionality 512. Each residual block is made up of two 1d convolutional layers with ReLU activations. The output of a residual block and the input multiplied by a constant 0.3 are added together as the final output. After five of the residual blocks, the output goes into final 1d convolutional layer and then is output from the generator.

The discriminator is the inverse of the generator. The input first goes through the tanh function, then an initial 1d convolutional layer, then 5 residual blocks, exactly as described above. The output of the residual blocks is then reshaped into a line and put through a linear layer, reducing it to a single value which is of course used for classification.

The model was trained for 10,000 iterations. There were 5 critic iterations for every 1 generator iteration. Each batch contained 64 training samples. The latent vector Z was 128 dimensional, and random samplings of Z space was done using normal distributions. A slower learning rate of 0.0001 was used for this experiment. The Adam optimizer with a beta1 of 0.5, beta2 of 0.999, and an epsilon value of 1×10^{-8} was used. The WGAN-GP loss described in the Background section was used, which is a combination of WGAN loss with a gradient penalty. Following Gulrajani 2017's advice, a gradient penalty lambda of 10 was used.

6.3.2. Results and Generator Output

Generator and discriminator (critic) costs are displayed in Figure 4. See Table 3 for examples of generated sentences.

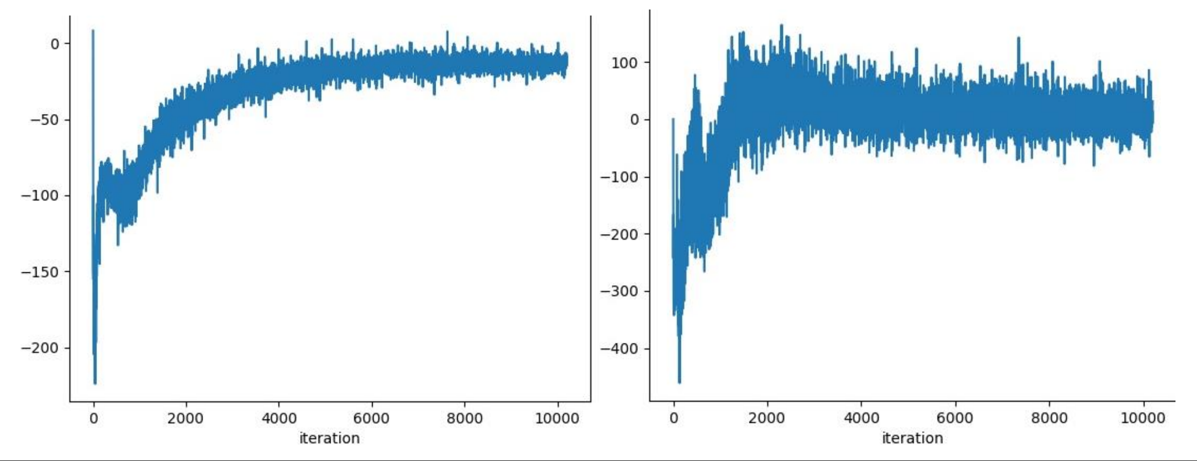

Figure 4: Discriminator (Left) and Generator (Right) Losses of WGAN Experiment 1

A sample of 100 generator outputs were evaluated by a pretrained BERT language model. This resulted in a score of 2,520. As a measure of uniqueness, the inter-sample BLEU score was also computed as 0.0089. A summary of this evaluation can be seen in Table 5.

Table 3: Example Generator Output of WGAN Experiment 1

1	says 202-887-8316 also famous redeemed others all upon was well same death among , time night . . well
2	for good but as well . mowhoush well this man . and with full opening is man gruden added far mueller under but again while . once well march . wiens
3	all that participants but just counted unfortunately right adding . making america have same actually indeed as that any finally also apparently when christie no-confidence
4	months rest any ago co responded mother before did which turned on with another turned to . the apart as face a which as by metz havelock
5	just take time instead psydrinae gadsden one , it ages telling

6.3.3. Discussion

As can be seen in Figure 4, the discriminator and generator losses both converged towards a stable value. This is a sign that training went well. The samples in Table 3 are slightly better formed than that of the first two experiments. The BERT language model score of 2520 confirms this. The inter-sample BLEU score of 0.0089 shows that there was high uniqueness among generator outputs. This model performed relatively well.

6.4. WGAN Experiment 2

6.4.1. Model Specifics

In this experiment, the WGAN-GP architecture was used once again but using 2-d convolution instead of 1-d like in the previous experiment. Instead of having the model's channels represent the word vector dimension, an additional dimension is introduced that makes this much more like experiment 1, where the task is comparable to image generation. Once again, residual blocks as proposed by He et al., 2016 are used.

The generator was built from a linear layer that reduces the 128 latent vector to 36. The output of this linear layer is then reshaped into a 6 by 6 square. The next four layers are "Residual block" layers. Each residual block is made up of an up-sampled convolution, followed by a final 2d convolutional layer. Each residual block also has 2d batch normalization and ReLU activations. The up-sampled convolution in the residual blocks reshapes channel space into the X and Y dimensions. The shortcut of the residual blocks is simply up-sampled convolution. After the residual blocks there is a special 2d transposed convolutional layer to arrive at the X and Y dimension of 50 by 50. The outputs of this special layer go through batch normalization, ReLU, and then one last final convolutional layer.

The discriminator (critic) is the inverse of the generator. The input first goes through the tanh function, then two 2d convolutional layers, and then 5 residual blocks. The residual blocks of the discriminator are slightly different than the ones in the generator. These residual blocks have 2d convolution and then 2d convolution with meaning pooling, alongside layer normalization and ReLU activation. Layer normalization is similar to batch normalization but does not look across the entire batch when calculating mean and variance. The output of the residual blocks is then reshaped into a line and put through a final linear layer, reducing it to a single value which is used for classification.

The model was trained for 10,000 iterations. There were 5 critic iterations for every 1 generator iteration. Each batch contained 64 training samples. The latent vector Z was 128 dimensional, and random samplings of Z space was done using normal distributions. A learning rate of 0.0001 was once again used. The Adam optimizer with a beta1 of 0.5, beta2 of 0.999, and an epsilon value of 1×10^{-8} was used. The WGAN-GP loss described in the Background section was used, which is a combination of WGAN loss with a gradient penalty. A gradient penalty lambda of 10 was used.

6.4.2. Results and Generator Output

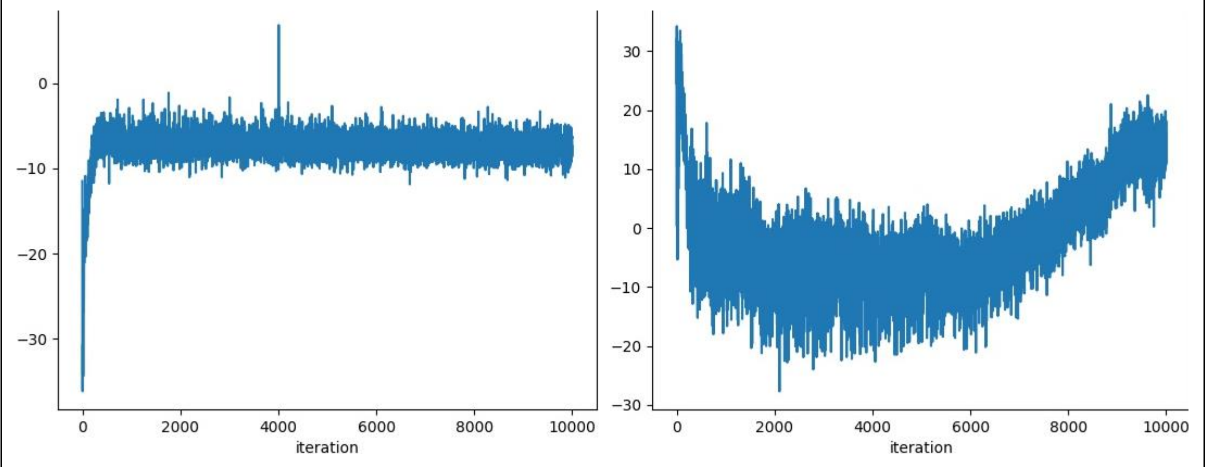

Figure 5: Discriminator (Left) and Generator (Right) Losses of WGAN Experiment 2

Generator and discriminator (critic) costs are displayed in Figure 5. See Table 4 for examples of generated sentences. A sample of 100 generator outputs were evaluated by a pretrained BERT language model. This resulted in a score of 1,653. As a measure of uniqueness, the inter-sample BLEU score was also computed as 0.0078. A summary of this evaluation can be seen in Table 5.

Table 4: Example Generator Output of WGAN Experiment 2	
1	the information . andrea as of the well skeleton . two , though thought been would be the percentage typical . anywhere . .
2	`` does did every way _ it `` .
3	even other seem rather all keep re hard .
4	adjourned had another while hkfa taking finally preliminary that time both the anchorage starting to come . once possible decision age father it told told . gladys
5	brought headed as suggested though be not to the month office its push action as the the that concerned for taking they turning all making other for carbon any that to understand the means .

6.4.3. Discussion

As can be seen in Figure 5, discriminator loss converged to a stable value. This was not true for the generator's training loss, which started increasing at around the 6000th iteration. It is possible the model started overfitting the training data, in which case the critic would know if a sample was from the generator or was a real training sample by simple memorization. More training data might be necessary to prevent this from happening in the future. As can be seen in Table 4, the generator's outputs were still of high well-formedness. In example 2, it impressively managed to generate quotation marks correctly. With a BERT score of 1,653 and an inter-sample BLEU score of 0.0078, this model performed the best in well-formedness and uniqueness out of all four experiments. This model performed relatively well.

Table 5: Summary of Generator Output Evaluations

Model	Average Perplexity	Inter-sample BLEU score
DCGAN 1	23,032	0.0565
DCGAN 2	13,110	0.7207
WGAN 1	2,520	0.0089
WGAN 2	1,653	0.0078

7. Conclusion

In this thesis I presented a technique of applying word embeddings to generative adversarial networks with the objective of generating natural language. I have shown that there are many advantages of using word embeddings over one-hot coded vectors, including reduced dimensionality and better generalization. Four models were created, trained, and evaluated, including two DCGAN models, a two-dimensional WGAN-GP model, and a one-dimensional WGAN model. The results of experiments using these models show that the technique is promising, but ultimately needs additional research.

There are several ways that the existing models could be improved. Using larger pretrained GloVe vectors, such as the 300-dimensional ones, would “spread out” words more so the model would be less confused. When translating back into English language tokens this would be especially advantageous. Certain tokens like punctuation marks are very close in the vector space and are easily confused by closest cosine similarity. To streamline the models, word embeddings that have their dimensions as powers of two would be advantageous, especially using up-sampled convolution, which was used in the WGAN-GP experiments. Additional GAN architectures could be explored. One promising architecture mentioned in the literature review was the diversity-promoting generative adversarial network (DP-GAN) by Xu et al. Smaller experiments could be conducted using these techniques. The training data could be constrained to questions or to very short sentences, less than ten words. The models could be trained with negative examples, for instance by using sentences from English language learners while training the discriminator. That might allow the discriminator to pick-out ungrammatical sentences easier, and give better feedback to the generator, ultimately resulting in a better model with better outputs. There are many possibilities to explore in fine-tuning the novel technique of using word embeddings in generative adversarial neural networks.

8. References

Arjovsky, M., Chintala, S., & Bottou, L. (2017). Wasserstein gan. *arXiv preprint arXiv:1701.07875*.

Bahdanau, D., Brakel, P., Xu, K., Goyal, A., Lowe, R., Pineau, J., ... & Bengio, Y. (2016). An actor-critic algorithm for sequence prediction. arXiv preprint arXiv:1607.07086.

Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... & Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems* (pp. 2672-2680).

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., & Courville, A. C. (2017). Improved training of wasserstein gans. In *Advances in Neural Information Processing Systems* (pp. 5767-5777).

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

Joulin, A., Grave, E., Bojanowski, P., Douze, M., Jégou, H., & Mikolov, T. (2016). Fasttext. zip: Compressing text classification models. arXiv preprint arXiv:1612.03651.

Kim, Y. (2014). Convolutional neural networks for sentence classification. arXiv preprint arXiv:1408.5882.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

Kiss, T., & Strunk, J. (2006). Unsupervised multilingual sentence boundary detection. Computational Linguistics, 32(4), 485-525.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Pennington, J., Socher, R., & Manning, C. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)* (pp. 1532-1543).

Radford, A., Metz, L., & Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434.

Rajeswar, S., Subramanian, S., Dutil, F., Pal, C., & Courville, A. (2017). Adversarial generation of natural language. arXiv preprint arXiv:1705.10929.

Xu, J., Sun, X., Ren, X., Lin, J., Wei, B., & Li, W. (2018). DP-GAN: Diversity-Promoting Generative Adversarial Network for Generating Informative and Diversified Text. arXiv preprint arXiv:1802.01345.

Yu, L., Zhang, W., Wang, J., & Yu, Y. (2017, March). SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient. In AAAI (pp. 2852-2858).