Spring 2020

Text Anomaly Detection with ARAE-AnoGAN

Tec Yan Yap

Text Anomaly Detection with ARAE-AnoGAN

Tec Yan Yap

Illinois Wesleyan University, Bloomington IL 61701, USA
tyap@iwu.edu

Abstract. Generative adversarial networks (GANs) are now one of the key techniques for detecting anomalies in images, yielding remarkable results. Applying similar methods to discrete structures, such as text sequences, is still largely an unknown. In this work, we introduce a new GAN-based text anomaly detection method, called ARAE-AnoGAN, that trains an adversarially regularized autoencoder (ARAE) to reconstruct normal sentences and detects anomalies via a combined anomaly score based on the building blocks of ARAE. Finally, we present experimental results demonstrating the effectiveness of ARAE-AnoGAN and other deep learning methods in text anomaly detection.

1 Introduction

Anomaly detection is the task of identifying events or observations that deviate from the expected behavior. This problem has applications in a wide variety of domains such as fraud detection, network intrusion, defect detection, and health monitoring. Various approaches have been developed for anomaly detection, including deviation analysis [11], unsupervised clustering methods [3], and rule-based systems [29]. The focus of recent anomaly detection studies, however, emphasizes on the use of generative adversarial networks (GANs), as surveyed in [9]. The vast majority of these GAN-based approaches have been employed to investigate anomalies in images and time-series, and to the best of our knowledge, there is only one method that uses a GAN to detect anomalies in text [10].

One reason that these GAN-based approaches have mostly been applied to images and time-series is the fact that a GAN is designed to work with real-valued, continuous data and has difficulties in directly generating discrete sequences of tokens, such as texts [14]. A GAN works by providing real-valued gradients from a discriminator to a generator in order to guide the generator in making a slight change to the generated data to become more realistic. If the generated data is based on discrete tokens, the "slight change" guidance does not work since a slight change to the discrete generated data does not correspond to a discrete token. This is called the non-differential objective, and so far, there are three types of methods that tackle this non-differentiable objective: policy gradient methods [30], Gumbel-Softmax distribution [17], and adversarially regularized autoencoder (ARAE) [32].

In this work, we will leverage the previous work in GAN-based anomaly detection methods and GANs for discrete sequences to develop a GAN-based method to detect anomalies in text. We present ARAE-AnoGAN, a semi-supervised learning method that trains an adversarially regularized autoencoder (ARAE) to model discrete sequences of normal training data and detects anomalies in text via a combined anomaly score based on the building blocks of ARAE. Finally, we present experimental results demonstrating the effectiveness of our approach and other deep learning methods in text anomaly detection.

The rest of this paper is organized as follows. Section 2 presents an overview of the related work. Section 3 provides a background on the concepts used in ARAE-AnoGAN framework. In Section 4, we introduce our proposed ARAE-AnoGAN framework. In Section 5, we show the experimental results comparing our proposed ARAE-AnoGAN framework with three other frameworks on a dataset. Finally, Section 6 summarizes the paper and suggests possible future work.

2 Related Work

Anomaly detection in text is a challenging task and only a limited number of approaches have been developed in the past years to address this problem. [20] have demonstrated the use of an one-class classifier as well as a simple autoencoder to detect anomalies in text by training their models to learn representations of normal text. Context Vector Data Description (CVDD) [25] extends the work on one-class classifer and uses the publicly available pre-trained word embedding models, such as word2vec [21], GloVe [22], and FastText [5,15] to convert tokenized sentences to word embeddings that the one-class classfiers can use to detect anomalies in text. CVDD uses a self-attention mechanism to transform variable-length word embeddings of normal sentences to fixed-length text representations. These text representations are also trained along with context vectors, in order to capture the diverse topics within normal sentences (i.e. distinct yet non-anomalous topics).

The traditional and simplest anomaly detection method is to identify data points that deviate from common statistical properties of a distribution, such as mean, median, mode, and quantiles. Other methods, such as density-based and clustering-based techniques, group normal data points into a dense neighborhood or clusters and identify anomalies that are far away. Recent developments in Generative Adversarial Networks (GANs) [12] have sparked new GAN-based approaches to the anomaly detection problem. Anomaly detection using GANs involves learning the distribution of normal samples and using features of GANs to classify anomalies with an anomaly score. Most existing GAN-based anomaly detection methods show promising, state-of-the-art results for images [26,31,2]. Few GAN-based approaches have been developed for sequential data, and those that do exist only examine multivariate time-series data [18]. To date, there is only one report of a GAN-based anomaly detection method in text. [10] uses a GAN with the policy gradient method to model discrete sequences and detect anomalies via an anomaly score presented in AnoGAN. Our work is similar to [10], but instead of using a GAN trained with the policy gradient method, we use an autoencoder adversarially trained with a GAN to model text for anomaly detection.

3 Background

To explain ARAE-AnoGAN in detail, it is essential to briefly introduce the background of the concepts used in our framework.

3.1 Generative Adversarial Networks

GANs are a class of deep generative models that were initially introduced by [12]. GANs were originally developed to generate realistic images, but since its inception, GANs used for different functions, such as image captioning [24], style transfer [33], and video generation [8]. A GAN trains two neural networks, a generator and a discriminator, to compete with each other. Here, we will use image generation as an example to explain how GANs work. Given a training dataset of images, the overall goal of a GAN is to generate images that look like they should belong to the training dataset. To achieve this goal, the generator takes random noise as an input and generates fake images that fool the discriminator into thinking they are real. The discriminator takes in both real images from the training dataset and fake images from the generator and outputs a verdict on whether a given image is real or fake. The final verdict that the discriminator outputs when the fake images are passed to the discriminator serves as crucial information for the generator to generate realistic images. If the discriminator identifies the generator's outputs as real, it means that the generator did a good job and there should be no change to how the generator generates the images. On the other hand, if the discriminator identifies the generator's outputs as fake, it means that the generator failed to do a good job and should change the way it generates the images. During training, the generator becomes better at generating realistic images and the discriminator becomes more accurate at identifying the fake images from the real, to a point where the discriminator can no longer distinguish between the real and fake images anymore. This occurs because the generator can now generate images that look similar to the real images in the training dataset. Thus, a GAN has been successfully trained and now the generator can generate images that look similar to the images in the training set.

Other variations of GANs that have received much attention in the literature are Deep Convolutional GANs (DCGANs) [23] and Wasserstein GANs (WGANs) [4]. DCGANs are a direct extension of the GANs mentioned above, except that they use convolutional layers instead of fully connected layers in both the generator and discriminator. Convolutional layers have been shown to be wildly successful at representing images, and thus DCGANs are more suitable for generating images. GANs may suffer a type of problem called mode collapse where the generator outputs the same image over and over again. This occurs when the generator has found a way to generate images that can successfully fool the discriminator again and again while the discriminator does not get better at distinguishing the fake from the real. To mitigate the occurrence of mode collapse, the generator of WGANs minimizes the distance between the data distribution observed in the training dataset and the distribution in the generated outputs as opposed to simply trying to fool the discriminator.

3.2 Autoencoders

Autoencoders, like GANs, are a type of generative model that can generate data points similar to the data points in the training dataset. The difference between autoencoders and GANs lies in the structures and the goals each of them are achieving during the training process. An autoencoder trains two neural networks, an encoder and a decoder. Using image generation as an example, the encoder takes in an image as an input and attempts to reduce the input to a compressed encoded vector, also referred to as the latent vector. This latent vector is then passed into the decoder, which attempts to recreate, or reconstruct the input image. During training, the encoder becomes better at finding an efficient way to compress and encode the input to a latent vector, while the decoder gets better at reconstructing the input from the latent vector.

A few examples of the use cases of autoencoders include anomaly detection, dimension reduction, and image denoising. In our text anomaly detection method, we employ a discrete autoencoder that takes in discrete sequences such as text (arrays of sentences of words) rather than continuous values such as pixels in images, in order to reconstruct text. To detect anomalous sentences, we train the discrete autoencoder to only reconstruct normal sentences. A trained autoencoder can then classify an input as anomalous if the autoencoder cannot fully reconstruct the given input.

Autoencoders have their limitations as well. When training is completed, the latent vector can have a large range of values, resulting in a large amount of gaps in the latent space distribution that the decoder does not know how to reconstruct. If we pass in values that the encoder has not fed to the decoder during training, the reconstructed output will not look like data points in the training dataset. To remove these gaps in the latent vector, we can constrain the latent vector to follow a known distribution, such as the normal distribution or uniform distribution. This constraint is especially helpful for anomaly detection since an autoencoder trained on a normal dataset should not have any gaps in the latent space distribution so that the autoencoder will always be able to reconstruct normal inputs.

There are three variations of autoencoders that have been developed to constrain the latent vector to follow a known distribution: variational autoencoders [16], adversarial autoencoders [19], and adversarially regularized autoencoders [32]. In our text anomaly detection method, we use ARAE to model sequences of normal text since it is the only one out of the three variations that uses both a generator and a discriminator of a WGAN to train the autoencoder. These three building blocks of ARAE, the autoencoder, generator, and discriminator, will serve as important components for our anomaly score, which will be introduced in Section 4.4.

3.3 AnoGAN

AnoGAN [26] is a deep convolutional generative adversarial network specifically developed to detect anomalies in images. It uses DCGAN as the base model and constructs an anomaly score using the generator and discriminator to evaluate whether or not an input image is anomalous. The training process of AnoGAN involves training a DCGAN with normal images so that the generator learns to only generate images that look like they should belong to the normal dataset. If an input image is significantly different than a generated image from the trained generator, then there is a high possibility that the input image is anomalous. However, since the generator is capable of generating a variety of normal images, we need to perform an optimization process such that the generated image is as similar as possible to the input image during testing. We can then use the difference between the generated image (that was already optimized to look the closest to the input image) and the input image, along with the verdict from the discriminator on whether the input image is fake or real, to come up with an anomaly score to evaluate whether or not the input image is anomalous. The generator in this GAN-based anomaly detection method is analogous to the decoder of an autoencoder trained on normal dataset. Both the generator and decoder are capable of generating some representations of the normal images in the dataset, and anomalies are detected if input images in the testing dataset are different than the generated output of the generator or decoder.

3.4 Word Embeddings

Deep learning models take arrays of numbers as inputs. When working with text, we need a way to convert strings to numbers before feeding them to the model. One simple approach is to convert each word in the vocabulary to a one-hot encoded vector. Consider the sentence "Have a good great day." The vocabulary (or unique words) in this sentence is (a, day, good, great, have). To convert each word to a one-hot encoded vector, we will create a vector of zeros with length equal to vocabulary size, then place a one in the index that corresponds to the word. For example, the first word "a" in the vocabulary will be represented by [1,0,0,0,0] and the second word will be converted to [0,1,0,0,0]. This approach is naive and inefficient. If we think about these words in a 5-D space, where each word occupies one dimension, "good" and "great" are as different as "day" and "have," which is not true. Furthermore, imagine that we have 100,000 words in the vocabulary. The length of the one-hot encoded vector will be large, and thus, increasing the input size and the parameters that need to be trained in our deep learning model.

To overcome these problems, we can instead convert each word to a word embedding. We first construct a dictionary where each word in the vocabulary is encoded with a unique integer (called index). We then apply word embeddings to each index, which converts each index to a vector of floating point values. For example, the first word "a" in the vocabulary can be represented by $[0.4, 2.5, 2.4, 0.8]$ and the second word can be converted to [3.0, 0.1, 0.23, -0.2]. Words with similar meanings will have embeddings that are close to each other and the size of the embeddings can be set to a number smaller than the vocabulary size. Instead of specifying the values for the embedding manually, one can train an embedding layer in a deep learning model or use pre-trained word embeddings such as word2vec [21], GloVe [22], and FastText [5,15].

3.5 Long Short-Term Memory

Text is sequential in nature, as the order of words in a sentence carries information. To deal with the sequential nature of text, after each word is converted to an embedding vector, the embedding vector is passed into a Long Short-Term Memory (LSTM) [13] cell. Multiple LSTM cells make up an LSTM layer, and mulitple LSTM layer make up an LSTM network. LSTM networks are a class of recurrent neural networks (RNNs) [28] that excel in learning from sequential data such as text and time-series. One advantage of LSTMs over other types of RNNs is their ability to retain information for a long period of time, allowing for important information learned early in a sequence to remain relevant at the end of the sequence when the model makes decisions.

Unlike a traditional feed-forward neural network where we assume that all inputs (and outputs) are independent of each other, RNNs and LSTMs perform the same operation to every element of an input sequence, with the output being dependent on the previous computations. Figure 1 illustrates an unrolled RNN layer, showing the RNN cell at various time steps.

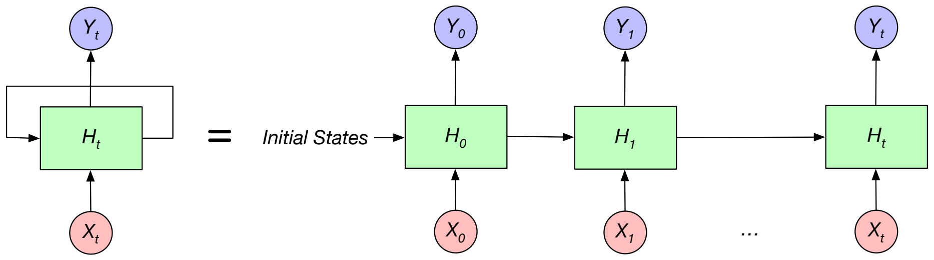

Fig. 1: Recurrent Neural Network (RNN) at various time steps.

A RNN cell first takes in some initial states and the current input. After processing these inputs with a neural network, the cell produces an output and a hidden state. The hidden state, along with the next input, are then passed as inputs to the RNN cell at the next time step, repeating this process until the RNN cell processes all inputs in the sequence. The RNN cell is the same unit at different points in time, where the same parameters are used to process each word at each time step. This is called parameter sharing and it helps in applying the RNN model to sequences of variable lengths. While reading a sequence, if the RNN model used different parameters for each step during training, it would not generalize to unseen sequences of different lengths.

An LSTM cell has a similar flow as an RNN cell but applies additional operations in order to keep relevant information for prediction and forget non-relevant data. In addition to the hidden state, a cell state is also passed to the next LSTM cell. As the cell state flows through the network, information is added to or removed from the cell state via gates. There are a total of three gates in an LSTM cell: a forget gate that decides what is relevant to keep from previous cell state, an input gate that decides what is relevant to keep from current input, and an output gate that decides what the next hidden state should be. Some applications of LSTM include question-answering [27], handwriting recognition [6], and music generation [7].

4 Text Anomaly Detection with ARAE-AnoGAN

4.1 Architecture Overview

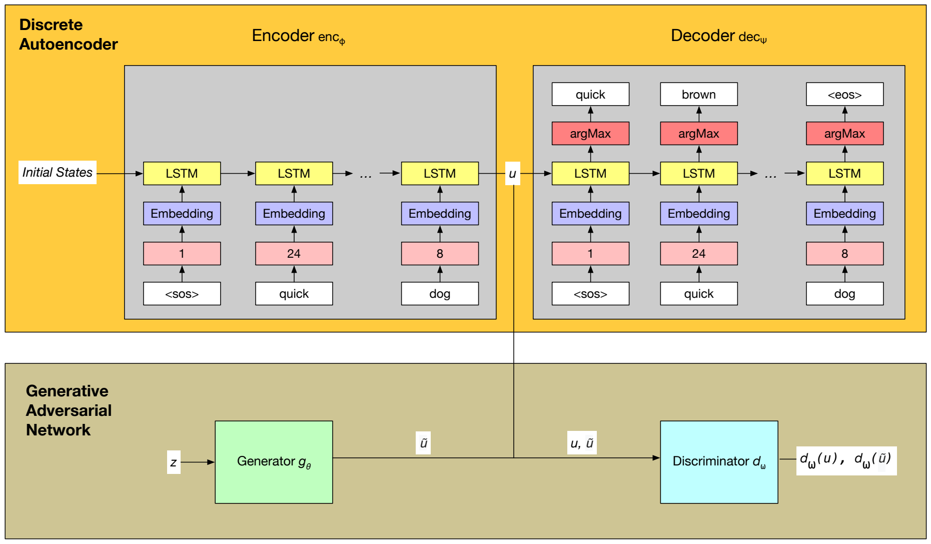

Fig. 2: ARAE architecture, composed of a discrete autoencoder and a generative adversarial network.

Figure 2 illustrates the ARAE model used in our text anomaly detection approach, which contains a discrete autoencoder (made up of an encoder and a decoder) and a generative adversarial network (made up of a generator and a discriminator). The following paragraphs provide an overview of how sentences are passed into our model and how anomalous sentences are detected.

Pre-processing Sentences in the training dataset come in variable lengths. Before passing a sentence into ARAE, we first need to pre-process the sentence. In our approach, we set a maxiumum sentence length to a fixed number (a tunable hyperparameter) and pad a sentence with zeros (`<pad>` tokens) if the length of the sentence is smaller than the maximum sentence length. A start-of-sequence token `<sos>` is added at the beginning of the input sentence for both the encoder and decoder and an end-of-sequence token `<eos>` is added at the end of the target sentence for the decoder. `<sos>` serves as an initiator for the decoder to generate subsequent tokens, and `<eos>` allows the decoder to generate variable length sentences. When this pre-processing step is completed, each word in a sentence is converted to a unique number (index) using a pre-defined dictionary where we can look-up the index that each word represents. For example, the tokenized, pre-processed input sentence `[<sos>, ‘you’, ‘are’, ‘beautiful’, <pad>, <pad>]` will be converted to a vector $[1, 5, 23, 19, 0, 0]$. The target sentence `[‘you’, ‘are’, ‘beautiful’, <eos>, <pad>, <pad>]` will be converted to $[5, 23, 19, 2, 0, 0]$. The array of indices is then passed to the embedding layer, where it generates an embedding for each word in the input sentence.

ARAE-AnoGAN Architecture Both the encoder and decoder use the LSTM architecture to process the words in a sentence. The encoder takes in an array of word embeddings and passes the last hidden state of its LSTM network as an initial state to the LSTM network of the decoder. The decoder then takes in an array of words, along with the last hidden state of the encoder, and outputs an array of words that are similar to the input sentence. Essentially, the last hidden state of the encoder LSTM network is the latent vector of an autoencoder. At each time step of the decoder, the output of the LSTM produces a vector with length equal to our vocabulary size. Each index at the output vector represents the probability that the word corresponding to the index should belong to the observation at that time step. We apply the argmax function to the output array at each time step to obtain the index at which the value in the output array is the highest to find the word that most likely belong to the observation at that time step. We can then use our dictionary to look up the word represented by the argmax index at each time step. In our text anomaly detection approach, if the decoder does not recognize the last hidden state fed by the encoder, it will not be able to output a sentence that looks similar to the sentences in the training set, which are composed of normal sentences. Thus, an anomalous sentence has been encoded, and subsequently, cannot be reconstructed by the decoder.

As mentioned in Section 3.2, we want to constrain the latent vector of an autoencoder (in this case, the hidden state of the LSTM encoder) to follow some known distribution. In this case, it is constrained to learn the distribution of the generator, which generates outputs based also on a known distribution. During training of ARAE, the discriminator treats the last hidden state of the encoder as real and the output of the generator as fake. After ARAE is trained, the generator should be able to generate hidden states that look similar to hidden states of the encoder.

We then perform the optimization process described in AnoGAN for each sentence in the testing dataset such that the output of the generator is the closest to the real hidden state. Finally, we calculate an anomaly score for each sentence in the testing set, as a linear combination of an autoencoder reconstruction loss, an encoder loss, and a discriminator loss. A high anomaly score indicates that the trained model is not capable of representing the sentence, hence, an anomaly. A low anomaly score indicates a similar sentence has been seen during training, hence, a normal sentence. More details on the anomaly score will be provided in Section 4.4.

4.2 Problem Definition & Notation

The formal definition of the text anomaly detection problem is as follows:

Given a training dataset $\mathcal{D}$ with m normal sentences ($\mathcal{D} = \{x^{(i)}\}_{i=1}^{m}$ where $x^{(i)}$ is the ith sentence in $\mathcal{D}$) and a testing dataset $\tilde{\mathcal{D}}$ with n normal and anomalous sentences ($\tilde{\mathcal{D}} = \{(\tilde{x}^{(i)}, y^{(i)})\}_{i=1}^{n}$ where $y^{(i)} = 0$ indicates ith sentence in $\tilde{\mathcal{D}}$ as normal and $y^{(i)} = 1$ indicates ith sentence in $\tilde{\mathcal{D}}$ as anomalous), the goal is to model $\mathcal{D}$ with ARAE and detect anomalies in $\tilde{\mathcal{D}}$ with an improved version of the anomaly score $A(\tilde{x})$ presented in AnoGAN [26], which will be introduced in Section 4.4. A high anomaly score $A(\tilde{x})$ indicates possible anomalous text within $\tilde{\mathcal{D}}$. A threshold τ can be chosen to predict the class rather than the probability, where $A(\tilde{x}) > \tau$ indicates an anomaly.

Define $\mathcal{X}$ to be a set of discrete sequences (e.g. tokenized sentences) from training dataset $\mathcal{D}$ with m normal text over a vocabulary $\mathcal{V}$. An **encoder** with parameters ϕ is the first part of the discrete autoencoder that learns a function $enc_{\phi} : \mathcal{X} \mapsto \mathcal{U}$ that maps the input space to code space (hidden state). A **decoder** with parameters ψ is the second part of the discrete autoencoder that learns a function $dec_{\psi} : \mathcal{U} \mapsto \hat{\mathcal{X}}$ that maps the code space to reconstructed space. The decoder is essentially a conditional decoder $dec_{\psi}(x|u)$ where it generates each word based on the previous words with LSTM. In the GAN part of our model, a **generator** g with parameters θ maps random variable z sampled from a probability distribution (typically Gaussian or uniform) to the code space. A **discriminator** d with parameters ω maps code space to a single scalar that represents the probability that the code space comes from $\mathcal{X}$ (real) rather than $\mathcal{Z}$ (fake). Table 1 summarizes the components of our model framework.

Having defined our overall architecture, we now move on to discuss how our proposed ARAE-AnoGAN framework is trained and used to detect anomalies in text.

Network	Function
Encoder	$enc_\phi(x) = u$
Decoder	$dec_\psi(x\|u) = \hat{x}$
Generator	$g_\theta(z) = \tilde{u}$
Discriminator	$d_\omega(u)/d_\omega(\tilde{u})$

Table 1: Summary of networks and notations

4.3 Pipeline

The full pipeline is shown in Algorithm 1. First, as proposed in [32], the model is trained with gradient descent across: (1) the encoder and decoder to minimize reconstruction, (2) the discriminator, and (3) the encoder and generator adversarially. When ARAE training is completed, we perform an optimization process for each sentence $\tilde{x}$ in the testing dataset $\tilde{\mathcal{D}}$ such that hidden state generated by the generator is most similar to the hidden state of the encoder. During the optimization process, we keep the parameters ϕ, ψ, θ, and ω unchanged but add a new input layer to the generator with the same size as z. The parameters of this new input layer are the only trainable weights during the iteration process. This process will run for $\mathit{\Gamma}$ steps, in the end yielding $z_{\mathit{\Gamma}}$, which the generator uses to generate $\tilde{u}$ that is most similar to the real encoded vector u. We can then use $z_{\mathit{\Gamma}}$ in our anomaly score for each sample in $\tilde{\mathcal{D}}$.

Algorithm 1: ARAE-AnoGAN Pipeline

```
(a) Training
for each training epoch do:
    (1) Train the encoder and decoder for reconstruction
    Sample $\{x^{(i)}\}_{i=1}^m$ from training dataset $\mathcal{D}$ and compute $u^{(i)} = enc_\phi(x^{(i)})$
    Backprop loss $-\frac{1}{m}\sum_{i=1}^m x^{(i)} log(dec_\psi(x^{(i)}|u^{(i)})$
    (2) Train the discriminator
    Sample $\{x^{(i)}\}_{i=1}^m$ and $\{z^{(i)}\}_{i=1}^m \sim \mathcal{N}(0,1)$
    Compute $u^{(i)} = enc_\phi(x^{(i)})$ and $\tilde{u}^{(i)} = g_\theta(z^{(i)})$
    Backprop loss $-\frac{1}{m}\sum_{i=1}^m d_\omega(u^{(i)}) + \frac{1}{m}\sum_{i=1}^m d_\omega(\tilde{u}^{(i)})$
    (3) Train the encoder and generator adversarially
    Sample $\{x^{(i)}\}_{i=1}^m$ and $\{z^{(i)}\}_{i=1}^m \sim \mathcal{N}(0,1)$
    Compute $u^{(i)} = enc_\phi(x^{(i)})$ and $\tilde{u}^{(i)} = g_\theta(z^{(i)})$
    Backprop loss $\frac{1}{m}\sum_{i=1}^m d_\omega(u^{(i)}) - \frac{1}{m}\sum_{i=1}^m d_\omega(\tilde{u}^{(i)})$
(b) Testing
for each sample $\{\tilde{x}^{(i)}\}_{i=1}^n$ in testing dataset $\tilde{\mathcal{D}}$:
    for $1, 2, ..., \mathit{\Gamma}$ do:
        (4) Map code space to noise space
        Add a new input layer to the generator $g_\theta$
        Keeping $\phi, \psi, \theta$, and $\omega$ unchanged,
        backprop loss $\alpha\mathcal{L}_{res} + \beta\mathcal{L}_{enc} + (1-\alpha-\beta)\mathcal{L}_{disc}$
```

4.4 Anomaly Score

We propose to utilize the discrete autoencoder, generator, and discriminator that have been trained jointly to represent normal sentences for text anomaly detection. Expanding on the work of [26] (AnoGAN), our approach consists of the following three parts:

1. **Reconstruction Loss**
   Since the discrete autoencoder has been trained to reconstruct normal sentences, it will output a similar normal sentence if the input sentence is normal. An anomalous sentence cannot be represented by this trained discrete autoencoder, resulting in a high reconstruction loss, defined as:

$$\mathcal{L}_{res}(\tilde{x}^{(i)}, \phi, \psi) = -\tilde{x}^{(i)} \cdot log(dec_{\psi}(\tilde{x}^{(i)} | enc_{\phi}(\tilde{x}^{(i)}))) \tag{1}$$

2. **Encoder Loss** The generator is trained to generate an hidden state $\tilde{u}$ that looks similar to the hidden state of the encoder when a normal sentence is encoded. After the optimization process has been performed as described in Section 4.3, the generator will generate a $\tilde{u}$ that has values closest to u. If, however, an anomalous sentence has been passed to the encoder to produce u, the generator will fail to minimize the distance between $\tilde{u}$ and u even after the optimization process has been performed. Thus, we can define an encoder loss as follows:

$$\mathcal{L}_{enc}(\tilde{x}^{(i)}, z_{\Gamma}, \phi, \theta) = \left\| enc_{\phi}(\tilde{x}^{(i)}) - g_{\theta}(z_{\Gamma}) \right\|_1 \tag{2}$$

3. **Discriminator Loss** As suggested by [26] (AnoGAN), we use only the output of an intermediate layer instead of the single scalar output of the discriminator for our discriminator loss. As mentioned in Section 3.1, the discriminator can no longer distinguish between the real and fake once the GAN is successfully trained. The intermediate layer, however, preserves feature representation of the real data points. Thus, we can define a discriminator loss as the L_1 distance between the feature representation of the real encoded vector u and the feature representation of the generated encoded vector $\tilde{u}$:

$$\mathcal{L}_{disc}(\tilde{x}^{(i)}, z_{\Gamma}, \phi, \theta, \omega) = \left\| f_{\omega}(enc_{\phi}(\tilde{x}^{(i)})) - f_{\omega}(g_{\theta}(z_{\Gamma})) \right\|_1 \tag{3}$$

   where $f_{\omega}(\cdot)$ is the output of an intermediate layer of the discriminator d_{ω}.

Hence, for a test sample $\tilde{x}^{(i)}$, the anomaly score $A(\tilde{x}^{(i)})$ is defined as the weighted sum of the three components:

$$A(\tilde{x}^{(i)}) = \alpha \mathcal{L}_{res} + \beta \mathcal{L}_{enc} + (1 - \alpha - \beta) \mathcal{L}_{disc} \tag{4}$$

To evaluate $A(\tilde{x}^{(i)})$ with other test samples, we first compute $A(\tilde{x}^{(i)})$ for all $\{\tilde{x}^{(i)}\}_{i=1}^{n}$ in testing dataset $\tilde{\mathcal{D}}$, yielding a set $\mathcal{S} = \{s_i = A(\tilde{x}^{(i)}) : \tilde{x}^{(i)} \in \tilde{\mathcal{D}}\}$. We then apply normalization to scale the anomaly scores to have values in the range of [0,1], following the method proposed by [2].

$$s'_i = \frac{s_i - min(S)}{max(S) - min(S)} \tag{5}$$

This equation yields a set of normalized anomaly scores S', which will be used to evaluate the performance of our anomaly detection method.

5 Experiments

We compared the performance of ARAE-AnoGAN with other existing frameworks quantitatively on the *Reuters-215781*[1] dataset. Since anomaly detection is a classification problem, the evaluation metrics we used are based around the number of anomalies correctly identified.

5.1 Setup

The *Reuters-215781* dataset was originally collected and labeled by Carnegie Group, Inc. and Reuters, Ltd. and has been widely used for text classification research. Each document comes with one or more related topics. In our experiment, we only consider documents that contain fewer than 30 words (since ARAE has only been shown to work successfully with short sentences) and have exactly one topic with at least 100 samples for our training and testing dataset. Even though the maximum sentence length is 30, we found that the average sentence length in both the training and testing dataset to be around 8. We also pre-process the dataset by converting text to lowercase and removing punctuation, numbers, and stopwords using the stop words list from the `nltk` Python library. In every anomaly detection experiment, we treat one topic as anomalous with the rest as normal and only train the model with the respective normal data. Then, we evaluate the test data, where normal samples are labeled $y = 0$ and anomalous samples are labeled $y = 1$.

5.2 Implementation Details

In order to understand more about the underlying structures of ARAE (originally implemented in PyTorch[2]), we implemented our ARAE-AnoGAN framework from scratch in Tensorflow 2.0 [1] instead of using the existing implementation. For both the encoder and decoder, we used embedding layers of size 300 and LSTM networks with depth 1 and size 300 for the hidden states. The generator is a feed-forward neural network with 3 fully connected layers of size 100, 300, and 300, respectively. The discriminator is also a feed-forward neural network with 3 fully connected layers of size 300, 300, and 1, respectively. The size of

[1] `https://daviddlewis.com/resources/testcollections/reuters21578`
[2] `https://github.com/jakezhaojb/ARAE`

the noise vector is 100, corresponding to the first layer of the generator. Full details on the model implementation and hyperparameters can be found at this project's repository[3].

5.3 Baselines

One way to measure the performance of our text anomaly detection framework is to compare it with other frameworks. We consider three models to compare with our framework. The first one is a simple discrete autoencoder, whose structure is similar to the discrete autoencoder used in our ARAE-AnoGAN approach. The second model is ARAE, where the discrete autoencoder is trained adversarially with a GAN. For these two models, we only consider the reconstruction loss $\mathcal{L}_{res}$ when calculating the anomaly score. The third and last model is called Context Vector Data Description (CVDD) [25], a one-class classification method that uses pre-trained word embeddings for anomaly detection on text.

5.4 Evaluation Metrics

In a classification problem, predicting probabilities of an observation belonging to each class is more flexible than predicting classes directly. This flexibility comes from the way that probabilities may be interpreted using different thresholds that allow the model to optimize different metrics, such as the number of false positives compared to the number of false negatives. For this experiment, we utilize two common machine learning metrics, precision and recall, to evaluate and compare our model with other baselines. Precision is defined as the ratio of the number of true positives divided by the sum of the true positives and false positives, which indicates the proportion of predicted anomalies that are truly anomalous.

$$Precision = \frac{TP}{(TP + FP)} \tag{6}$$

Recall, on the other hand, is defined as the ratio of the number of true positives divided by the sum of the true positives and the false negatives, which indicates the proportion of actual anomalies that are correctly classified.

$$Recall = \frac{TP}{(TP + FN)} \tag{7}$$

Precision and recall have an inverse relationship - improving precision typically reduces recall and vice versa. Thus, to fully evaluate the performance of a model, we construct a precision-recall curve, which is a plot of the precision (y-axis) and the recall (x-axis) for different thresholds. The model with the greatest area under the precision-recall curve (also known as average precision) has the highest overall precision and recall, and thus, performs the best.

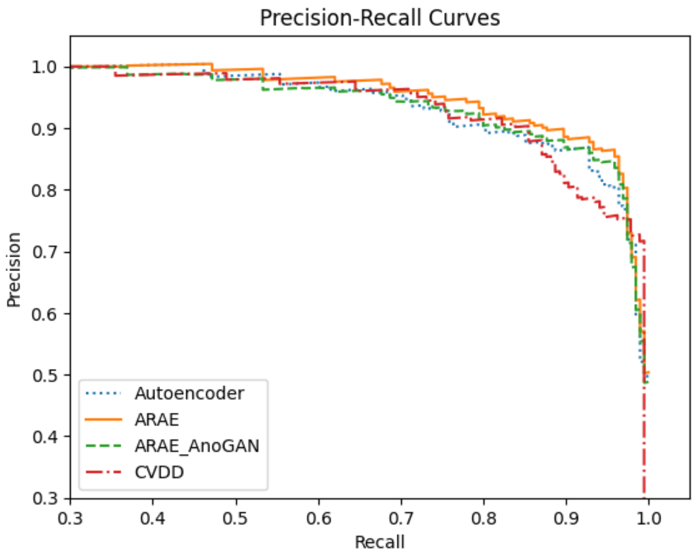

Fig. 3: Precision vs. recall plot of each framework.

Method	Average Precision
Autoencoder	0.9493
ARAE	**0.9532**
CVDD	0.9485
ARAE-AnoGAN	0.9502

Table 2: Average precision of each framework.

5.5 Results

Figure 3 illustrates the precision-recall curve and Table 2 shows the average precision achieved by each framework in our experiment. We see that ARAE achieves the best average precision, with ARAE-AnoGAN following after. One possible reason that ARAE-AnoGAN did not outperform the rest of the frameworks could stem from not having the best set of hyperparameters to successfully train the GAN in ARAE. As noted by the developers of ARAE, ARAE is "quite sensitive to hyperparameters." Hyperparameters such as network size and learning rates could be tuned to yield higher average precision for ARAE-AnoGAN. Another possible reason is that the weights α and β in the anomaly score have not been optimized. In our experiment, α and β were arbitrarily set to 0.6 and 0.2 respectively. Future work could involve a hyperparameter search and an optimization of α and β to yield the best results.

Even though ARAE-AnoGAN did not outperform all other frameworks, we can see that in Figure 4, ARAE-AnoGAN does have the ability to assign low

[3] https://github.com/tedyap/ARAE-AnoGAN

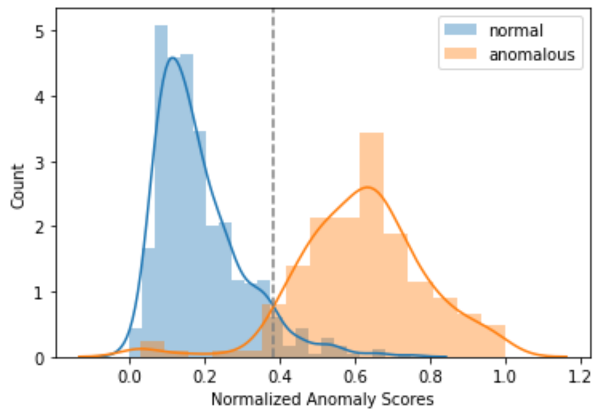

Fig. 4: Histogram of normalized anomaly scores from ARAE-AnoGAN for both normal and anomalous sentences.

anomaly scores to normal sentences and high anomaly scores to anomalous sentences. According to Figure 4, if we set the threshold τ to around 0.38, ARAE-AnoGAN will be able to successfully identify most of the anomalous sentences.

6 Conclusion

In this paper, we have explored the possibility of using GANs for text anomaly detection. We introduce ARAE-AnoGAN, a GAN-based approach that utilizes a discrete autoencoder, a generator, and a discriminator trained on normal sentences to detect anomalous sentences using an anomaly score.

Since this is an early adoption of GANs for text anomaly detection, there are many interesting issues that are worth investigating. For instance, no one has yet utilized a GAN with the Gumbel-Softmax distribution to model normal text and detect anomalies using an anomaly score. Another area worth exploring is to develop a new model architecture to allow computing the anomaly score without the computationally expensive Γ optimization process. Other future work could involve training a deeper ARAE to model more complex and longer discrete structures (e.g. documents).

Acknowledgements

I would like to thank Dr. Mark Liffiton for his guidance and assistance as my research advisor for this project. I would also like to thank Dr. Andrew Shallue, Dr. Brad Sheese, and Dr. Daniel Roberts from Illinois Wesleyan University for serving on my thesis committee. In addition, I would like to thank the people who provided tutorials and wrote articles to help me understand some of the concepts used in this project.

References

1. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: Large-scale machine learning on heterogeneous systems (2015), https://www.tensorflow.org/, software available from tensorflow.org
2. Akcay, S., Atapour-Abarghouei, A., Breckon, T.P.: Ganomaly: Semi-supervised anomaly detection via adversarial training. In: Asian Conference on Computer Vision. pp. 622–637. Springer (2018)
3. Angiulli, F., Pizzuti, C.: Fast outlier detection in high dimensional spaces. In: European Conference on Principles of Data Mining and Knowledge Discovery. pp. 15–27. Springer (2002)
4. Arjovsky, M., Chintala, S., Bottou, L.: Wasserstein GAN. arXiv preprint arXiv:1701.07875 (2017)
5. Bojanowski, P., Grave, E., Joulin, A., Mikolov, T.: Enriching word vectors with subword information. Transactions of the Association for Computational Linguistics 5, 135–146 (2017)
6. Carbune, V., Gonnet, P., Deselaers, T., Rowley, H.A., Daryin, A., Calvo, M., Wang, L.L., Keysers, D., Feuz, S., Gervais, P.: Fast multi-language lstm-based online handwriting recognition. International Journal on Document Analysis and Recognition (IJDAR) pp. 1–14 (2020)
7. Choi, K., Fazekas, G., Sandler, M.: Text-based lstm networks for automatic music composition. arXiv preprint arXiv:1604.05358 (2016)
8. Clark, A., Donahue, J., Simonyan, K.: Efficient video generation on complex datasets. arXiv preprint arXiv:1907.06571 (2019)
9. Di Mattia, F., Galeone, P., De Simoni, M., Ghelfi, E.: A survey on GANs for anomaly detection. arXiv preprint arXiv:1906.11632 (2019)
10. Drozdyuk, A., Eke, N.: Anomaly detection with generative adversarial networks and text patches https://norberte.github.io/assets/pdf/GAN%20Project%20Report.pdf, accessed: 2020-04-26
11. Ghoting, A., Parthasarathy, S., Otey, M.E.: Fast mining of distance-based outliers in high-dimensional datasets. Data Mining and Knowledge Discovery 16(3), 349–364 (2008)
12. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. In: Advances in neural information processing systems. pp. 2672–2680 (2014)
13. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation 9(8), 1735–1780 (1997)
14. Huszár, F.: How (not) to train your generative model: Scheduled sampling, likelihood, adversary? arXiv preprint arXiv:1511.05101 (2015)
15. Joulin, A., Grave, E., Bojanowski, P., Mikolov, T.: Bag of tricks for efficient text classification. arXiv preprint arXiv:1607.01759 (2016)
16. Kingma, D.P., Welling, M.: Auto-encoding variational bayes. arXiv preprint arXiv:1312.6114 (2013)
17. Kusner, M.J., Hernández-Lobato, J.M.: GANs for sequences of discrete elements with the gumbel-softmax distribution. arXiv preprint arXiv:1611.04051 (2016)

18. Li, D., Chen, D., Jin, B., Shi, L., Goh, J., Ng, S.K.: MAD-GAN: Multivariate anomaly detection for time series data with generative adversarial networks. In: International Conference on Artificial Neural Networks. pp. 703–716. Springer (2019)
19. Makhzani, A., Shlens, J., Jaitly, N., Goodfellow, I., Frey, B.: Adversarial autoencoders. arXiv preprint arXiv:1511.05644 (2015)
20. Manevitz, L.M., Yousef, M.: One-class svms for document classification. Journal of machine Learning research 2(Dec), 139–154 (2001)
21. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: Advances in neural information processing systems. pp. 3111–3119 (2013)
22. Pennington, J., Socher, R., Manning, C.D.: GloVe: Global vectors for word representation. In: Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). pp. 1532–1543 (2014)
23. Radford, A., Metz, L., Chintala, S.: Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434 (2015)
24. Reed, S., Akata, Z., Yan, X., Logeswaran, L., Schiele, B., Lee, H.: Generative adversarial text to image synthesis. arXiv preprint arXiv:1605.05396 (2016)
25. Ruff, L., Zemlyanskiy, Y., Vandermeulen, R., Schnake, T., Kloft, M.: Self-attentive, multi-context one-class classification for unsupervised anomaly detection on text. In: Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics. pp. 4061–4071 (2019)
26. Schlegl, T., Seeböck, P., Waldstein, S.M., Schmidt-Erfurth, U., Langs, G.: Unsupervised anomaly detection with generative adversarial networks to guide marker discovery. In: International conference on information processing in medical imaging. pp. 146–157. Springer (2017)
27. Wang, D., Nyberg, E.: A long short-term memory model for answer sentence selection in question answering. In: Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers). pp. 707–712 (2015)
28. Werbos, P.J.: Backpropagation through time: what it does and how to do it. Proceedings of the IEEE 78(10), 1550–1560 (1990)
29. Wong, W.K., Moore, A., Cooper, G., Wagner, M.: What's strange about recent events (wsare): An algorithm for the early detection of disease outbreaks. Journal of Machine Learning Research 6(Dec), 1961–1998 (2005)
30. Yu, L., Zhang, W., Wang, J., Yu, Y.: SeqGAN: Sequence generative adversarial nets with policy gradient. In: Thirty-First AAAI Conference on Artificial Intelligence (2017)
31. Zenati, H., Romain, M., Foo, C.S., Lecouat, B., Chandrasekhar, V.: Adversarially learned anomaly detection. In: 2018 IEEE International Conference on Data Mining (ICDM). pp. 727–736. IEEE (2018)
32. Zhao, J., Kim, Y., Zhang, K., Rush, A.M., LeCun, Y.: Adversarially regularized autoencoders. arXiv preprint arXiv:1706.04223 (2017)
33. Zhu, J.Y., Park, T., Isola, P., Efros, A.A.: Unpaired image-to-image translation using cycle-consistent adversarial networks. In: Proceedings of the IEEE international conference on computer vision. pp. 2223–2232 (2017)