Generative Adversarial Networks for text using word2vec intermediaries

Akshay Budhkar[1, 2, 4], Krishnapriya Vishnubhotla[1], Safwan Hossain[1, 2] and Frank Rudzicz[1, 2, 3, 5]

[1]Department of Computer Science, University of Toronto
{abudhkar, vkpriya, frank}@cs.toronto.edu
[2]Vector Institute
safwan.hossain@mail.utoronto.ca
[3]St Michael's Hospital
[4]Georgian Partners
[5]Surgical Safety Technologies Inc.

Abstract

Generative adversarial networks (GANs) have shown considerable success, especially in the realistic generation of images. In this work, we apply similar techniques for the generation of text. We propose a novel approach to handle the discrete nature of text, during training, using word embeddings. Our method is agnostic to vocabulary size and achieves competitive results relative to methods with various discrete gradient estimators.

1 Introduction

Natural Language Generation (NLG) is often regarded as one of the most challenging tasks in computation (Murty and Kabadi, 1987). It involves training a model to do language generation for a series of abstract concepts, represented either in some logical form or as a knowledge base. Goodfellow introduced generative adversarial networks (GANs) (Goodfellow et al., 2014) as a method of generating synthetic, continuous data with realistic attributes. The model includes a discriminator network (D), responsible for distinguishing between the real and the generated samples, and a generator network (G), responsible for generating realistic samples with the goal of fooling the D. This setup leads to a minimax game where we maximize the value function with respect to D, and minimize it with respect to G. The ideal optimal solution is the complete replication of the real distributions of data by the generated distribution.

GANs, in this original setup, often suffer from the problem of mode collapse - where the G manages to find a few modes of data that resemble real data, using them consistently to fool the D. Workarounds for this include updating the loss function to incorporate an element of multi-diversity. An optimal D would provide G with the information to improve, however, if at the current stage of training it is not doing that yet, the gradient of G vanishes. Additionally, with this loss function, there is no correlation between the metric and the generation quality, and the most common workaround is to generate targets across epochs and then measure the generation quality, which can be an expensive process.

W-GAN (Arjovsky et al., 2017) rectifies these issues with its updated loss. Wasserstein distance is the minimum cost of transporting *mass* in converting data from distribution P_r to P_g. This loss *forces* the GAN to perform in a min-max, rather than a max-min, a desirable behavior as stated in (Goodfellow, 2016), potentially mitigating mode-collapse problems. The loss function is given by:

$$L_{critic} = \min_G \max_{D \in \mathcal{D}} (E_{x \sim p_r(x)}[D(x)] - E_{\tilde{x} \sim p_g(x)}[D(\tilde{x})]) \quad (1)$$

where $\mathcal{D}$ is the set of 1-Lipschitz functions and P_g is the model distribution implicitly defined by $\tilde{x} = G(z)$, $z \sim p(z)$. A differentiable function is 1-Lipschtiz *iff* it has gradients with norm at most 1 everywhere. Under an optimal D minimizing the value function with respect to the generator parameters minimizes the $\mathcal{W}(p_r, p_g)$, where $\mathcal{W}$ is the Wasserstein distance, as discussed in (Vallender, 1974). To enforce the Lipschitz constraint, the authors propose clipping the weights of the gradient within a compact space $[-c, c]$.

(Gulrajani et al., 2017) show that even though this setup leads to more stable training compared to the original GAN loss function, the architecture suffers from exploding and vanishing gradient problems. They introduce the concept of *gradient penalty* as an alternative way to enforce the Lipschitz constraint, by penalizing the gradient norm

directly in the loss. The loss function is given by:

$$L = L_{critic} + \lambda E_{\hat{x} \sim p(\hat{x})}[(||\nabla_{\hat{x}} D(\hat{x})||_2 - 1)^2] \quad (2)$$

where $\hat{x}$ are random samples drawn from P_x, and L_{critic} is the loss defined in Equation 1.

Empirical results of GANs over the past year or so have been impressive. GANs have gotten state-of-the-art image-generation results on datasets like ImageNet (Brock et al., 2018) and LSUN (Radford et al., 2015). Such GANs are fully differentiable and allow for back-propagation of gradients from D through the samples generated by G. However, if the data is discrete, as, in the case of text, the gradient cannot be propagated back from D to G, without some approximation. Workarounds to this problem include techniques from reinforcement learning (RL), such as policy gradients to choose a discrete entity and reparameterization to represent the discrete quantity in terms of an approximated continuous function (Williams, 1992; Jang et al., 2016).

1.1 Techniques for GANs for text

SeqGAN (Yu et al., 2017) uses policy gradient techniques from RL to approximate gradient from discrete G outputs, and applied MC rollouts during training to obtain a loss signal for each word in the corpus. MaliGAN (Che et al., 2017) rescales the reward to control for the vanishing gradient problem faced by SeqGAN. RankGAN (Lin et al., 2017) replaces D with an *adversarial ranker* and minimizes pair-wise ranking loss to get better convergence, however, is more expensive than other methods due to the extra sampling from the original data. (Kusner and Hernández-Lobato, 2016) used the Gumbel-softmax approximation of the discrete one-hot encoded output of the G, and showed that the model learns rules of a context-free grammar from training samples. (Rajeswar et al., 2017), the state of the art in 2017, forced the GAN to operate on continuous quantities by approximating the one-hot output tokens with a softmax distribution layer at the end of the G network.

MaskGAN (Fedus et al., 2018) uses policy gradient with REINFORCE estimator (Williams, 1992) to train the model to predict a word based on its context, and show that for the specific blank-filling task, their model outperforms maximum likelihood model using the perplexity metric. LeakGAN (Guo et al., 2018) allows for long sentence generation by *leaking* high-level information from D to G, and generates a latent representation from the features of the already generated words, to aid in the next word generation. TextGAN (Zhang et al., 2017) adds an element of diversity to the original GAN loss by employing the Maximum Mean Discrepancy objective to alleviate mode collapse.

In the latter half of 2018, (Zhu et al., 2018) introduced *Texygen*, a benchmarking platform for natural language generation, while introducing standard metrics apt for this task. (Lu et al., 2018) surveys all these new methods along with other baselines, and documents model performance on standard corpus like EMNLP2017 WMT News[1] and Image COCO[2].

2 Motivation

2.1 Problems with the Softmax Function

The final layer of nearly all existing language generation models is the softmax function. It is usually the slowest to compute, leaves a large memory footprint and can lead to significant speedups if replaced by approximate continuous outputs (Kumar and Tsvetkov, 2018). Given this bottleneck, models usually limit the vocabulary size to a few thousand and use an unknown token (*unk*) for the rare words. Any change in the allowed vocabulary size also means that the researcher needs to modify the existing model architecture.

Our work breaks this bottleneck by having our G produce a sequence (or stack) of continuous distributed word vectors, with n dimensions, where $n << V$ and V is the vocabulary size. The expectation is that the model will output words in a semantic space, that is produced words would either be correct or close synonyms (Mikolov et al., 2013; Kumar and Tsvetkov, 2018), while having a smaller memory footprint and faster training and inference procedures.

2.2 GAN2vec

In this work, we propose GAN2vec - GANs that generate real-valued word2vec-like vectors (as opposed to discrete one-hot encoded outputs). While this work mainly focuses specifically on word2vec-based representation, it can be easily extended to other embedding techniques like GloVe and fastText.

---

[1] http://www.statmt.org/wmt17/
[2] http://cocodataset.org/

Expecting a neural network to generate text is, intuitively, expecting it to learn all the nuances of natural language, including the rules of grammar, context, coherent sentences, and so on. Word2vec has shown to capture parts of these subtleties by capturing the inherent semantic meaning of the words, and this is shown by the empirical results in the original paper (Mikolov et al., 2013) and with theoretical justifications by (Ethayarajh et al., 2018). GAN2vec breaks the problem of generation down into two steps, the first is the word2vec mapping, with the following network expected to address the other aspects of sentence generation. It also allows the model designers to swap out word2vec for a different type of word representation that is best suited for the specific language task at hand.

As a manifestation of the similar-context words getting grouped in word embedding space - we expect GAN2vec to have synonymic variety in the generation of sentences. Generating real-valued word vectors also allows the G architecture to be vocabulary-agnostic, as modifying the training data would involve just re-training the word embedding with more data. While this would involve re-training the weights of the GAN network, the initial architectural choices could remain consistent through this process. Finally, as discussed in Section 2.1, we expect a speed-up and smaller memory footprint by adapting this approach.

All the significant advances in the adaptation of GANs since its introduction in 2016, has been focused in the field of images. We have got to the point, where sometimes GAN architectures have managed to generate images even *better* than real images, as in the case of BigGAN (Brock et al., 2018). While there have been breakthroughs in working with text too, the rate of improvement is no-where close to the success we have had with images. GAN2vec attempts to bridge this gap by providing a framework to swap out image representations with word2vec representations.

3 The Architecture

Random normal noise is used as an input to the G which generates a sequence of word2vec vectors. We train the word2vec model on a real text corpus and generate a stack word vector sequences from the model. The generated and the real samples are then sent to D, to identify as real or synthetic. The generated word vectors are converted to text at

regular intervals during training and during inference for human interpretation. A nearest-neighbor approach based on cosine similarity is used to find the closest word to the generated embedding in the vector space.

4 The Algorithm

The complete GAN2vec flow is presented in Algorithm 1.

Algorithm 1 GAN2vec Framework

1: Train a word2vec model, e, on the train corpus

2: Transform text to a stack of word2vec vectors using e
3: Pre-train D for t iterations on real data
4: **for** k iterations **do**
5:     Send minibatch of real data to D
6:     $G(z)$ = Sample random normal z and feed to G
7:     Send minibatch of G generated data, $G(z)$, to D
8:     Update D using gradient descent
9:     Update G using gradient ascent
10: **end for**
11: $G(z)$ = Sample random normal z and feed to G
12: $w_{generated} = \underset{w}{argmin}\{d(\hat{e}, e(w))\}$, for every $\hat{e}$ in $G(z)$ and every w in the corpus

---

4.1 Conditional GAN2vec

We modify GAN2vec to measure the adaptability of GAN2vec to conditions provided *a priori*, as seen in (Mirza and Osindero, 2014). This change can include many kinds of conditions like positive/negative, question/statement or dementia/controls, allowing for the ability to analyze examples from various classes on the fly during inference. Both the G (and D) architectures get passed the condition at hand as an input, and the goal of G now is to generate a realistic sample given the specific condition.

5 Environmental Setup

All the experiments are run using Pytorch (Paszke et al., 2017). Word2vec training is done using the *gensim* library (Řehůřek and Sojka, 2010). Unless specified otherwise, we use the default parameters for all the components of these libraries, and all
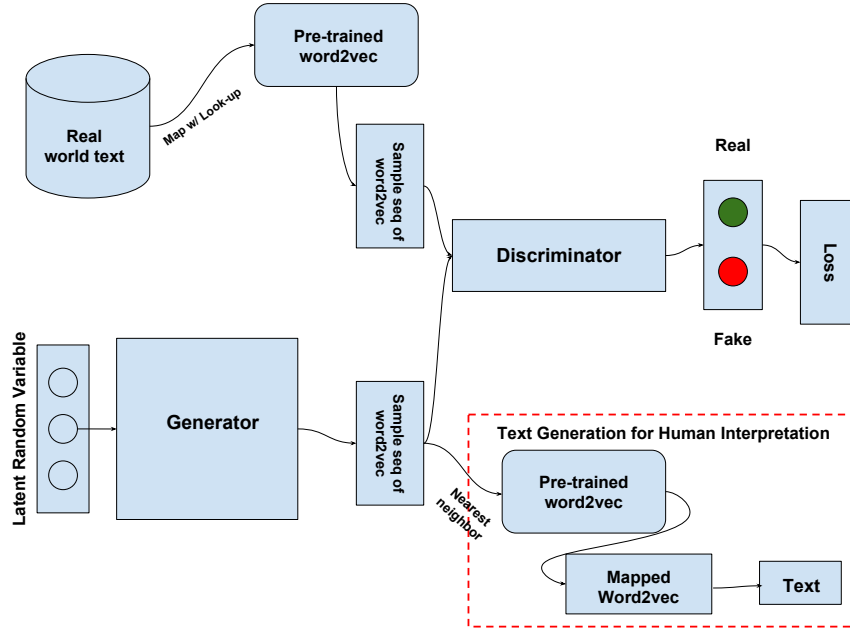
17

Figure 1: Structure of the GAN2vec model. Random normal noise is given as input to the generator network G. The discriminator network D is responsible for determining whether a sample originated from G or from the training set. At inference time, we use a nearest-neighbor approach to convert the output from G into human-readable text.

our models are trained for **100** epochs. The word embedding dimensions are set to 64. The learning rate for the ADAM optimizers for D and G are set to 0.0001, with the exponential decay rates for the first and second moments set to 0.5 and 0.999 respectively.

All our Ds take the word2vec-transformed vectors as an input and apply two 2-D convolutions, followed by a fully connected layer to return a single value. The dimensions of the second 2-D convolution are the only things varied to address the different input dimensions. Similarly, our Gs take a random normal noise of size 100 and transform it to the desired output by passing it through a fully-connected layer, and two 2-D fractionally-strided convolution layers. Again, the dimensions of the second fractionally-strided convolution are the only variables to obtain different output dimensions.

Normalizing word vectors after training them has no significant effect on the performance of GAN2vec, and all the results that we present do not carry out this step. Keeping in punctuation helped improve performance, as expected, and none of the experiments filter them out.

To facilitate stable GAN training, we make the following modifications, covered by (Chintala et al., 2016), by running a few preliminary tests on

a smaller sample of our dataset:

- Use LeakyRELU instead of RELU

- Send generated and real mini-batches to D in separate batches

- Use label smoothing by setting the target labels to 0.9 and 0.1 instead of 1 and 0 for real and fake samples respectively (for most of our experiments).

6 Metrics

6.1 BLEU

BLEU (Papineni et al., 2002) originated as a way to measure the quality of machine translation given certain ground truth. Many text generation papers use this as a metric to compare the quality of the generated samples to the target corpus. A higher n-gram coverage will yield a higher BLEU score, with the score reaching a 100% if all the generated n-grams are present in the corpus. The two potential flaws with this metric are: 1) It does not take into account the diversity of the text generation, this leads to a situation where a mode-collapsing G that produces the same one sentence from the corpus gets a score of 100%. 2) It penalizes the generation of grammatically coherent sentences with novel n-grams, just because

they are absent from the original corpus. Despite these problems, we use BLEU to be consistent with other GANs for text papers. We also present generated samples for the sake of qualitative evaluation by the reader.

6.2 Self-BLEU

Self-BLEU is introduced as a metric to measure the diversity of the generated sentences. It does a corpus-level BLEU on a set of generated sentences, and reports the average BLEU as a metric for a given model. A lower self-BLEU implies a higher diversity in the generated sentences, and accordingly a lower chance that the model has mode collapsed.

It is not clear from (Zhu et al., 2018)'s work on how many sentences *Texygen* generates to calculate Self-BLEU. For purposes of GAN2vec's results, we produce 1000 sentences, and for every sentence do a corpus-level BLEU on remaining 999 sentences. Our results report the average BLEU across all the outputs.

7 Chinese Poetry Dataset

The Chinese Poetry dataset, introduced by (Zhang and Lapata, 2014) presents simple 4-line poems in Chinese with a length of 5 or 7 tokens (henceforth referred to Poem 5 and Poem 7 respectively). Following previous work by (Rajeswar et al., 2017) and (Yu et al., 2017), we treat every line as a separate data point. We modify the Poem 5 dataset to add start and end of tokens, to ensure the model captures (at least) that pattern through the corpus (given our lack of Chinese knowledge). This setup allows us to use identical architectures for both the Poem 5 and Poem 7 datasets. We also modify the GAN2vec loss function with the objective in Eq. 2, and report the results below.

	Rajeswar et al.	GAN2vec	GAN2vec (wGAN)
Poem 5	– (train)	37.90% (train)	**53.5%** (train)
	87.80% (test)	22.50% (test)	**25.78%** (test)
Poem 7	– (train)	30.14% (train)	**66.45%** (train)
	65.60% (test)	10.20% (test)	**22.07%** (test)

Table 1: Chinese Poetry BLEU-2 scores.

The better performance of the GAN2vec model with the wGAN objective is in-line with the im-

age results in Gulrajani et al. (2017)'s work. We were not able to replicate (Rajeswar et al., 2017)'s model on the Chinese Poetry dataset to get the reported results on the test set. This conclusion is in-line with our expectation of lower performance on the test set, given the small overlap in the bigram coverage between the provided train and test sets. (Lu et al., 2018) also point out that this work is unreliable, and that their replicated model suffered from severe mode-collapse. On 1000 generated sentences of the Poem-5 dataset, our model has a self BLEU-2 of **66.08%** and self BLEU-3 of **35.29%**, thereby showing that our model does not mode collapse.

8 CMU-SE Dataset

CMU-SE[3] is a pre-processed collections of simple English sentences, consisting of 44,016 sentences and a vocabulary of 3,122-word types. For purposes of our experiments here, we limit the number of sentences to 7, chosen empirically to capture a significant share of the examples. For the sake of simplicity in these experiments, for the *real* corpus, sentences with fewer than seven words are ignored, and those with more than seven words are cut-off at the seventh word.

Table 1 presents sentences generated by the original GAN2vec model. Appendix A.2 includes additional examples. While this is a small subset of randomly sampled examples, on a relatively simple dataset, the text quality appears competitive to the work of (Rajeswar et al., 2017) on this corpus.

Rajeswar et al. (2017)
<s> will you have two moment ? </s>
<s> how is the another headache ? </s>
<s> what s in the friday food ? ? </s>
<s> i d like to fax a newspaper . </s>
GAN2vec
<s> i dropped my camera . </s>
<s> i 'd like to transfer it
<s> i 'll take that car ,
<s> prepare whisky and coffee , please

Table 2: Example sentences generated by the original GAN2vec. We report example sentences from Rajeswar et al. (2017) and from our GAN2vec model on CMU-SE.

8.1 Conditional GAN2vec

We split the CMU-SE dataset into *questions* and *sentences*, checking for the presence of a question mark. We modify the original GAN2vec, as seen in Section A.1, to now include these labels. Our conditional GANs learn to generate mainly coherent sentences on the CMU-SE dataset, as seen in Table 3.
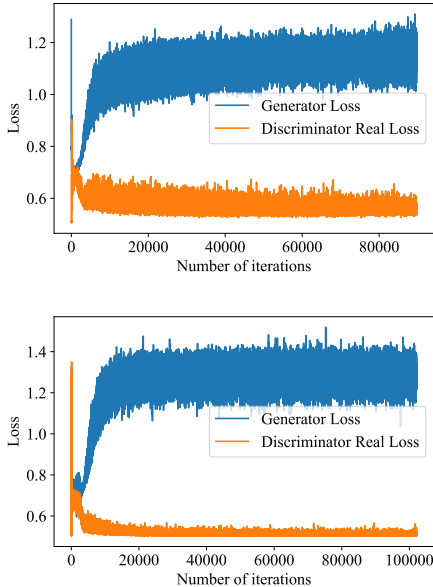

Figure 2: The minimax loss of D and G, with increasing iterations for the GAN2vec model (top) and the conditional GAN2vec (bottom).

Figure 2 shows the loss graphs for our GAN2vec and conditional GAN2vec trained for ~300 epochs. As seen above, the conditional GAN2vec model generates relatively atypical sentences. This is supported by the second loss curve in Figure 2. The G loss follows a progression similar to the normal GAN2vec case, but the loss is about 16% more through the 100 epochs.

8.2 Hyperparameter Variation Study

We study the effects of different initial hyperparameters for GAN2vec by reporting the results in Table 4. All the experiments were run ten times, and we report the best scores for every configuration. It must be noted that for conditional GAN2vec training for this experiment, we randomly sample points from the CMU-SE corpus to enforce a 50-50 split across the two labels (question and sentence).

The overall performance of most of the models is respectable, with all models generating gram-

matically coherent sentences. GAN2vec with wGAN objective outperforms original GAN2vec, and is inline with the results of (Gulrajani et al., 2017) and our results in Section 7. Sense2vec does not have a significant improvement over the original word2vec representations. In agreement with (Goodfellow, 2016), providing labels in the conditional variant leads to better performance.
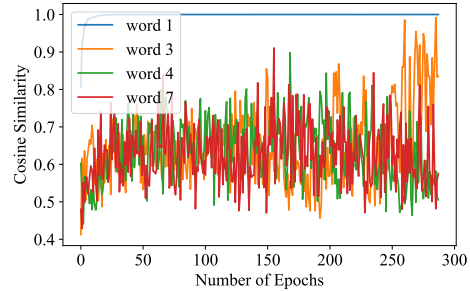
8.3 Word2vec cosine similarity

Figure 3: Cosine similarities of the first, third, fourth, and seventh words to the closest words from sentences generated by GAN2vec trained on the CMU-SE dataset.

During training, we map our generated word2vec vectors to the closest words in the embedding space and measure the point-wise cosine similarity of the generated vector and the closest neighbour's vector. Figure 3 shows these scores for the first, third, fourth and seventh word of the 7-word generated sentences on the CMU-SE dataset for about 300 epochs. The model immediately learns that it needs to start a sentence with $<s>$ and gets a cosine similarity of around 1. For the other words in that sentence, the model tends to get better at generating word vectors that are close to their real-valued counterparts of the nearest neighbours. It seems as if the words close to the start of the sentence follow this trend more strongly (as seen with words 1 and 3) and it is relatively weaker for the last word of the sentence.

9 Coco Image Captions Dataset

The Coco Dataset is used to train and generate synthetic data as a common dataset for all the best-performing models over the last two years. In *Texygen*, the authors set the sentence length to 20. They train an oracle that generates 20,000 sentences, with one half used as the training set and the rest as the test set. All the models in this benchmark are trained for 180 epochs.

Questions	Sentences
<s> can i get you want him	<s> i bring your sweet inexpensive beer
<s> where 's the hotel ?	<s> they will stop your ship at
<s> what is the fare ? </s>	<s> i had a pocket . </s>
<s> could you buy the timetable ?	<s> it 's ten at detroit western

Table 3: Examples of sentences generated by the conditional GAN. We report examples of sentences with our model conditioned on sentence type, i.e., question or sentence.

Architecture	Conditional	Vector Type	Loss function	BLEU-2	BLEU-3
R.1	No	Sense2vec	Original	0.743	0.41
R.1	No	Sense2vec	wgan	0.7933	0.4728
R.1	No	Word2Vec	wgan	0.74	0.43
C.1	Yes	word2vec	Original (Real)	0.717	0.412
C.1	Yes	word2vec	Original	0.743	0.4927
C.1	Yes	word2vec	wgan	**0.7995**	**0.5168**
C.1	Yes	word2vec	wgan	**0.821**	**0.51**
C.2	Yes	word2vec	wgan	**0.8053**	**0.499**

Table 4: Performance of different models on the CMU-SE train dataset. $R.1$ is the original GAN2vec, $C.1$ is $R.1$ modified with addition of labels, $C.2$ adds batch normalization on the CNN layer of G. Original (Real) sets the real label to 0.9, the rest use 1.
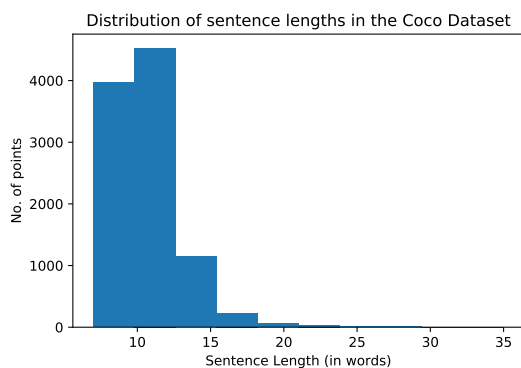

Figure 4: Distribution of sentence lengths in the Coco dataset. Most of the captions have less than 20 words, the cut-off set by *Texygen*.

Figure 4 shows the distribution of the sentence lengths in this corpus. For purposes of studying the effects of longer training sentences on GAN2vec, we set the sentence lengths to 7, 10 and 20 (with the respective models labeled as GAN2vec-7, GAN2vec-10, GAN2vec-20 going forward). Any sentence longer than the predefined sentence length is cut off to include only the initial words. Sentences shorter than this length are padded with an end of sentence character to fill up the remaining words (we use a comma (,) for purposes of our experiments as all the sentences in the corpus end with either a full stop or a word). We tokenize the sentences us-

ing NLTK's word tokenizer[4] which uses regular expressions to tokenize text as in the Penn Treebank corpus[5]. We also report the results of a naive *split at space* approach for the GAN2vec-20 architecture (GAN2vec-20-a), to compare different ways of tokenizing the corpus. We only use the objective from Equation 2, given its superior performance to original GAN2vec, as seen in the previous sections.

The results are summarized in the tables below:

Model	BLEU-2	BLEU-3
LeakGAN	0.926	0.816
SeqGAN	0.917	0.747
MLE	0.731	0.497
TextGAN	0.65	0.645
GAN2vec-7	0.548	0.271
GAN2vec-10	0.641	**0.342**
GAN2vec-20-a	0.618	0.294
GAN2vec-20	**0.661**	0.335

Table 5: Model BLEU scores on Train Set of the Coco Dataset (higher is better).

On the train set (Table 5), GAN2vec models have BLEU-2 scores comparable to its SOTA

[4]https://www.nltk.org/api/nltk.tokenize.html#nltk.tokenize.word_tokenize
[5]https://catalog.ldc.upenn.edu/docs/LDC95T7/cl93.html

counterparts, with the GAN2vec-20 model having better bigram coverage that TextGAN. The BLEU-3 scores, even though commendable, do not match up as well, possibly signaling that our models cannot keep coherence through longer sentences. The increase in the cut-off sentence length, surprisingly, does not degrade performance. As expected, a trained word tokenizer outperforms its *space-split* counterpart. The performance of the GAN2vec models on the test set (Table 6) follows the same trends as that on the train set.

Model	BLEU-2	BLEU-3
LeakGAN	0.746	0.816
SeqGAN	0.745	0.53
MLE	0.731	0.497
TextGAN	0.593	0.645
GAN2vec-7	0.429	0.196
GAN2vec-10	0.527	**0.245**
GAN2vec-20-a	0.484	0.206
GAN2vec-20	**0.551**	0.232

Table 6: Model BLEU scores on Test Set of the Coco Dataset (higher is better).

Model	BLEU-2	BLEU-3
LeakGAN	0.966	0.913
SeqGAN	0.95	0.84
MLE	0.916	0.769
TextGAN	0.942	0.931
GAN2vec-7	**0.537**	**0.254**
GAN2vec-10	0.657	0.394
GAN2vec-20-a	0.709	0.394
GAN2vec-20	0.762	0.518

Table 7: Self BLEU scores of the models trained on the Coco dataset (lower is better).

Table 7 reports the self-BLEU scores, and all the GAN2vec models significantly outperform the SOTA models, including MLE. This implies that GAN2vec leads to more diverse sentence generations and is less susceptible to mode collapse.

10 Discussions

Overall, GAN2vec can generate grammatically coherent sentences, with a good bi-gram and tri-gram coverage from the chosen corpus. BLEU does not reward the generation of semantically and syntactically correct sentences if the associated n-grams are not present in the corpus, and

coming up with a new standard evaluation metric is part of on-going work. GAN2vec seems to have comparable, if not better, performance compared to Rajeswar et al. (2017)'s work on two distinct datasets. It depicts the ability to capture the critical nuances when trained on a conditional corpus. While GAN2vec performs slightly worse than most of the SOTA models using the *Texygen* benchmark, it can generate a wide variety of sentences, possibly given the inherent nature of word vectors, and is less susceptible to mode collapse compared to each of the models. GAN2vec provides a simple framework, with almost no overhead, to transfer state of the art GAN research in computer vision to natural language generation.

We observe that the performance of GAN2vec gets better with an increase in the cut-off length of the sentences. This improvement could be because of *extra* training points for the model. The drop from BLEU-2 to BLEU-3 scores is more extreme than the other SOTA models, indicating that GAN2vec may lack the ability to generate *long* coherent sentences. This behavior could be a manifestation of the chosen D and G architectures, specifically the filter dimensions of the convolution neural networks. Exploration of other structures, including RNN-based models with their ability to remember long term dependencies, might be good alternatives to these initial architecture choices. Throughout all the models in the *Texygen* benchmark, there seems to be a mild negative correlation between diversity and performance. GAN2vec in its original setup leans more towards the generation of new and diverse sentences, and modification of its loss function could allow for tilting the model more towards accurate NLG.

11 Conclusion

While various research has extended GANs to operate on discrete data, most approaches have approximated the gradient in order to keep the model end-to-end differentiable. We instead explore a different approach, and work in the continuous domain using word embedding representations. The performance of our model is encouraging in terms of BLEU scores, and the outputs suggest that it is successfully utilizing the semantic information encoded in the word vectors to produce new, coherent and diverse sentences.

Acknowledgements

References

Martin Arjovsky, Soumith Chintala, and Léon Bottou. 2017. Wasserstein GAN. *arXiv preprint arXiv:1701.07875*.

Andrew Brock, Jeff Donahue, and Karen Simonyan. 2018. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*.

Tong Che, Yanran Li, Ruixiang Zhang, R Devon Hjelm, Wenjie Li, Yangqiu Song, and Yoshua Bengio. 2017. Maximum-likelihood augmented discrete generative adversarial networks. *arXiv preprint arXiv:1702.07983*.

Soumith Chintala, Emily Denton, Martin Arjovsky, and Michael Mathieu. 2016. How to train a gan? tips and tricks to make gans work.

Kawin Ethayarajh, David Duvenaud, and Graeme Hirst. 2018. Towards understanding linear word analogies. *arXiv preprint arXiv:1810.04882*.

William Fedus, Ian Goodfellow, and Andrew M Dai. 2018. MaskGAN: Better Text Generation via Filling in the _. *arXiv preprint arXiv:1801.07736*.

Ian Goodfellow. 2016. NIPS 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. 2017. Improved training of wasserstein gans. In *Advances in Neural Information Processing Systems*, pages 5767–5777.

Jiaxian Guo, Sidi Lu, Han Cai, Weinan Zhang, Yong Yu, and Jun Wang. 2018. Long text generation via adversarial training with leaked information. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Eric Jang, Shixiang Gu, and Ben Poole. 2016. Categorical reparameterization with Gumbel-softmax. *arXiv preprint arXiv:1611.01144*.

Sachin Kumar and Yulia Tsvetkov. 2018. Von mises-fisher loss for training sequence to sequence models with continuous outputs. *arXiv preprint arXiv:1812.04616*.

Matt J Kusner and José Miguel Hernández-Lobato. 2016. GANs for sequences of discrete elements with the Gumbel-softmax distribution. *arXiv preprint arXiv:1611.04051*.

Kevin Lin, Dianqi Li, Xiaodong He, Zhengyou Zhang, and Ming-Ting Sun. 2017. Adversarial ranking for language generation. In *Advances in Neural Information Processing Systems*, pages 3155–3165.

Sidi Lu, Yaoming Zhu, Weinan Zhang, Jun Wang, and Yong Yu. 2018. Neural text generation: Past, present and beyond. *arXiv preprint arXiv:1803.07133*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

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

Katta G Murty and Santosh N Kabadi. 1987. Some np-complete problems in quadratic and nonlinear programming. *Mathematical programming*, 39(2):117–129.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics.

Adam Paszke, Sam Gross, Soumith Chintala, and Gregory Chanan. 2017. Pytorch.

Alec Radford, Luke Metz, and Soumith Chintala. 2015. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*.

Sai Rajeswar, Sandeep Subramanian, Francis Dutil, Christopher Pal, and Aaron Courville. 2017. Adversarial generation of natural language. *arXiv preprint arXiv:1705.10929*.

Radim Řehůřek and Petr Sojka. 2010. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta. ELRA. http://is.muni.cz/publication/884893/en.

SS Vallender. 1974. Calculation of the wasserstein distance between probability distributions on the line. *Theory of Probability & Its Applications*, 18(4):784–786.

Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Reinforcement Learning*, pages 5–32. Springer.

Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. 2017. SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient. In *AAAI*, pages 2852–2858.

Xingxing Zhang and Mirella Lapata. 2014. Chinese poetry generation with recurrent neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 670–680, Doha, Qatar. Association for Computational Linguistics.

Yizhe Zhang, Zhe Gan, Kai Fan, Zhi Chen, Ricardo Henao, Dinghan Shen, and Lawrence Carin. 2017. Adversarial feature matching for text generation. *arXiv preprint arXiv:1706.03850*.

Yaoming Zhu, Sidi Lu, Lei Zheng, Jiaxian Guo, Weinan Zhang, Jun Wang, and Yong Yu. 2018. Texygen: A benchmarking platform for text generation models. *arXiv preprint arXiv:1802.01886*.

A CMU-SE

A.1 Conditional Architecture

While designing GAN2vec to support conditional labels, as presented in Mirza and Osindero (2014), we used the architecture in Figure 5 for our G. The label is sent as an input to both the fully connected and the de-convolution neural layers. The same change is followed while updating D to support document labels.
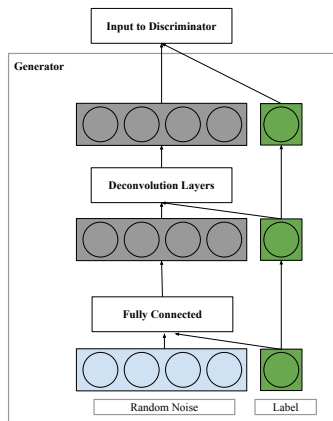

Figure 5: Generator Architecture for Conditional GAN2vec.

A.2 Examples of Generated Sentences

Table 8: Sentences Generated with GAN2vec on CMU-SE

Generated Sentences
<s> can you have a home </s>
<s> i 'd like to leave the
<s> this is the baggage . </s>
<s> i 'd like a driver ?
<s> do you draw well . </s>
<s> i 'd like to transfer it
<s> please explain it <unk>. </s>
<s> can i book a table .
<s> i 'll take that car ,
<s> would i like a stay ?
<s> will you check it . </s>
<s> do you find it ? </s>
<s> i want some lovely cream .
<s> could you recommend a hotel with
<s> can you get this one in
<s> where 's the petrol station ?
<s> what 's the problem ? </s>
<s> i have a hangover dark .
<s> i come on the monday .
<s> i appreciate having a sushi .
<s> the bus is busy , please
<s> i dropped my camera . </s>
<s> i want to wash something .
<s> it 's too great for <unk>
<s> i have a driver in the
<s> it is delicious and <unk><unk>
<s> please leave your luggage . .
<s> i had alcohol wow , </s>
<s> it is very true . </s>
<s> where 's the hotel ? </s>
<s> will you see the cat ? </s>
<s> where is this bus ? </s>
<s> how was the spirits airline warranty
<s> i would n't sunburn you
<s> prepare whisky and coffee , please