Abstract

Generative adversarial networks are an effective approach for learning rich latent representations of continuous data, but have proven difficult to apply directly to discrete structured data, such as text sequences or discretized images. Ideally we could encode discrete structures in a continuous code space to avoid this problem, but it is difficult to learn an appropriate general-purpose encoder. In this work, we consider a simple approach for handling these two challenges jointly, employing a discrete structure autoencoder with a code space regularized by generative adversarial training. The model learns a smooth regularized code space while still being able to model the underlying data, and can be used as a discrete GAN with the ability to generate coherent discrete outputs from continuous samples. We demonstrate empirically how key properties of the data are captured in the model’s latent space, and evaluate the model itself on the tasks of discrete image generation, text generation, and semi-supervised learning.

1 Introduction

Recent work on generative adversarial networks (GANs) [9] and other deep latent variable models has shown significant progress in learning smooth latent variable representations of complex, high-dimensional continuous data such as images [1 2 25 37]. These latent representations facilitate the ability to apply smooth transformations and interpolations in latent space in order to produce complex modifications of generated outputs, while still remaining on the data manifold.

Unfortunately, learning similar latent representations of discrete structures, such as text sequences or discretized images, remains a challenging problem. Applying GANs directly to this task produces discrete output from the generator, which then requires clever approaches for backpropagation. Furthermore this issue is compounded in cases where the generative model is recurrent, e.g. in sequence modeling. Researchers have circumvented some of these issues by using policy gradient

Submitted to 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.
methods \cite{5,12,36} or with the Gumbel-Softmax distribution \cite{17}. However, neither approach can yet produce robust latent representations directly from samples, particularly for discrete structures.

An alternative approach is to instead encode discrete structures in a continuous code space to circumvent this problem altogether. As this space is continuous, traditional GAN training can be directly applied to learn a latent representation of the code space itself. Samples from the GAN can then be decoded to generate discrete outputs. While in theory this technique can be applied directly, in practice, learning general-purpose autoencoders is in itself a difficult problem.

In this work, we propose a simple extension of this technique by jointly training a code-space GAN and a discrete structure autoencoder, which we call an adversarially regularized autoencoder (ARAE). The approach allows us to use a general-purpose GAN architecture that generates continuous code representations, while at the same time deploying task-specific autoencoder architectures, like a recurrent neural network for text, to produce and decode from these latent representations.

The ARAE approach can be used both as a generative model and as a way to obtain an encoding of the input. First, it learns a GAN with a Gaussian latent space that can be sampled to produce discrete structures. This model can be compared directly with existing generative models. Second, it learns an adversarially regularized encoder that can produce useful code space representations from discrete structures, without requiring an explicit code-space prior. We can compare this method to other specialized autoencoders such as denoising and variational autoencoders.

Our experiments test ARAE on two different discrete domains: discretized images and text sequences. We show that this approach successfully learns latent representations for both tasks, as the model is able to generate coherent samples, ignore or fix corrupted inputs, and produce predictable changes in the outputs when performing manipulations in the latent space. We find that we are able perform consistent sentence manipulations by moving around in the latent space via offset vectors. A similar property was observed in image GANs \cite{25} and word representations \cite{23}. Finally, experiments on a semi-supervised learning task for natural language inference provide quantitative evidence that this approach improves upon continuous representations learned by autoencoders. Code is available at \url{https://github.com/jakezhaojb/ARAE}.

## 2 Related Work

### GANs for Discrete Structures

The success of GANs on images have led many researchers to consider applying GANs to discrete data such as text. Policy gradient methods are a natural way to deal with the resulting non-differentiable generator objective when training directly in discrete space \cite{8,34}. When trained on text data however, such methods often require pre-training/co-training with a maximum likelihood (i.e. language modeling) objective \cite{5,36,18}. This precludes there being a latent encoding of the sentence, and is also a potential disadvantage of existing language models (which can otherwise generate locally-coherent samples).

Another direction of work has been through reparameterizing the categorical distribution with the Gumbel-Softmax trick \cite{14,19}—while initial experiments were encouraging on a synthetic task \cite{17}, scaling them to work on natural language is a challenging open problem. There has also been a flurry of recent, related approaches that work directly with the soft outputs from a generator \cite{10,28,30,24}. For example, Shen et al. \cite{30} train with adversarial loss for unaligned style transfer between text by having the discriminator act on the RNN hidden states and using the soft outputs at each step as input to an RNN generator.

### Variational Autoencoders

Ideally, autoencoders would learn useful coded feature representations of their inputs. However in practice simple autoencoders often learn a degenerate identity mapping where the latent code space is free of any structure. One way—among others—to regularize the code space is through having an explicit prior on the code space and using a variational approximation to the posterior, leading to a family of models called variational autoencoders (VAE) \cite{16,27}. Unfortunately VAEs for text can be challenging to train—for example, if the training procedure is not carefully tuned with techniques like word dropout and KL annealing \cite{4}, the decoder simply becomes a language model and ignores the latent code (although there has been some recent successes with convolutional models \cite{29,35}).

A possible reason for the difficulty in training VAEs is due to the strictness of the prior (usually a spherical Gaussian) and/or the parameterization of the posterior. There has been some work on making
the prior/posterior more flexible through explicit parameterization \cite{26,15,6}. One notable technique is adversarial autoencoders (AAE) \cite{21} which attempt to imbue the model with a more flexible prior implicitly through adversarial training. In AAEs, the discriminator is trained to distinguish between samples from a fixed prior distribution and the input encoding, thereby pushing the code distribution to match the prior. Our approach has similar motivation, but notably we do not sample from a fixed prior distribution—our ‘prior’ is instead parameterized through a generator. Nonetheless, this view (which has been observed by various researchers \cite{32,22,20}) provides an interesting connection between VAEs and GANs.

3 Background

3.1 Generative Adversarial Networks

Generative Adversarial Networks (GANs) are a class of parameterized implicit generative models \cite{9}. The method approximates drawing samples from a true distribution \( c \sim P_r \) by instead employing a latent variable \( z \) and a parameterized deterministic generator function \( \hat{c} = g_\theta(z) \) to produce generated samples \( \tilde{c} \sim P_g \). The aim is to characterize the complex data manifold described by the unknown \( P_r \) within the latent space \( z \).

GAN training utilizes two separate models: a generator \( g(z) \) maps a latent vector from some easy-to-sample source distribution to a value and a critic/discriminator \( f(c) \) aims to distinguish real data and fake samples, generated by \( g \). The generator is trained to fool the critic, and the critic to separate out real from generated.

In this work, we utilize the recently-proposed Wasserstein GAN (WGAN) \cite{1}. WGAN replaces the Jensen-Shannon divergence in standard GANs with Earth-Mover (Wasserstein-1) distance. WGAN training uses the following min-max optimization over generator parameters \( \theta \) and critic parameters \( w \):

\[
\min_{\theta} \max_{w \in W} \mathbb{E}_{c \sim P_r}[f_w(c)] - \mathbb{E}_{\tilde{c} \sim P_g}[f_w(\tilde{c})],
\]

where \( f_w : \mathcal{C} \rightarrow \mathbb{R} \) denotes the critic function, \( \tilde{c} \) is obtained from the generator, \( \tilde{c} = g_\theta(z) \), and \( P_r \) and \( P_g \) are real (true samples) and fake (generated samples) distribution respectively.

Notably the critic parameters \( w \) are restricted to an 1-Lipschitz function set \( W \), which can be shown to make this term correspond to Wasserstein-1 Distance. We follow \cite{1} and use a naive implementation to approximately enforce this property by weight-clipping, i.e. \( w = [-\epsilon, \epsilon]^d \).

Throughout this work we only use \( g_\theta \) and \( f_w \) as fully-connected networks, namely MLPs.

3.2 Discrete Structure Autoencoders

An autoencoder is any model trained to map an input to a code space and then back to the original form. Ideally the code represents important abstracted features of the original input (although this is difficult to formalize) instead of learning to simply copy any given input.

We are interested in probabilistic autoencoders for discrete structures. Define \( X = V^n \) to be a discrete set of structures where \( V \) is a vocabulary of symbols. For instance, for binarized images \( V = \{0, 1\} \) and \( n \) is the number of pixels, or for sentences \( V = \{1, \ldots, \text{#words}\} \) and \( n \) is the sentence length.

A discrete structure autoencoder consists of two parameterized functions: a deterministic encoder function \( \text{enc}_\phi : X \rightarrow \mathcal{C} \) with parameters \( \phi \) and a decoder distribution \( p_\psi(x \mid c) \) with parameters \( \psi \) that gives a distribution over structures \( X \). The model is trained on cross-entropy reconstruction loss where we learn parameters to minimize the negative log-likelihood of reconstruction:

\[
\mathcal{L}_{\text{AE}}(\phi, \psi) = -\log p_\psi(x \mid \text{enc}_\phi(x))
\]

Computing this for arbitrary sets is intractable, so the choice of \( p_\psi \) is important and problem specific. Finally it is often useful to use the decoder to produce a point estimate from \( X \). We call this \( \hat{x} = \arg \max_x p_\psi(x \mid \text{enc}_\phi(x)) \) When \( x = \hat{x} \) the autoencoder is said to copy the input, or perfectly reconstruct \( x \).

4 Model

An adversarially regularized autoencoder (ARAE) combines a discrete autoencoder with a code-space GAN. Our model employs a discrete autoencoder to learn continuous codes based on discrete inputs.
and a WGAN to learn an implicit probabilistic model over these codes. The aim is to exploit the GAN’s ability to learn the latent structure of code data, while using an autoencoder to abstract away the encoding and generation of discrete structure to support GAN training.

The main difference with WGANs as described above, is that we no longer have access to observed data samples for the GAN. Instead we have access to discrete structure $x \sim P_x$ where $P_x$ is the distribution of interest. (Working with this space directly would require backpropagating through non-differentiable operations and is the basis for policy gradient methods for GAN training.) We handle this issue by integrating an encoder into the procedure which first maps $x$ to a continuous code $c = \text{enc}_\phi(x)$, i.e. using the code vector for each observed structure defined by $\text{enc}_\phi$.

The full model has a three part objective. We minimize reconstruction error in the AE while employing adversarial training on its code space.

$$\min_{(\phi, \psi)} \mathcal{L}_{AE}(\phi, \psi) \tag{3}$$

$$\min_w \max_{w \in W} \mathcal{L}_{\text{WGAN-Gen}}(w, \phi) = \min_{w \in W} \max_w \mathbb{E}_{x \sim P_x} [f_w(\text{enc}_\phi(x))] - \mathbb{E}_{\tilde{c} \sim P_g} [f_w(\tilde{c})] \tag{4}$$

$$\min_\theta \mathcal{L}_{\text{WGAN-Crit}}(\theta) = \min_\theta \mathbb{E}_{\tilde{c} \sim P_g} [f_w(\tilde{c})] \tag{5}$$

where $P_x$ is the real distribution in the input space. We minimize the three objectives all jointly in this work. Our model is visually depicted in Figure 1. The algorithm used for training is shown in Algorithm 1. We use block coordinate descent to optimize the AE, critic and generator in turn.

Notably with this change we now receive gradients through the encoder from the adversarial loss. This gradient will allow the encoder to help the generator to produce samples in the support of the true data learned by the WGAN critic. Theoretically, the effect of such term should decrease (and eventually diminish) as the GAN converges to a Nash-Equalibrium.

5 Architectures

We consider two different instantiations of ARAEs, one for discrete images and the other for text sequences. For both models we use the same WGAN architecture but substitute in different autoencoder architectures. The generator architecture uses a low dimensional $z$ with a Gaussian prior $p(z) = \mathcal{N}(0, I)$, and maps it to $c$. Both the critic $f_w$ and the generator $g_{\theta}$ are parameterized as feed-forward MLPs. The structure of the deterministic encoder $\text{enc}_\phi$ and probabilistic decoder $p_\psi$ is specialized for the domain.

**Image Model** Our first model uses a fully-connected neural network to encode binarized images. Here $\mathcal{X} = \{0, 1\}^n$ where $n$ is the image size. The encoder used is a feed-forward MLP network mapping from $\{0, 1\}^n \rightarrow \mathbb{R}^m$, $\text{enc}_\phi(x) = \text{MLP}(x; \phi) = c$. The decoder predicts each pixel in $x$ as a parameterized logistic regression, $p_\psi(x \mid c) = \prod_{j=1}^m \sigma(h_j)^{x_j} (1 - \sigma(h_j))^{1-x_j}$ where $h = \text{MLP}(c; \psi)$. 
We consider two different settings for testing the ARAE: (1) images, utilizing the binarized version
\{ψ\}.

\section*{Algorithm 1 ARAE Training Procedure}

\begin{algorithm}
\begin{algorithmic}
\For{number of training iterations \textbf{do}}
\State \textbf{Train the autoencoder}
\State Sample \{x^{(i)}\}_{i=1}^{m} \sim P_x a \text{ batch from the training data}
\State Compute the latent representations \(c^{(i)} = \text{enc}_\phi(x^{(i)})\)
\State Compute the autoencoder loss, \(L_{AE} = \frac{1}{m} \sum_{i=1}^{m} \log p_\psi(x^{(i)} | c^{(i)})\), backpropagation gradients, update the decoder \(\psi\) and the encoder \(\phi\)
\EndFor
\State \textbf{Train the critic}
\For{k steps \textbf{do}}
\State \textbf{Positive sample phase}
\State Compute the adversarial loss on the real samples \(\frac{1}{m} \sum_{i=1}^{m} E_{x \sim P_x} [f_w(c^{(i)})]\) and backpropagation gradients, update the critic \(w\) and the encoder \(\phi\)
\State \textbf{Negative sample phase}
\State Sample a batch of random noise \{z^{(i)}\}_{i=1}^{m} \sim \mathcal{N}(0, I)
\State Generate code representation \(\hat{c}^{(i)} = g_\theta(z^{(i)})\) by passing \(z^{(i)}\) through the generator
\State Compute the adversarial loss \(\frac{1}{m} \sum_{i=1}^{m} E_{z \sim P_z} [f_w(\hat{c}^{(i)})]\), backpropagation gradients, update the critic \(w\), clip the weights of the critic \(w\) to \([-\epsilon, \epsilon]^d\)
\EndFor
\EndAlgorithm
\end{algorithmic}
\end{algorithm}

\textbf{Text Model} \quad Our second model is devised for text. Here \(X = V^n\) where \(n\) is the sentence length and \(V\) is the vocabulary of the underlying language (typically \(|V| \in [10k, 100k]\)). Following usual practice we use a recurrent neural network (RNN) as both the text encoder and decoder. Define an RNN as a parameterized recurrent function \(h_j = \text{RNN}(x_j, h_{j-1}; \phi)\) for \(j = 1 \ldots n\) (with \(h_0 = 0\)) that maps a discrete input structure \(x\) to hidden vectors \(h_1 \ldots h_n\). For the encoder, we define \(\text{enc}_\phi(x) = h_n = c\), the last hidden state in this recurrence. The decoder is defined in a similar way, with parameters \(\psi\). For prediction we combine \(c\) with \(h_j\) to produce a distribution over \(V\) at each time step, \(p_\psi(x | c) = \prod_{j=1}^{n} \text{softmax}(W[h_j; c] + b)_{x_j}\) where \(W\) and \(b\) are parameters (part of \(\psi\)). Finding the most likely sequence \(\hat{x}\) under this distribution is intractable, but we can approximate it using greedy search or beam search. In our experiments we use an LSTM architecture \[13\] for both the encoder/decoder, and train with teacher-forcing.

\textbf{Semi-Supervised Model} \quad Our model is trained in an unsupervised manner as a combination of an autoencoder and a GAN. As an extension we also consider the use case where the code vector is additionally used as input to a supervised classification task. As in the standard semi-supervised setup, we assume that our data consist of a small set of labeled data \(\{x_i, y_i\}_i\) and large set of unlabeled data \(\{x_j\}_j\). We can set up a standard supervised classification loss function using the code vectors from the encoder and a new set of parameters \(\gamma\):

\[L_{\text{NLL}}(\gamma, \phi) = \sum_i \log p_\gamma(y_i | \text{enc}_\phi(x_i))\]  \hspace{1cm} (6)

We then extend our multi-task loss function to include this objective.

\section{6 Methods and Data}

We consider two different settings for testing the ARAE: (1) images, utilizing the binarized version of MNIST, and (2) text, using the Stanford Natural Language Inference corpus \[3\]. This corpus provides a useful testbed as it comprises of sentences with relatively simple structure. The corpus is additionally annotated for pairwise sentence classification, which allows us to experiment with semi-supervised learning in a controlled setting. For this task the model is presented with two sentences—premise and hypothesis—and has to predict their relationship: entailment, contradiction, or neutral. For training, we used a subset of the corpus consisting of sentences of less than 15 words, although preliminary results suggest this approach works up to 30 words.
Figure 2: **Left**: Produced by an AE. **Middle**: Produced by an ARAE. The arrangement of the left and middle figures are: (i)-top blocks are the input to the AE, clean and noised; (ii)-bottom blocks are the corresponding reconstruction. **Right**: Results of the ARAE. The top block consists of the reconstruction of the real MNIST samples; the middle blocks are the output of the decoder taking fake hidden codes generated by the GAN; the bottom blocks are the sample interpolation results, constructed by linearly interpolating in the latent space and then decoding back to the pixels.

We consider several different empirical tasks to test the performance of the model as both an autoencoder (ARAE), by use the encoder aspect, and as latent-variable model (ARAE-GAN), by sampling $z$’s (the two are trained identically). Experiments include: (1) code space structure; does the model preserve natural inputs $x \sim P_x$ while not preserving noised inputs $x'$; (2) semi-supervised learning; does the performance of a supervised model improve when it is additionally trained as an autoencoder; (3) sample generation; how well does a simple model do when trained on generated samples; (4) interpolation and arithmetic; how easily can we manipulate vectors in $\mathcal{Z}$ to smoothly control the generated text samples $\tilde{x}$.

For these experiments we compare to a standard AE, trained without the code-space GAN component, as well as a standard language model. We also attempted to train VAEs on the text dataset but found that it was unable to learn meaningful latent representations despite tuning the latent dimension size, KL annealing, and word dropout. Refer to the appendix for a detailed description of the hyperparameters, model architecture, and training regime.

7 Experiments

7.1 Code Space Structure

As the code space we use by definition does not have the capacity to represent the entire discrete input space, ideally the autoencoder would learn to maintain valid representations for only real inputs which roughly exist along a low-dimensional manifold determined by the space of natural images or natural language sentences. This property is difficult to maintain in standard autoencoders, which often learn a partial identity mapping, but ideally should be improved by code space regularization.

We test this property by passing two sets of samples through ARAE, one of true held-out samples and the other of explicitly-noised examples existing off this manifold.

Table 1 (left) shows empirical results for these experiments. We obtain the reconstruction error (i.e. negative log likelihood) of the original
Table 1: **Left.** Reconstruction error (negative log-likelihood averaged over sentences) of the original sentence from a corrupted sentence. Here $k$ is the number of swaps performed on the original sentence. **Right.** Samples generated from AE and ARAE where the input is noised by swapping words.

| $k$ | AE | ARAE |
|-----|-----|------|
| 0   | 1.06 | 2.19 |
| 1   | 4.51 | 4.07 |
| 2   | 6.61 | 5.39 |
| 3   | 9.14 | 6.86 |
| 4   | 9.97 | 7.47 |

Table 2: **Left.** Semi-Supervised accuracy on the natural language inference (SNLI) test set, respectively using 22.2% (medium), 10.8% (small), 5.25% (tiny) of the supervised labels of the full SNLI training set (rest used for unlabeled AE training). **Right.** Perplexity (lower is better) of language models trained on the real data and synthetic samples from a GAN/AE/LM.

| Model | Medium | Small | Tiny |
|-------|--------|-------|------|
| Supervised Encoder | 65.9% | 62.5% | 57.9% |
| Semi-Supervised AE | 68.5% | 64.6% | 59.9% |
| Semi-Supervised ARAE | 70.9% | 66.8% | 62.5% |

| Data for LM | PPL |
|-------------|-----|
| Real data   | 27.4 |
| LM samples  | 90.6 |
| AE samples  | 97.3 |
| ARAE-GAN samples | 82.2 |

We note that unlike denoising autoencoders which require a domain-specific noising function \[11, 33\], the ARAE is not explicitly trained to denoise an input, but learns to do so as a byproduct of adversarial regularization.

### 7.2 Semi-Supervised Training

Next we utilize ARAE for semi-supervised training on a natural language inference task, shown in Table 2 (right). We experiment with using 22.2% (medium), 10.8% (small), 5.25% (tiny) of the original labeled training data, and use the rest of the training set for unlabeled AE training. The labeled set is randomly picked. The full SNLI training set contains 543k sentence pairs, and we use supervised sets of 120k, 59k and 28k sentence pairs respectively for the three settings. As a baseline we use an AE trained on the additional data, similar to the setting explored in \[7\]. For ARAE we use the subset of unsupervised data of length < 15, which roughly includes 653k single sentences (due to the length restriction, this is a subset of 715k sentences that were used for AE training). As observed by Dai and Le \[7\], training on unlabeled data with an AE objective improves upon a model just trained on labeled data. Training with adversarial regularization provides further gains.

**ARAE-GAN Samples**
- A woman preparing three fish .
- A man is hugging and art .
- Two man are <unk> a.
- The two boys in glasses are all girl .
- The two children are eating the balloon animal .
- A woman is trying on a microscope .
- The two dogs are running in bed .

**AE Samples**
- A woman in a cart tearing over of a tree .
- A dog are <unk> a.
- A surfer and a couple waiting for a show .
- Two kids are <unk> a man in a blue shirt .

**LM Samples**
- A man walking outside on a dirt road , sitting on the dock .
- A large group of people is taking a photo for Christmas and at night .
- Someone is avoiding a soccer game .
- The man and woman are dressed for a movie .

Figure 3: Text samples generated from ARAE-GAN, a simple AE, and from a baseline LM trained on the same data. To generate from an AE we fit a multivariate Gaussian to the learned code space and generate code vectors from this Gaussian.
A common test for a GANs ability to generate realistic samples that cover the original data space is to train a simple model on the samples from the GAN itself. Acknowledging the pitfalls of such quantitative evaluations [31], for text GANs we can do this by producing a large set of sampled sentences, and training a simple language model over the generations. For these experiments we generate 100k samples from (i) ARAE-GAN, (ii) an AE, (iii) a RNN LM trained on the same data, and (iv) the real training set. To “sample” from an AE we fit a multivariate Gaussian to the code space (of the training data) after training the AE and generate code vectors from this Gaussian and decode back into sentence space. All models are of the same size to allow for fair comparison. Samples from the models are shown in Figure 3.

We subsequently train a standard RNN language model on the generated data and evaluate perplexity on held-out real data. The language model is of the same size as the decoder of the ARAE. As can be seen from Table 2, training on real data (understandably) outperforms training on generated data by a large margin. Surprisingly however, we find that a language model trained on ARAE-GAN data performs slightly better than one trained on LM-generated/AE-generated data.

### 7.4 Interpolation and Vector Arithmetic

A widely observed property of GANs (and VAEs) is that the Gaussian prior \( p(z) \) induces the ability to smoothly interpolate between outputs by exploiting the structure of the latent space. While language models may provide a better estimate of the underlying probability space, constructing this style of interpolation would require combinatorial search, which makes this a useful feature of text GANs. We experiment with this property by sampling two points \( z_0 \) and \( z_1 \) from \( p(z) \) and constructing intermediary points \( z_\lambda = \lambda z_1 + (1 - \lambda) z_0 \). For each we generate the argmax output \( \hat{x}_\lambda \). The samples are shown in Figure 4 for text and in Figure 2 (right-bottom) for MNIST. While it is difficult to assess the “accuracy” of these interpolations, we generally qualitatively observe smooth changes in the output sentences/images as we move from one latent space to another.

Another intriguing property of image GANs is the ability to move in the latent space via offset vectors (similar to the case with word vectors [23]). For example, Radford et al. [25] observe that when the mean latent vector for “men with glasses” is subtracted from the mean latent vector for “men without glasses” and applied to an image of a “woman without glasses”, the resulting image is that of a “woman with glasses”. We experiment to see if a similar property holds for sentences.

We generate 1 million sentences from the ARAE-GAN and parse the sentences to obtain the main verb, subject, and modifier. Then for a given sentence, to change the main verb we subtract the mean latent vector \( \mu \) for all other sentences with the same main verb (in the first example in Figure 5 this would correspond to all sentences that had “sleeping” as the main verb) and add the mean latent vector for all sentences that have the desired transformation (with the running example this would be all sentences whose main verb was “walking”). We do the same to transform the subject and the modifier. We decode back into sentence space with the transformed latent vector via sampling from \( p_{\psi}(g(z + t)) \). Some examples of successful transformations are shown in Figure 5 (right).

Quantitative evaluation of the success of the vector transformations is given in Figure 5 (left). For each original vector \( z \) we sample 100 sentences from \( p_{\psi}(g(z + t)) \) over the transformed new latent vector and consider it a match if any of the sentences demonstrate the desired transformation. Match \( \% \) is proportion of original vectors that yield a match post transformation. As we ideally want the generated samples to only differ in the specified transformation, we also calculate the average word precision against the original sentence (Prec) for any match.
A man in a tie is sleeping and clapping on balloons.⇒ walking
A person is standing in the air beneath a criminal.⇒ standing
The Jewish boy is trying to stay out of his skateboard.⇒ standing
The people work in a new uniform studio.⇒ man

| Transform   | Match % | Prec |
|-------------|---------|------|
| walking     | 85      | 79.5 |
| man         | 92      | 80.2 |
| two         | 86      | 74.1 |
| dog         | 88      | 77.0 |
| standing    | 89      | 79.3 |
| several     | 70      | 67.0 |

A man in a tie is clapping and walking dogs.⇒ walking
A person is walking in the air beneath a pickup.⇒ walking
The Jewish man is trying to stay out of his horse.⇒ man
A man works in a new studio uniform.⇒ man
two children playing a head with plastic drink.⇒ two
Two workers watching baby steak with the grass.⇒ two
The boy’s babies is wearing a huge factor.⇒ baby
The dog arrives or looks into an area.⇒ dog
Three women are standing near a man walking.⇒ standing
Two dogs are standing in front of the dinner.⇒ standing
two child playing a guitar on side with a table.⇒ several
Several child playing a guitar on side with a table.⇒ several
Two workers watching baby steak with the grass.⇒ several

A baby workers is watching steak with the water.⇒ baby
The people shine or looks into an area.⇒ several
The Jewish boy is trying to stay out of his skateboard.⇒ several
The people shine or looks into an area.⇒ several

Some child head a playing plastic with drink.⇒ one
The Jewish boy is trying to stay out of his skateboard.⇒ one
The people shine or looks into an area.⇒ one
The Jewish boy is trying to stay out of his skateboard.⇒ one

A man in a tie is sleeping and clapping on balloons.⇒ one
A person is standing in the air beneath a criminal.⇒ one
The Jewish boy is trying to stay out of his skateboard.⇒ one
The people shine or looks into an area.⇒ one

Figure 5: Left. Quantitative evaluation of transformations. Match % refers to the % of samples where at least one decoder samples (per 100) had the desired transformation in the output, while Prec. measures the average precision of the output against the original sentence. Right. Examples (out of 100 decoder samples per sentence) where the offset vectors produced successful transformations of the original sentence. See Section 7.4 for methodology.

8 Conclusion
We present adversarially regularized autoencoders, as a simple approach for training a discrete structure autoencoder jointly with a code-space generative adversarial network. The model learns an improved autoencoder as demonstrated by semi-supervised experiments and analysis of the manifold structure for text and images. It also learns a useful generative model for text that exhibits a robust latent space, as demonstrated by natural interpolations and vector arithmetic. We however note that (as has been frequently observed when training GANs) our model seemed to be quite sensitive to hyperparameters. Finally, while many useful models for text generation already exist, text GANs provide a qualitatively different approach influenced by the underlying latent variable structure. We envision that such a framework could be extended to a conditional setting, combined with other existing decoding schemes, or used to provide a more interpretable model of language.

Acknowledgment
We thank Sam Wiseman, Kyunghyun Cho, Sam Bowman, Joan Bruna, Yacine Jernite, Martin Arjovsky, Mikael Henaff and Michael Mathieu for fruitful discussions. Yoon Kim is sponsored by a SYSTRAN research award. We also thank the NVIDIA Corporation for the donation of a Titan X Pascal GPU that was used for this research.

References
[1] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. arXiv:1701.07875, 2017.
[2] David Berthelot, Tom Schumm, and Luke Metz. Began: Boundary equilibrium generative adversarial networks. arXiv:1703.10717, 2017.
[3] Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. A large annotated corpus for learning natural language inference. In Proceedings of EMNLP, 2015.
[4] Samuel R. Bowman, Luke Vilnis, Andrew M. Dai Oriol Vinyal and, Rafal Jozefowicz, and Samy Bengio. Categorical Reparameterization with Gumbel-Softmax. In Proceedings of CoNLL, 2016.
[5] Tong Che, Yanran Li, Ruixiang Zhang, R Devon Hjelm, Wenjie Li, Yangqui Song, and Yoshua Bengio. Maximum-Likelihood Augment Discrete Generative Adversarial Networks. arXiv:1702.07983, 2017.
[6] Xi Chen, Diederik P. Kingma, Tim Salimans, Yan Duan, Prafulla Dhariwal, John Schulman, Ilya Sutskever, and Pieter Abbeel. Variational Lossy Autoencoder. In Proceedings of ICLR, 2017.
[7] Andrew M Dai and Quoc V Le. Semi-supervised sequence learning. In Proceedings of NIPS, 2015.
[8] Peter Glynn. Likelihood Ratio Gradient Estimation: An Overview. In Proceedings of Winter Simulation Conference, 1987.
[9] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Proceedings of NIPS, 2014.
[10] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, and Aaron Courville Vincent Dumoulin. Improved Training of Wasserstein GANs. arXiv:1704.00028, 2017.
[11] Felix Hill, Kyunghyun Cho, and Anna Korhonen. Learning distributed representations of sentences from unlabelled data. In *Proceedings of NAACL*, 2016.

[12] R Devon Hjelm, Athul Paul Jacob, Tong Che, Kyunghyun Cho, and Yoshua Bengio. Boundary-Seeking Generative Adversarial Networks. *arXiv:1702.08431*, 2017.

[13] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[14] Eric Jang, Shixiang Gu, and Ben Poole. Categorical Reparameterization with Gumbel-Softmax. In *Proceedings of ICLR*, 2017.

[15] Diederik P. Kingma, Tim Salimans, and Max Welling. Improving Variational Inference with Autoregressive Flow. *arXiv:1606.04934*, 2016.

[16] Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes. In *Proceedings of ICLR*, 2014.

[17] Matt Kusner and Jose Miguel Hernandez-Lobato. GANs for Sequences of Discrete Elements with the Gumbel-Softmax Distribution. *arXiv:1611.04051*, 2016.

[18] Jiwei Li, Will Monroe, Tianlin Shi, Sébastien Jean, Alan Ritter, and Dan Jurafsky. Adversarial Learning for Neural Dialogue Generation. *arXiv:1701.06547*, 2017.

[19] Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables. In *Proceedings of ICLR*, 2017.

[20] Alireza Makhzani and Brendan Frey. PixelGAN Autoencoders. *arXiv:1706.00531*, 2017.

[21] Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, Ian Goodfellow, and Brendan Frey. Adversarial Autoencoders. *arXiv:1511.05644*, 2015.

[22] Lars Mescheder, Sebastian Nowozin, and Andreas Geiger. Adversarial Variational Bayes: Unifying Variational Autoencoders and Generative Adversarial Networks. *arXiv:1701.04722*, 2017.

[23] Tomas Mikolov, Scott Wen tau Yih, and Geoffrey Zweig. Linguistic Regularities in Continuous Space Word Representations. In *Proceedings of NAACL*, 2013.

[24] Ofir Press, Amir Bar, Ben Bogin, Jonathan Berant, and Lior Wolf. Language Generation with Recurrent Generative Adversarial Networks without Pre-training. *arXiv:1706.01399*, 2017.

[25] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. In *Proceedings of ICLR*, 2016.

[26] Danilo J. Rezende and Shakir Mohamed. Variational Inference with Normalizing Flows. In *Proceedings of ICML*, 2015.

[27] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic Backpropagation and Approximate Inference in Deep Generative Models. In *Proceedings of ICML*, 2014.

[28] Francis Dutil Christopher Pal Aaron Courville Sai Rajeswar, Sandeep Subramanian. Adversarial Generation of Natural Language. *arXiv:1705.10929*, 2017.

[29] Stanislau Semeniuta, Aliaksei Severyn, and Erhardt Barth. A Hybrid Convolutional Variational Autoencoder for Text Generation. *arXiv:1702.02390*, 2017.

[30] Tianxiao Shen, Tao Lei, Regina Barzilay, and Tommi Jaakkola. Style Transfer from Non-Parallel Text by Cross-Alignment. *arXiv:1705.09655*, 2017.

[31] Lucas Theis, Aaron van den Oord, and Matthias Bethge. A note on the evaluation of generative models. In *Proceedings of ICLR*, 2016.

[32] Dustin Tran, Rajesh Ranganath, and David M. Blei. Deep and Hierarchical Implicit Models. *arXiv:1702.08896*, 2017.

[33] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and Composing Robust Features with Denoising Autoencoders. In *Proceedings of ICML*, 2008.

[34] Ronald J. Williams. Simple Statistical Gradient-following Algorithms for Connectionist Reinforcement Learning. *Machine Learning*, 8, 1992.
[35] Zichao Yang, Zhiting Hu, Ruslan Salakhutdinov, and Taylor Berg-Kirkpatrick. Improved Variational Autoencoders for Text Modeling using Dilated Convolutions. In *Proceedings of ICML*, 2017.

[36] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient. In *Proceedings of AAAI*, 2017.

[37] Junbo Jake Zhao, Michael Mathieu, and Yann LeCun. Energy-based generative adversarial network. *CoRR*, abs/1609.03126, 2016.
Appendix: Experiments details

MNIST experiments

- The encoder is a three-layer MLP, 784–800–400–100. The output of the encoder is normalized onto a unit ball (having $l_2$ norm 1), denoted as $c \in \mathbb{R}^{100}$ before being forwarded further. Except for the output layer, batch normalization and ReLU are used following the linear layers.
- We also add some additive Gaussian noise into $c$ which is then fed into the decoder. The standard deviation of that noise is initialized to be 0.4, and then exponentially decayed to 0.
- The decoder is a four-layer MLP, 100–400–800–1000–784. Except for the output layer, batch normalization and LeakyReLU (scale = 0.2) are used following the linear layers.
- The autoencoder is optimized by Adam, with learning rate $5 \times 10^{-4}$.
- The GAN employs a MLP generator, with structure 32–64–100–150–100. The noise vector is $z \in \mathbb{R}^{32}$.
- The GAN employs a MLP critic, with structure 100–100–60–20–1. The clipping factor $\epsilon = 0.05$. The critic is trained with 10 iterations in each GAN loop.
- The GAN is optimized by Adam, with learning rate $5 \times 10^{-4}$ on the generator, and $5 \times 10^{-5}$ on the critic.
- When updating the encoder, we multiply the critic gradient by 0.2 before backpropping to the encoder.

Text experiments

The architecture we used for the text generation task is described below:

- The encoder is an one-layer LSTM with 300 hidden units. The output of the encoder is normalized onto a unit ball (having $l_2$ norm 1), denoted as $c \in \mathbb{R}^{300}$, before being forwarded further.
- We add Gaussian noise into $c$ before feeding it into the decoder. The standard deviation of that noise is initialized to be 0.2, and then exponentially decayed every 100 iterations by a factor of 0.995.
- The decoder is a one-layer LSTM with 300 hidden units.
- The decoding process at each time step takes the top layer LSTM hidden state and concatenates it with the hidden codes $c$, before feeding them into the output (i.e. vocabulary projection) and the softmax layer.
- The word embedding is of size 300.
- We adopt a grad clipping on the encoder/decoder, with max grad norm = 1.
- The encoder/decoder is optimized by vanilla SGD with learning rate 1.
- The GAN employs a MLP generator, with structure 100–300–300, batch normalization, and ReLU nonlinearity. The noise vector $z \in \mathbb{R}^{100}$.
- The GAN employs a MLP critic, with structure 300–300–1, batch normalization, and LeakyReLU (with scale 0.2) nonlinearity. The clipping factor $\epsilon = 0.01$. The critic is trained with 5 iterations in each GAN loop.
- The GAN is optimized by Adam, with learning rate $5 \times 10^{-4}$ on the generator, and $1 \times 10^{-5}$ on the critic.
- When we update the encoder, we normalize both sources of gradients, i.e. from the critic and decoder, based on their norms. After that, a weight factor 0.01 is imposed on the critic backproped gradient.
- We increment the number of GAN training loop\(^1\) by 1 (it initially is set to 1) , respectively at the beginning of epoch #2, epoch #4 and epoch #6.
- We train for a total of 6 epochs.

\(^1\)The GAN training loop refers to how many times we train GAN in each entire training loop (one training loop contains training autoencoder for one loop, and training GAN for one or several).
Semi-supervised experiments

The architecture we used for semi-supervised learning task is described blow:

- The encoder is a three-layer LSTM, with the hidden state size being 300. The output of the encoder is normalized onto a unit ball (having $l_2$ norm 1), denoted as $c \in \mathbb{R}^{300}$, before being forwarded further.
- The decoder is a one-layer LSTM, with the hidden state size being 300.
- The decoding process at each time step takes the top layer LSTM hidden state and concatenates it with the hidden codes $c$, before forward them to a word transition matrix.
- The initial decoder hidden state is initialized with $c$, with a linear transformation.
- The word embedding is of size 300.
- We adopt a grad clipping on both LSTMs, with a maximum allowed gradient norm being 1.
- The encoder/decoder is optimized by vanilla SGD, learning rate 1.
- The GAN employs a MLP generator, with structure $100-150-300-500$, batch normalization, and ReLU nonlinearity. The noise vector $z \in \mathbb{R}^{100}$.
- The GAN employs a MLP critic, with structure $500-500-150-80-20-1$, batch normalization, and LeakyReLU (scale = 0.2) nonlinearity. The clipping factor $\epsilon = 0.02$. The critic is trained with 10 iterations in each GAN loop.
- The GAN is optimized by Adam, with learning rate $5e^{-05}$ on the generator, and $1e^{-05}$ on the critic.
- When we update the encoder, we normalize both sources of gradients, i.e. from the critic and decoder, based on their norms. After that, we multiply the critic gradients by 0.01 before backpropping to the encoder.

Note we use the same architecture for all three experiment settings (i.e. label set portion: Medium (22.2%), Small (10.8%), Tiny (5.5%)). The baseline models under comparison (Supervised Encoder, Semi-Supervised AE) use the same setting as described above.