Generating Text like an Author with GANs
Final Report
Group 14

Eda Ayan
Bilkent University, Turkey
Email: eda.ayan@ug.bilkent.edu.tr
Student ID: 21801891

Eduin E. Hernandez
NYCU, Taiwan
Email: eduin.ee08@nycu.edu.tw
Student ID: 22104094

Umut Utku Erdem
Bilkent University, Turkey
Email: utku.erdem@bilkent.edu.tr
Student ID: 22004099

***Abstract*—Generative Adversarial Networks (GANs) are a two system model that pits a generator against a discriminator in a zero-sum game with the aim to mutually improve their performance so that at the end of the training, the generator can produce high quality data. However, despite its success for image generation, GANs face many limitations for natural language generation. In this paper, we evaluate the performance of SeqGAN for text generation, a network that solves many of limitations of previous GANs in the discrete domain, and compare it with character-based RNNs, its precursor in the text generation task. Specifically, we train these models on a given author's works so that these models are able to generate a similar text. We also employ text classifier and BLEU metrics to evaluate these generated text.**

I. Introduction

Natural language generation (NLG) is a sub-field of natural language processing (NLP), that focuses on building systems that can produce coherent and readable text [1]. NLG is commonly considered a general term which encompasses a wide range of tasks that take a form of input such as a natural language prompt and output a sequence of text that is coherent and understandable by humans. Traditionally, researchers focus on generating text using Recurrent Neural Networks (RNN) [2]. RNNs seem to have the ideal structure for text generation problems due to the high-dimensional hidden states and non-linear relations that enable them to remember and process prior knowledge to predict future information [3]. But RNNs are vulnerable to slight modifications in data that were not introduced in the learning process, resulting in mispredictions and poor performance [4] [5].

As an alternative for the generation tasks, Goodfellow et al. [6] proposed the Generative Adversarial Networks (GANs). It is a two model system with a generator and a discriminator that provide feedback to each in order to improve their mutual performance. GANs have showed a great success in Computer Vision and it is mainly used for image construction. GANs are designed for real and continuous data, so using GANs for text generation is problematic since data is discrete in text generation problem. Therefore, using GANs in this field is an active research topic and there are different approaches in the literature to deal with this issue.

The evaluation of the the generated prompts is important in terms of assessing the information's quality and comparing with the prompts created by humans. However, evaluation of the output is challenging because the task of generating a text is open ended. Human evaluation is still the gold standard for almost all NLG tasks [7].

In our work, we implement a text generating model using GANs, which in our scenario, attempts to create a text with a similar writing style to the author the model was trained on. The purpose of doing so is to create a model that can write monologues, dialogues, plays, or even books like the original author. For this work, we mainly use Sequence Generative Adversarial Nets, a model proposed in [8]. We have trained six different instances of our model, one for each author and we have generated texts from each model. We used character-based RNN (char-RNN) [3] as a benchmark in our work because i) it is a predecessor to GANs and ii) it is not restricted by the vocab size, but rather the number of characters-types. We also constructed a SVM-based classifier to distinguish the author of the generated texts from the proposed model, which serves as simplification of the human evaluation task. We observe the performance of NLG models using text classifiers to predict the likelihood of the generated text belonging to a specified author compared to the rest. This is an idea borrowed from the Inception Score used for evaluating GANs [9]. We also use the Bilingual Evaluation Understudy (BLEU) score metric as method to evaluate the quality of the text.

This paper is divided as follows: In Sec. II we further explain the backgrounds of text classification, RNN for NLG tasks, working principle of GANs, and GANs for NLG; Sec. III, the details on the dataset employed, the metrics for evaluating the NLG text, and our NLG models; Sec. IV, the key results of our experiments, along with the samples of generated text; and finally, Sec. V, our final thoughts on this work.

II. Related Work

A. Natural Language Processing

Natural Language Processing tackles the problem of creating an interaction between humans and machines through natural language. In the most widely-cited survey of NLG methods to date [10], NLG is described as "the subfield of artificial intelligence and computational linguistics that is concerned with the construction of computer systems that can produce understandable texts in English or other human languages from some underlying non-linguistic representation of information".

Here are the relevant literature for NLG before the introduction of GANs:

1) Text Classification: Text classification has always been a widely studied subject in NLP due to the very large amount of text documents that we deal with daily [11]. Text classification aims to classify a given text under a given category, which is mostly commonly a topic or genre. While topic based classification classifies the text according to the discussed subject or content, genre based classification is more focused on how a text was created, how it was edited, the register of language it uses, and the audience it addresses. Some well-known genres are scientific articles, news reports, movie reviews, and advertisements.

Text classification can be done using either supervised or unsupervised learning methods. In supervised learning, an initial dataset, where each document is assigned to one or more category is needed. In the case where only one category is assigned to each document is called Hard Classification, assigning more than one category is called Ranking Classification [12].

Text classification is similar to other machine learning classification tasks with a special emphasis put on the representation of the data. Often, size of the documents can reach tens of thousands, most of the words that don't give much information about the desired category of the text in accordence with Zipf's word frequency law. Thus, size of the corpus is usually decreased using either feature selection or feature extraction [13].

Then machine learning algorithms are applied on the data. This is the most important step in text classification. In depth knowledge about the data and different models is required to choose the classifier. Some of these algorithms are Naïve Bayes Classifier, Logistic Regression, Support Vector Machines, and Deep Neural Networks. Some algorithms have been proven to perform better in Text Classification tasks and are more often used: such as Support Vector Machines [12].

The last step is to evaluate the results in order to understand how a model performs. Some evaluation methods for supervised learning are accuracy calculation, F Score, Matthews Correlation Coefficient (MCC), receiver operating characteristics (ROC), and area under the ROC curve (AUC) [13].

2) RNN for Text Generation: A Recurrent Neural Network adapts a similar structure to a Deep Neural Network (DNN) that enables modelling the sequential data. As can be seen in Fig. 1, at each timestep, the input received by RNN is used to update the values of the hidden states, which are used to make predictions [3].

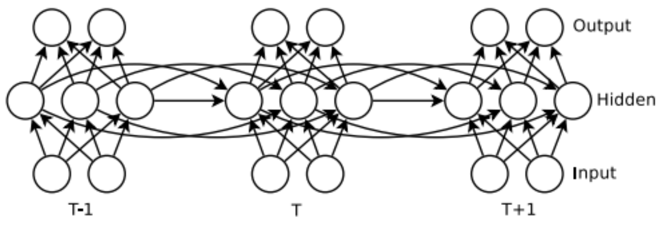

Fig. 1: Architecture of a Recurrent Neural Network. Image from [3]

The nonlinear activation functions used in the hidden states enable RNN to express a large variety of information. Information is iterated many times in the large number of hidden states in an RNN and create a rich dynamic.

Given a set of input vectors, $(x_1, ..., x_T)$ and hidden states $(h_1, ..., h_T)$, at each timestep, RNN makes the following computations to obtain a sequence of outputs $(o_1, ..., o_T)$,

$$h_t = tanh(W_{hx}x_t + W_{hh}h_{t-1} + b_h) \tag{1}$$

$$o_t = W_{oh}h_t + b_o \tag{2}$$

where W_{hx} represents the weight matrix between the inputs and the hidden states, W_{hh} represents the weight matrix between hidden layers, and W_{oh} represents the weight matrix between hidden states and the outputs. Vectors b_h and b_o are the biases at each hidden state and output.

In the case of character-level language modeling and text generation, the training sequence $(x_1, ..., x_T)$ are characters and the goal is to obtain a sequence of predictive distributions using the sequence of output vectors $(o_1, ..., o_T)$ associated with each input.

The predictive distribution can be formalized as follows:

$$P(x_{t+1}|x_{\leq t}) = softmax(o_t) \tag{3}$$

where the softmax distribution is defined by

$$P(softmax(o_t) = j) = exp(o_t^{(j)})/\Sigma_k exp(o_t^{(k)}). \tag{4}$$

RNNs learns by trying to maximize the total log likelihood of the training sequence $\Sigma_{t=0}^{T-1} logP(x_{t+1}|x_{\leq t})$, so what RNN actually learns is a probability distribution over sequences [3].

Gradients of the RNN can be computed using backpropagation through time, and computing the gradients that way is actually very cheap and easy [3]. But in practice, what was observed is that gradient decays exponentially as it is backpropagated through time and leads to a problem that is formalized by [14] as "vanishing/exploding gradients problem". This problem created two main arguments against the usage of RNNs for text generation. First, RNNs can't learn long-range temporal dependencies with gradient descent and second, highly variant gradients makes the learning very unstable [3].

To deal with the problem of learning long-range temporal structures, idea of ”memory” is added to the RNN models and “Long-Short Term Memory” (LSTM) models are introduced [14]. For datasets that require long term memorization, LSTM performs better than RNN [3].

B. Generative Adversarial Networks

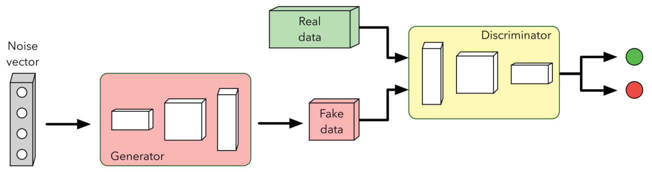

Fig. 2: Standard architecture of a Generative Adversarial Network. Image from [6].

1) GANs: GANs were first introduced in [6] as a novel method to train generative models. The framework, as in Fig. 2, consists of two models: a generative model G that learns the real data distribution and a discriminative model D that learns to determine whether the sample is from G or the real data distribution. The goal of D is to maximize the probability of assigning the correct label for both the training samples and generated samples while G tries to minimize D guessing the generated samples correctly, hence the adversarial nature of the network. The loss is defined as follows,

$$min_G max_D V(D,G) = \mathbb{E}_{x \sim p_{data}(x)}[log(D(x))] + \mathbb{E}_{z \sim p_z(z)}[log(1 - D(G(z))] \quad (5)$$

where z is the input noise variable introduced into G to generate the samples and x the training samples from the data distribution. The authors warn that although D and G should be trained simultaneously, D should be updated k times before G is updated once, with the intention of avoiding over-fitting and maintaining D near the optimal solution. If, however, the generated samples are poor like in early training, D can reject them with high confidence saturating the second part of (5). As such, maximizing $log(D(G(z))$ can provide better gradients for training G.

The GAN was trained on MNIST, CIFAR10, and Toronto Face dataset for the image generation task and evaluated using the gaussian parzen window-based log-likelihood estimates.

2) Conditional GANs: In [15], the authors argue that there is no control on the data being generated using GANs, as such they propose conditioning both G and D on auxiliary information. This information could be the class label, segments of an image, text, etc. The input of G would receive both the input noise z as in [6] and the auxilary information y combined in joint hidden representations. D would receive the samples, whether the generated or real, along with auxilary information. As such, the loss function in (5) becomes,

$$min_G max_D V(D,G) = \mathbb{E}_{x \sim p_{data}(x)}[log(D(x|y))] + \mathbb{E}_{z \sim p_z(z)}[log(1 - D(G(z|y))]. \quad (6)$$

Their model was trained on the MNIST for image generation and MIR Flickr for image tag generation. Although for MNIST experiment, the images weren’t realistic enough, the authors point out that this is just a proof of concept showing its potential usage. For MIR Flickr, the model shows an interesting performance generating tags that can easily describe the images, including the synonyms.

3) Improving GANs Training: In [9], the authors propose techniques for encouraging the convergence in GAN Training and a metric for assessing the image quality.

In the two-player game of GANs, there must be an equilibrium between both of their respective loss functions. Unfortunately, any modification in model G or D can cause an increase in loss of the other, thus failing to converge at any point. To encourage the convergence despite this issue, the following techniques are proposed: Feature Matching, Minibatch Discrimination, Historical Averaging, One-sided Label Smoothing, and Virtual Batch Normalization.

In feature matching, the authors argue that the instability of the GAN might come from G trying to match the data distribution without having any guarantee of achieving it. Therefore, to prevent G from over-training on D, we can instead train it to match the expected value of the feature of an intermediate layer of D. This entices G to train on the features that are more discriminative on the real data versus generated. The loss for D is redefined as,

$$||\mathbb{E}_{x \sim p_{data}} f(x) - \mathbb{E}_{z \sim p_z f(G(z))}||_2^2 \quad (7)$$

where f is the intermidiate feature of D. As for Minibatch Discrimination, G may collapse to a parameter setting where it emits the same point. D can pick this up and learn this point comes from G, however gradient descent is unable to separate the identical outputs, causing the learning for G to stagnate. As such, having the D look into multiple samples in combination instead of in isolation, could potential avoid the collapse of G.

For the intention of setting an image quality metric similar to human annotators, the authors propose using the Inception model on the generated images to get the conditional label distribution $p(y|x)$. Well generated images should have conditional label distribution with low entropy since the probability is biased towards one label. As a whole, since a variety of images are generated, the marginal $\int p(y|x = G(z)))dz$ with provide a high entropy. The metric is as follows,

$$exp \mathbb{E}_x[KL(p(y|x)||p(y))] \quad (8)$$

which they coin Inception Score (IS). Exponential terms are used since they are easier to compare.

Using the Inception model introduces a semi-surpervised setting for GANs. To separate real image from generated, an extra class label is considered for the generated images. Thus, classification goes from K to $K+1$ labels. The losses become,

$$L_{supervised} = -\mathbb{E}_{x,y \sim p_{data(x,y)}}[log(p_{model}(y|x, y < K + 1))] \quad (9)$$

and

$$L_{unsupervised} = -\mathbb{E}_{x \sim p_{data(x)}}[log(1 - p_{model}(y = K+1|x))] \\ -\mathbb{E}_{x \sim G}[log(p_{model}(y = K+1|x))] \quad (10)$$

where (9) and (10) are decomposition of (5).

The techinques and metrics proposed were evaluated using MNIST, CIFAR-10, SVHN, and ImageNet for image generation. The results provide visually and quantitatively better results. Interestingly enough, using the feature matching techniques provide a higher IS score, yet using the Bach Discrimination provide visually better images than Feature Matching. It is important to note that we shouldn't rely too much on IS, since it is just a rough guide to evaluate the models.

C. Generative Adversarial Networks in Natural Language Generation

GANs have achieved huge success in creating images. On the other hand, using GANs for text generation proves to be a challenging task. GANs are formally generated for continuous data, therefore training GANs for discrete data generation is an active research topic. Due to non-differentiable nature of discrete data, the problem can be solved by considering Reinforcement Learning (RL) methods or moving the problem to the continuous space. There are several GANs that are implemented for text generation and most of them are based on RL algorithms. In SeqGAN [8], text generation problem is modeled as sequential decision-making progress. In MaliGAN [16], training with maximum-likelihood objective is proposed to decrease the gradient variance. Some of the other RL-based GANs include RankGAN [17], LeakGAN [18], and MaskGAN [19].

Other GANs that are not RL-based make an approximation on discrete data or moves the problem to the continuous plane. Some of the non-RL-based GANs for NLP are TextGAN [20], FM-GAN [21], and RelGAN [22].

1) SeqGAN: GANs have limitations for discrete scenarios. The main reason behind these limitation lies in that G tries to pass the gradient update to D, but this process is challenging for discrete data. Another limitation is that D can only analyze the complete sequence and balancing the *current/future* scores for a partially generated sequence becomes a difficult problem. In [8], these two main problems of GANs are addressed and Sequence Generative Adversarial Nets (SeqGAN) is proposed.

The SeqGAN architectures consists of a RNN in G and a Convolutional Neural Network (CNN) in D. D is trained with both the real and generated data via Stochastic Gradient Descent (SGD), like typical D in GANs, but G is trained using policy gradient [23], a RL technique. Monte Carlo (MC) search is employed and intermediate action is fed by the generated reward signal by D. The Interactive process between G and D is shown in Fig. 3 [8]. To test the performance of the proposed model, both synthetic and real data experiments are conducted. For synthetic data experiments the randomly initialized LSTM (oracle) is used. Oracle is considered as the

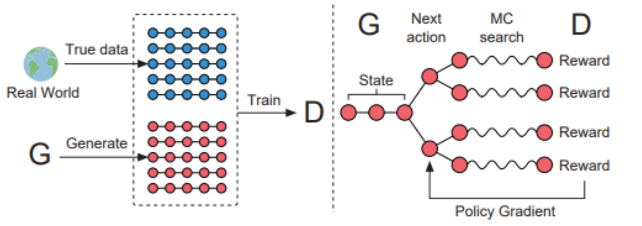

Fig. 3: Sequence Generation Framework Illustration. Image from [8].

human observer in synthetic data tests and an evaluation metric called the negative log likelihood ($\text{NLL}_{\text{oracle}}$) is used to measure the performance of the model. Results are compared with the Maximum Likelihood Estimation (MLE) trained LSTM, Policy Gradient with BLEU, and Scheduled Sampling models [24]. According to the synthetic data experiments, SeqGAN can provide better performance than other models with correct training strategy. The stability of the model depends on some specific parameter choices.

For real-word data experiments, three main tasks are tested. At first, text generation of the model is tested by generating some Chinese poems. Secondly, speech language generation is performed by using Barack Obama's speeches. Lastly, music generation is done by using the proposed model. BLEU score is used as an evaluation metric for all of the tasks.

2) RelGAN: Relational Generative Adversarial Networks (RelGANs) are proposed for text generation problem and it consists of three parts: i) a relational memory based G which is able to model long distance dependencies; ii) Gumbel-Softmax relaxation [25], [26] which is used to train GANs on discrete data; and iii) a D with various embedded representations to supply more information to G. The ability of controlling the trade-off between quality and diversity is one of the most important advantage of RelGAN. A specific tunable parameter is used to control the trade-off.

Most of the GAN's models that are implemented for text generation are constructed using LSTM for generator architecture [14]. However, LSTM has limitation in creating long-distance dependencies due to its working mechanism. Therefore, relational memory architecture for G is used in RelGAN [27].

There are a fixed number of memory slots in relational memory architecture and interactions between memory slots is allowed according to the *self-attention* mechanism [28]. Relational memory architecture provides an ability for longer sentence generation. Self-attention mechanism between memory slots (M_t) is shown in Fig. 4 [22] where x_t denotes new observation at time t, $Q_t^{(h)}$, $K_t^{(h)}$, $V_t^{(h)}$ represent the queries, keys and values. Concatenation operation of memory slot and new observation is denoted as *concat* in Fig. 4.

To overcome non-differentiability problem, Gumbel-Softmax relatation is applied. This process is composed of two main parts. In the first part, sampling can be calculated

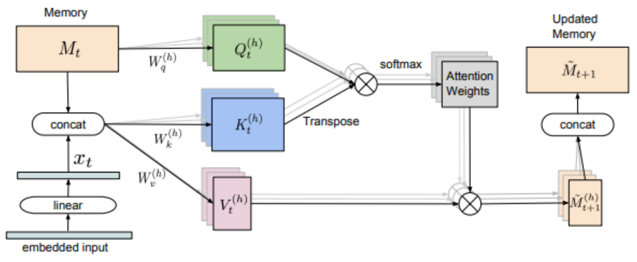

Fig. 4: Self-attention mechanism for RelGAN. Image from [27].

by using Gumbel-Max trick as follows,

$$y_{t+1} = onehot(argmax_{1\leq i\leq V}(o_t^{(i)} + g_t^{(i)})) \tag{11}$$

where y_{t+1} denotes next generated token, V represents the vocabulary size, $o_t \in \mathcal{R}^V$ are the i-th entry of output logits of G, and $g_t^{(i)}$ is standard Gumbel distribution. In the second part, discreteness is relaxed by replacing $onehot(argmax(.))$ expression with $softmax$ $(\sigma(.))$ operator. The resulting expression can be read as

$$\hat{y}_{t+1} = \sigma(\beta(o_t + g_t)) \tag{12}$$

where $\hat{y}_{t+1}$ is differentiable with respect to o_t. To control the weights between quality and diversity, an adjustable parameter β is used. Larger β results in better sample diversity and smaller β results in better sample quality. Thanks to tunable β parameter, referred as *inverse temperature*, RelGAN approach provides flexibility in terms of diversity and quality.

The D for text generation process commonly relies on CNN-based classifier [29]. However, in RelGAN, multiple embedded representations for each sentence are used and every individual representation passed over CNN classifier. The score of each operation is stored and the average of the scores are used for generation of the guide information for updating G. The main motivation behind this approach is that the extensiveness of the guide information can be increased by using different representations of each sentence. The proposed D architecture is shown in Fig. 5 [22].

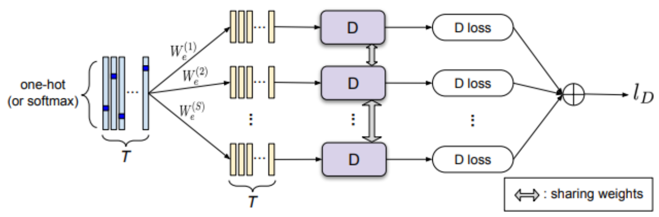

Fig. 5: Discriminator Architecture for RelGAN. Image from [27].

Performance of RelGAN is measured based on both synthetic and real data. For synthetic data, sample quality and sample diversity are measured by using negative log likelihood (NLL) for generated sentence and real sentence distribution. For real data, again NLL is used for sample diversity calculation. To measure sample quality for real data, BLEU score is used. Proposed model for different *inverse temperature* parameters is compared with state-of-the-art models such as LeakGAN [18], RankGAN [17], SeqGAN [8] and MLE. According to the performance evaluations in [22], RelGAN shows promising results compared to the other models.

III. Methods

A. Dataset

In our work, we have selected six authors to train our models. We have chosen famous English authors for the English text generation task with the main purpose of training various models so each can generate random text like the author they were trained on, i.e., a Shakespeare-trained model would output Shakespearean text while an Austen-trained model Austen-like text. We avoided using authors whose works are translated into English, since it is no longer just their writing style and word choices, but rather an amalgamation between theirs and the translators. As such, we limited our scope to native English authors. Our dataset was obtained through the Project Gutenberg [30]. We have collected several works from various authors. Each author's corpora consists approximately of 6 MB, which we split into a training dataset of nearly 4.5 MB and 1.5 MB for the test set. We avoided cutting off works when doing the split so that if "Work A" was present in the training set, it wouldn't be in the test set. The author and the works used are provided in Tab I.

After acquiring the works listed in Tab. I, manual preprocessing is applied to the collected data. We removed the watermarks of Gutenberg Project, located at the top and at the bottom of each text. Also, we remove the table of contents and any unnecessary text content such as chapter, volume, image captions, links, and title information from each work. In doing so, we remove any unnecessary aspects that does not represent the author's writings in terms of sentences. At the end, we obtained the processed corpora that can be used as an input for training text classifiers and NLG models.

B. Text Classifiers and BLEU Score

Before we delve into the specific architectures we employed for the NLG task, we present the various performance metrics used for the evaluation of the NLG models and the reasons behind their usage. Firstly, we have used text classifier models to compare the performance of NLG models in terms of the generated texts. This evaluation is done to determine if the NLG models were able to properly learn the distribution of the vocabs of each author. If the NLG model is able to learn well, we should be seeing a high accuracy in the classification task, otherwise the predictions should become spread out on various authors. However, this is assuming that the trained classifier model works well enough on the real data. This, like the Inception Score, is just a guideline and shouldn't be heavily relied for reasons further explained in Sec. IV.

There are different classifiers in the literature and to select the best classifier for our dataset, we have borrowed and modified 7 different classifiers from [11] and compared their performances in terms of f-1 scores and storage memory. The imple-

Authors	Author Corpora
A.C.Doyle	1-A Study in Scarlet 2-The Adventures of Sherlock Holmes 3-The Great Boer War 4-The Hound of the Baskervilles 5-The Lost World 6-The Return of Sheclock Holmes 7-The Sign of the Four 8-The White Company
C.Dickens	1-A Tale of Two Cities 2-Great Expectations 3-Little Dorrit 4-Oliver Twist 5-The Pickwick Papers
G.Eliot	1-Adam Bede 2-Daniel Deronda 3-Middlemarch 4-Salis Marner 5-The Mill on the Floss
H.G.Wells	1-Tales of Space and Time 2-The First Men in the Moon 3-The Invisible Man 4-The Island of Doctor Moreau 5-The New Machiavelli 6-The Sleeper Awakes 7-The Time Machine 8-The War in the Air 9-The War of the Worlds 10-The World Set Free
J.Austen	1-Emma 2-Lady Susan 3-Mansfield Part 4-Northanger Abbey 5-Persuasion 6-Pride and Prejudice 7-Sense and Sensibility
W.Shakespeare	1-All's Well That Ends Well 2-Antony and Cleopatra 3-As You Like It 4-Coriolanus 5-Cymbeline 6-Hamlet 7-Henry V 8-Henry VI 9-Julius Caesar 10-King John 11-Much Ado About Nothing 12-Othello 13-Richard II 14-The Merchant of Venice 15-The Winter's Tale 16-Twelfth Night

Table I: Author Corpora. List of authors and their respective works used in the dataset mentioned in Sec. III-A

mented classifiers are Support Vector Machine (SVM), Multilayer Perceptron (MLP), Multinomial Naive Bayes (MNB), Random Forest (RF), Rocchio, K-Nearest Neighbour (KNN) and Decision Tree (DT). For the case of MLP, we used a 3 layer MLP with hidden units sizes of 512 trained using the Adam optimizer with a learning rate of 0.01 for 12 epochs. For all the classifier models, we have trained them on the entire training dataset of 6 authors with the following constrictions: i) the corpora of each author is split into batches of paragraphs and ii) the minimum input length must be of 200 characters. The reason behind this input restriction is because training on the entire corpora of an author easily leads to overfitting and not enough samples would be produced for the training of the classifiers. If the input were a few words, then the classifiers would suffer of low accuracy due to overlapping words used by various authors. So instead we focus on paragraphs samples with a minimum size of 200 characters, roughly 30 to 50 words and converted to lowercase letters. These inputs are then converted into sparse vectors using token count, normalized using term frequency- inverse document frequency (tf-idf) to reflect how important a word is in the corpus, and then feed into the classifier which predicts to whom the text belongs.

Just like authors in [8], [27], we use the BLEU score as another measure to evaluate the quality of the generated text. BLEU is mainly a score used to evaluate machine translated text from human translation. This is done by comparing the reference data to the translation using n-grams, with score ranging from 0 and 1, with values closer to 1 signifying a closer resemblance to the reference data. The reference data is generally small, however in [8], [27], the entire train corpora is used as the reference data to compare the generated samples. In our case we use n-grams of 2, 3, 4, and 5 as an inspection on the quality of the generated samples since with the text classification task, only uni-grams are evaluated. With the BLEU score we should be able to determine if the generated sample looks "machine" or "human generated". For the BLEU score, we use the training datatset of each author as the reference data used to evaluate the test dataset and the generated samples from the NLG models.

C. Char-RNN

We have used character-based RNN (Char-RNN) as a benchmark in our work to compare its performance with SeqGAN. As mentioned briefly in the Sec. I, we employ it as benchmark because it is a precursor to GANs, it is not restricted to the vocabs size used by the current text generative models, and additionally to validate that our author-text classifiers works on the generated text, the last of which proved more convenient since Char-RNNs were faster to train than SeqGAN. In Char-RNN, we feed a RNN model with huge quantity of texts corresponding to a single author. The model finds a distribution for the probability of next character according to previous hidden layer information and former characters. A Char-RNN example model with four dimensional input and output layers is provided in Fig. 6.

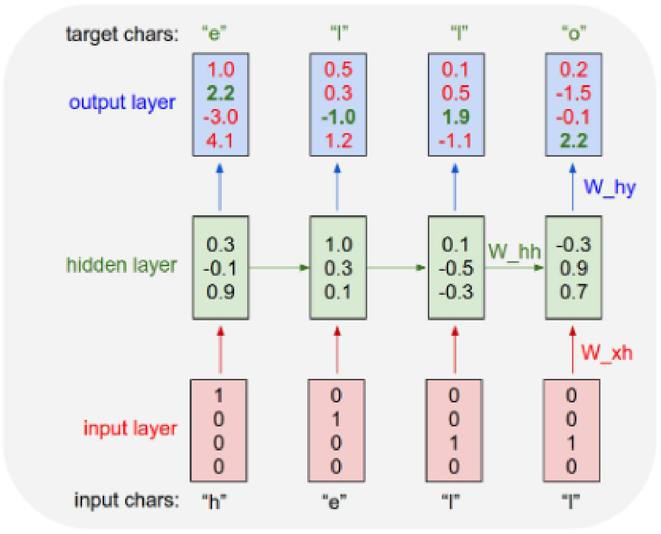

Fig. 6: Character-based RNN Example

To use char-RNN as a Languge Model, we have borrowed and modified the works from [31], a PyTorch implementation of character-based RNN models. In particular, for each author we train a char-RNN consisting of an embedding layers, followed by a Gated Recurrent Unit (GRU) of two layers with hidden units size of 100, and a decoding layers made from a linear unit. Each model is trained for the span of 2000 epochs using Cross Entropy classification loss for the next character prediction, Adam optimizer with a learning of 0.01, batch size of 100, and a sequence generation length of 300 characters.

D. *SeqGAN*

We have chosen to use SeqGAN as our text generative GAN since it solved two of the previous limitations of GANs: i) GANs are designed to generate real and continuous data, not discrete and ii) the loss or score is passed to the generator for the entire sequence in GAN models, so assessing partially generated sequence is problematic for the discriminator. SeqGAN solves these problems by regarding the generator model as an agent of Reinforcement Learning. The generator is modeled by a LSTM while the discriminator is modeled by a CNN.

To construct and train SeqGAN models, we relied on the TextGAN repository [32]. TextGAN is a PyTorch framework for GAN-based text generation models. In this work, important GAN-based text generation models such as SeqGAN [8], RelGAN [27], LeakGAN [18], MaliGAN [16] and more models have been implemented. However, these implementations are based on synthetic data. Therefore, to train the model based on collected author corpora, we have modified the implemented models. Just like with Char-RNN from Sec. III-C, we have trained one SeqGAN model for each of the six authors.

For the implemented SeqGAN model, an embedding layer, LSTM, and a linear decoding layer are used for the generator and an embedding layer, CNN, fully connected layer for the discriminator. Both embedding layers take a vector of the size of the vocabs and output the feature size of 32 for the generator and 64 for the discriminator. The LSTM for the generator uses a hidden unit size of 32 and the CNN for the discriminator relies on 12 layer 2D-Convolutions with gradually increasing kernel sizes. Both the generator and discriminator are trained using Adam optimizer, but with a 0.01 learning rate for the generator and 0.0001 for the discriminator. The loss for the discriminator is calculated from the Cross Entropy Loss for pre-training and adversarial traing phase. For the generator, the pre-training loss is NLL while the adversarial loss is calculated from the Policy Gradient using the reward provided by the discriminator for each sentence. The pre-training phase involves 120 pre-training epochs for the generator followed by 5 pre-training epochs for the discriminator. The adversarial training process consists of 1200 epochs (recommended), but training a model for 5 epochs on an RTX 3060 GPU on one author took $\sim$26 hours. Since we had to train one model on each author, the time to train all the models would take over a month. Therefore, we have limited adversarial training procedure to 5 epochs. Additionally, this SeqGAN model uses lowercase-vocabs for the sequence mapping.

E. *Overall System*

The overall system consists of two type of NLG models and one cherry picked text classifier. We first train various text classifier on the train dataset and based on the performance on the test dataset, we pick the best performing text classifier model for evaluating NLG models. Separately, we train each NLG model. For each NLG model type (char-RNN and SeqGAN) we train one instance of the model on each of the authors. In total, there are 12 trained NLG models: 6 char-RNN models and 6 SeqGAN models. After the training is completed for any NLG model, samples are generated and labeled according to the author the NLG model was trained on. After all models have generated their samples, the classifier predicts which of the authors the samples belong to and the BLEU score evaluates how well the generated sample represents the the reference data of their respective author. The overall block diagram of our work is given in Fig. 7.

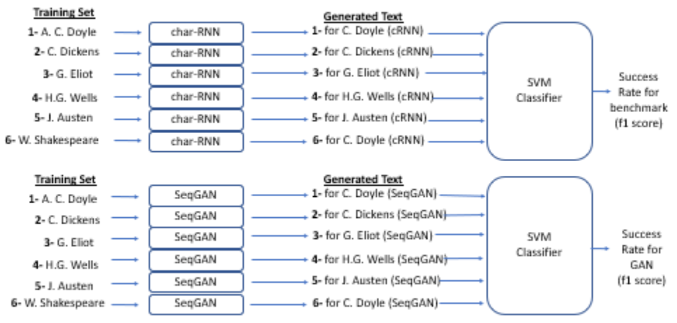

Fig. 7: Overall Block Diagram

IV. Results

In Sec. III, we have explained the procedure for the implementation of the project and the parameters used. As mentioned in Sec.III-B, one of the most important concept of

Models	Accuracy	F1-Score	Precision	Recall	Memory (MB)
SVM	73%	73%	76%	73%	3.59
MLP	70%	71%	72%	70%	181
MNB	56%	52%	70%	56%	5.82
RF	52%	51%	64%	52%	239
Rochio	48%	49%	53%	48%	3.59
KNN	44%	42%	49%	44%	27.1
DT	36%	36%	37%	36%	2.39

Table II: Test result comparison of different classifiers after training on all 6 authors. The models are ordered in descending f1-score.

Text Samples	Accuracy	F1-Score
Authors' Test Set	73%	73%
Char-RNN Samples	96%	96%
SeqGAN Samples	98%	98%

Table III: Classification Accuracy and F1-Score for Various Text Samples

the project is to select the most appropriate classifier for our dataset. Therefore we compared 7 different classifiers in terms of various metrics: overall accuracy, f1-score, precision, recall, and storage memory. The results of their performance on the authors' test dataset are given in Tab. II.

Since SVM classifier has the highest f1-score and relatively less memory requirement compared to the other classifiers, we have chosen SVM as our classifier model. Note that we could have further explored other training parameters and other deep learning models for the text classification tasks, but the performance of the SVM was sufficient enough for our interests. The generated sample size from the NLG models is relatively small ($\sim$30 kb) in comparison to the test size ($\sim$1.5 MB) since we do not generate entire books, so the classification accuracy should be higher.

According to the restrictions set for the paragraph separations and 200 minimum character threshold, 10157 paragraphs samples are used in the evaluation from the authors' test set corpora. The confusion matrix from SVM predictions are provided in Fig. 8.

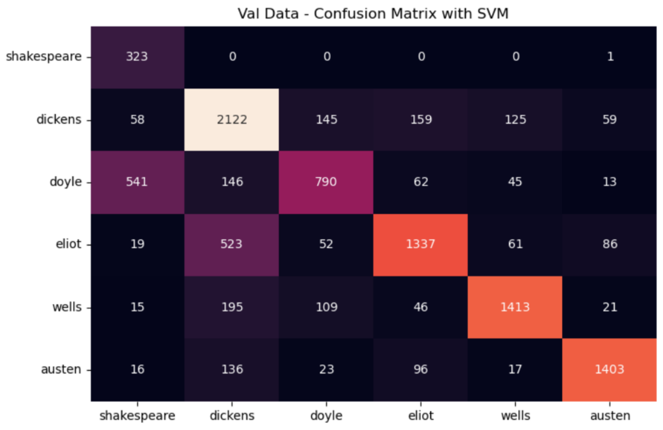

Fig. 8: Confusion matrix with SVM for the test set. X-axis is the prediction label and y-axis the ground truth.

From the matrix, we notice that Shakespeare contains small number of sample in comparison to the rest of the authors since they've been dropped due to the earlier restriction set. Many of the classifications of the model lie on mislabeling Doyle's works as Shakespeare's and Eliot's works as Dicken's. According to Fig. 8 and Tab. III, the task has a passing grade, albeit barely. If however, we chose to reduce the char-threshold limit to 100, the performance of the model drastically drop to below 50%, which is to be expected.

Text Samples	2-gram	3-gram	4-gram	5-gram
Authors' Test Set	74%	48%	27%	16%
Char-RNN Samples	40%	15%	8%	5%
SeqGAN Samples	39%	16%	8%	5%

Table IV: BLEU Score for Various Text Samples

To measure the similarity between test set and training set we have used BLEU score for 2-gram, 3-gram, 4-gram, 5-gram scenarios and resulting scores are provided in Tab. IV. For n-gram of 2, we receive a score of 74%, which shows a high resemblance with the training set, but with increasing n-gram sizes, the score value decreases. We use the scores for the test set as a benchmark to determine the quality of the generated samples from the NLG models: relatively close score would indicate high quality text.

From the same restrictions set to the earlier authors' test set, 336 paragraphs are evaluated from the character-based RNN generated text. The confusion matrix from the SVM Classifier predictions for character-based RNN model's generated text is given in Fig. 9.

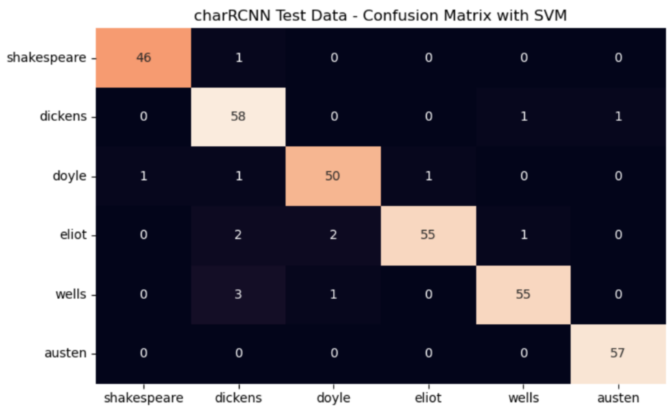

Fig. 9: Confusion matrix with SVM for charRNN generated text. X-axis is the prediction label and y-axis the ground truth.

As it can be observed from Fig. 9, most of the paragraphs that are generated by char-RNN are classified correctly despite the model not storing the vocabs of the authors. According to Tab. III, the classification task has improved significantly for char-RNN, but note again that this is due to the small sample size in comparison to the authors' test set. However, the BLEU-score in Tab. IV shows a significant drop, which indicates a lower quality text.

Some of the char-RNN generated text are as follows,

> "ANTONIO:
> Methen must a plain; and then between your eye:
> Who? who can state a country with bosopher the request

That stronged is the seven in his brows plots Of last tall."

-Shakespeare (Char-RNN Generated Sample)

"No doubt a bicycles. All about to high alone decent so to east in the prepared out and less very song because into my distations against white, and is delight they were to expled to anny been all the man, in its with two away by life at her limited out."

-H.G Wells (Char-RNN Generated Sample)

"I should not have done, I am a little station all at your ladyship of the suffered than we come in the draspized by see him away from a look before at least to be following a great going to asside too much rather defect, very moment. In might have enough, my dear holdir."

-Austen (Char-RNN Generated Sample)

"Why could not be so fireted as not gown, the youngest lools of the criminal details which was a glance at the lands and at this strange might astonished upon the grim. On the gallar was wants away to go out the end with the bust or a rest upon a brigade of the otherwist good struced the dog man on the thought not open some of their red."

-Doyle (Char-RNN Generated Sample)

Notice that from a far, the generated text almost looks human written, but if read, it no longer feels like so due to the content and syntax, hence why we use another metric like BLEU score to evaluate this behaviour.

Similarly, we evaluate the generated samples from SeqGAN using a total of 178 paragraph samples following the restrictions set with the authors' test set. The confusion matrix with SVM classification prediction are provided in Fig. 10.

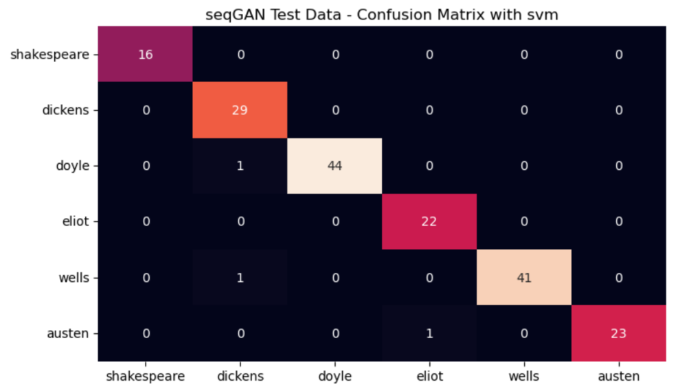

Fig. 10: Confusion matrix with SVM for SeqGAN generated text. X-axis is the prediction label and y-axis the ground truth.

From the matrix, most of the generated texts are classified correctly. Like with Char-RNN, SeqGAN classification performance for the generated text is high according to Tab. III. BLEU scores, provided in Tab. IV, indicate that compared to the character-based RNN model, there is no improvement for the NLG task. This might be related to the limitation we stated in Sec. III-D of not being able to train the model to its full extent since the adversarial training is time consuming. Some of the SeqGAN generated text are as follows,

"antony:
like our men so like a a guest; yet at this lady.
great deeds of their bottles than,
then of impart, his grace i will become in thee.
diomed , abhorred villain divine, officious and lowly dry on
about to bid me go, duke i solemnly compounded."

-Shakespeare (SeqGAN Generated Sample)

"the last i had no hand, "a certain devil was a sitting their progress that touched his time. for a moment are unhampered by temperate tramp. and he found them it is evident, this dealer on an obvious deformity was three faint' , offices, i saw beyond the martians had already dismay thought, and from the red weed be much when i had seen accord."

-H.G Wells (SeqGAN Generated Sample)

"happiness to say; and that she might confess that elizabeth though i can not know that you were so? i hope never summer in the garden of preference may wish to have best elder, was not merely false just professions of family she said."

-Austen (SeqGAN Generated Sample)

"the latter of one occasion, at once round. you have satisfied us in the road, and then bounding in a queer convulsion. on that morning called. he had allowed him even personally 1st in it, watson, to spare one who has to lure the loyal europeans."

-Doyle (SeqGAN Generated Sample)

Like Char-RNN, content and syntax-wise, the generated sample are of poor quality shown from the sample text and the BLEU score.

V. Discussion and Conclusion

We were unable to see any significant improvements in using SeqGAN for the NLG task, but this is debatable as we significantly reduced the adversarial training. For a fair evaluation of SeqGAN's performance, it would be crucial to train it fully. Unfortunately, this would require more computational power, more than we are currently able to provide and a more optimized implementation of the code. We intended to train a RelGAN model for NLG, but due to time constraints and the time required to train SeqGAN, we had to abandon the idea.

From comparing all the classification results, we have determined that using a text classifier is not a reliable metric for evaluating NLG's model's performance. Despite indicating that the choice of vocab is correct, it does not help indicate how well structured is the generated text. Likewise with BLEU score, we can determine if a text is machine or human generated if the sample size is small. However, this score is also tied to the size of the input. We experimented with increasing the generated text size from 30 KB to 0.5 MB, which increased the score up to 70%.

If we've managed to find a reliable metric for evaluating the generated text in terms of syntax and content, we could have used it to guide SeqGAN during the adversarial training stage by introducing it as another reward in the Policy Gradient. In this case, it would be rewarding the generator for better syntax and content.

We conclude this investigation by providing our trained models, code, and pre-processed dataset which are all readily available at our github repo.

REFERENCES

[1] A. Belz and E. Reiter, "Comparing automatic and human evaluation of nlg systems," in *11th conference of the european chapter of the association for computational linguistics*, 2006, pp. 313–320.

[2] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur, "Recurrent neural network based language model." in *Interspeech*, vol. 2, no. 3. Makuhari, 2010, pp. 1045–1048.

[3] I. Sutskever, J. Martens, and G. E. Hinton, "Generating text with recurrent neural networks," in *ICML*, 2011.

[4] B. Biggio and F. Roli, "Wild patterns: Ten years after the rise of adversarial machine learning," *Pattern Recognition*, vol. 84, pp. 317–331, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320318302565

[5] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[6] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in neural information processing systems*, vol. 27, 2014.

[7] G. H. de Rosa and J. P. Papa, "A survey on text generation using generative adversarial networks," *Pattern Recognition*, vol. 119, p. 108098, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320321002855

[8] L. Yu, W. Zhang, J. Wang, and Y. Yu, "Seqgan: Sequence generative adversarial nets with policy gradient," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 31, no. 1, 2017.

[9] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," *Advances in neural information processing systems*, vol. 29, 2016.

[10] E. Reiter and R. Dale, "Building applied natural language generation systems," *Nat. Lang. Eng.*, vol. 3, pp. 57–87, 1997.

[11] K. Kowsari, K. Jafari Meimandi, M. Heidarysafa, S. Mendu, L. Barnes, and D. Brown, "Text classification algorithms: A survey," *Information*, vol. 10, no. 4, p. 150, 2019. [Online]. Available: https://github.com/kk7nc/Text_Classification

[12] M. Ikonomakis, S. Kotsiantis, and V. Tampakas, "Text classification using machine learning techniques." *WSEAS transactions on computers*, vol. 4, no. 8, pp. 966–974, 2005.

[13] G. Zu, W. Ohyama, T. Wakabayashi, and F. Kimura, "Accuracy improvement of automatic text classification based on feature transformation," in *Proceedings of the 2003 ACM Symposium on Document Engineering*, ser. DocEng '03. New York, NY, USA: Association for Computing Machinery, 2003, p. 118–120. [Online]. Available: https://doi.org/10.1145/958220.958242

[14] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[15] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *arXiv preprint arXiv:1411.1784*, 2014.

[16] T. Che, Y. Li, R. Zhang, R. D. Hjelm, W. Li, Y. Song, and Y. Bengio, "Maximum-likelihood augmented discrete generative adversarial networks," *arXiv preprint arXiv:1702.07983*, 2017.

[17] K. Lin, D. Li, X. He, Z. Zhang, and M.-T. Sun, "Adversarial ranking for language generation," *Advances in neural information processing systems*, vol. 30, 2017.

[18] J. Guo, S. Lu, H. Cai, W. Zhang, Y. Yu, and J. Wang, "Long text generation via adversarial training with leaked information," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[19] W. Fedus, I. Goodfellow, and A. M. Dai, "Maskgan: better text generation via filling in the_," *arXiv preprint arXiv:1801.07736*, 2018.

[20] Y. Zhang, Z. Gan, K. Fan, Z. Chen, R. Henao, D. Shen, and L. Carin, "Adversarial feature matching for text generation," in *International Conference on Machine Learning*. PMLR, 2017, pp. 4006–4015.

[21] L. Chen, S. Dai, C. Tao, H. Zhang, Z. Gan, D. Shen, Y. Zhang, G. Wang, R. Zhang, and L. Carin, "Adversarial text generation via feature-mover's distance," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[22] W. Nie, N. Narodytska, and A. Patel, "Relgan: Relational generative adversarial networks for text generation," in *International conference on learning representations*, 2018.

[23] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," *Advances in neural information processing systems*, vol. 12, 1999.

[24] S. Bengio, O. Vinyals, N. Jaitly, and N. Shazeer, "Scheduled sampling for sequence prediction with recurrent neural networks," *Advances in neural information processing systems*, vol. 28, 2015.

[25] E. Jang, S. Gu, and B. Poole, "Categorical reparametrization with gumble-softmax," in *International Conference on Learning Representations (ICLR 2017)*. OpenReview. net, 2017.

[26] C. J. Maddison, A. Mnih, and Y. W. Teh, "The concrete distribution: A continuous relaxation of discrete random variables," *arXiv preprint arXiv:1611.00712*, 2016.

[27] A. Santoro, R. Faulkner, D. Raposo, J. Rae, M. Chrzanowski, T. Weber, D. Wierstra, O. Vinyals, R. Pascanu, and T. Lillicrap, "Relational recurrent neural networks," *Advances in neural information processing systems*, vol. 31, 2018.

[28] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[29] Y. Chen, "Convolutional neural network for sentence classification," Master's thesis, University of Waterloo, 2015.

[30] "Project gutenberg." [Online]. Available: https://www.gutenberg.org/

[31] "char-rnn-generation," 2018. [Online]. Available: https://github.com/spro/practical-pytorch/blob/master/char-rnn-generation/char-rnn-generation.ipynb

[32] W. Lam, "Textgan-pytorch," 2018. [Online]. Available: https://github.com/williamSYSU/TextGAN-PyTorch

Eduin Hernandez	Collection of training data Manual preprocessing of collected data Implementation and integration of benchmark, GAN, text classifier models Comparing different classifiers in terms of appropriate parameters Collaborative work during the documentation process of the project
Eda Ayan	Manual preprocessing of collected data Literature research of benchmark models for text generation Collaborative work during the documentation process of the project
Utku Erdem	Manual preprocessing of collected data Literature research based on GANs for text generation Reporting/investigating the novel SeqGAN and RelGAN models Training on some of the benchmark models and text classifier models Collaborative work during the documentation process of the project

Table V: Contribution of each member to the project

VI. Appendix

A. Contributions

Contribution of each member to the project is stated in Tab. V. As it can be seen from Tab. V, we divide the important tasks between group members every member take part in the project. Division of the tasks are done by regarding the skills and tendencies of each member.

B. Main Codes

```
#char-RNN Code Block

#train.py
import torch
import torch.nn as nn
from torch.autograd import Variable
import argparse
import os

from tqdm import tqdm

from helpers import *
from model import *
from generate import *

# Parse command line arguments
argparser = argparse.ArgumentParser()
argparser.add_argument('filename', type=str)
argparser.add_argument('--model', type=str, default="gru")
argparser.add_argument('--n_epochs', type=int, default=2000)
argparser.add_argument('--print_every', type=int, default=100)
argparser.add_argument('--hidden_size', type=int, default=100)
argparser.add_argument('--n_layers', type=int, default=2)
argparser.add_argument('--learning_rate', type=float, default=0.01)
argparser.add_argument('--chunk_len', type=int, default=200)
argparser.add_argument('--batch_size', type=int, default=100)
argparser.add_argument('--shuffle', action='store_true')
argparser.add_argument('--cuda', action='store_true')
args = argparser.parse_args()

if args.cuda:
    print("Using CUDA")

file, file_len = read_file(args.filename)

def random_training_set(chunk_len, batch_size):
    inp = torch.LongTensor(batch_size, chunk_len)
    target = torch.LongTensor(batch_size, chunk_len)
    for bi in range(batch_size):
        start_index = random.randint(0, file_len - chunk_len)
        end_index = start_index + chunk_len + 1
        chunk = file[start_index:end_index]
        inp[bi] = char_tensor(chunk[:-1])
        target[bi] = char_tensor(chunk[1:])
    inp = Variable(inp)
    target = Variable(target)
    if args.cuda:
        inp = inp.cuda()
        target = target.cuda()
    return inp, target

def train(inp, target):
    hidden = decoder.init_hidden(args.batch_size)
    if args.cuda:
        hidden = hidden.cuda()
    decoder.zero_grad()
    loss = 0

    for c in range(args.chunk_len):
        output, hidden = decoder(inp[:,c], hidden)
        loss += criterion(output.view(args.batch_size, -1), target[:,c])

    loss.backward()
    decoder_optimizer.step()

    return loss.data / args.chunk_len

def save():
    save_filename = os.path.splitext(os.path.basename(args.filename))[0] + '.pt'
    torch.save(decoder, save_filename)
    print('Saved as %s' % save_filename)

# Initialize models and start training

decoder = CharRNN(
    n_characters,
    args.hidden_size,
    n_characters,
    model=args.model,
    n_layers=args.n_layers,
)
decoder_optimizer = torch.optim.Adam(decoder.parameters(), lr=args.learning_rate)
criterion = nn.CrossEntropyLoss()

if args.cuda:
    decoder.cuda()

start = time.time()
all_losses = []
loss_avg = 0

try:
    print("Training for %d epochs..." % args.n_epochs)
    for epoch in tqdm(range(1, args.n_epochs + 1)):
        loss = train(*random_training_set(args.chunk_len, args.batch_size))
        loss_avg += loss

        if epoch % args.print_every == 0:
            print('[%s (%d %d%%) %.4f]' % (time_since(start), epoch, epoch / args.n_epochs * 100, loss))
            print(generate(decoder, 'Wh', 100, cuda=args.cuda), '\n')

    print("Saving...")
    save()

except KeyboardInterrupt:
    print("Saving before quit...")
    save()

#generate.py
import torch
import os
import argparse

from helpers import *
from model import *
```

```
def generate(decoder, prime_str='A', predict_len=100, temperature=0.8, cuda=False):
    hidden = decoder.init_hidden(1)
    prime_input = Variable(char_tensor(prime_str).unsqueeze(0))

    if cuda:
        hidden = hidden.cuda()
        prime_input = prime_input.cuda()
    predicted = prime_str

    # Use priming string to "build up" hidden state
    for p in range(len(prime_str) - 1):
        _, hidden = decoder(prime_input[:,p], hidden)

    inp = prime_input[:,-1]

    # for p in range(predict_len):
    p = 0
    predicted_char = ''
    while p < predict_len or predicted_char != '.':
        p += 1

        output, hidden = decoder(inp, hidden)

        # Sample from the network as a multinomial distribution
        output_dist = output.data.view(-1).div(temperature).exp()
        top_i = torch.multinomial(output_dist, 1)[0]

        # Add predicted character to string and use as next input
        predicted_char = all_characters[top_i]
        predicted += predicted_char
        inp = Variable(char_tensor(predicted_char).unsqueeze(0))
        if cuda:
            inp = inp.cuda()

    return predicted

# Run as standalone script
if __name__ == '__main__':

# Parse command line arguments
    argparser = argparse.ArgumentParser()
    argparser.add_argument('filename', type=str)
    argparser.add_argument('--output-filename', type=str)
    # argparser.add_argument('-p', '--prime_str', type=str, default='A')
    argparser.add_argument('-l', '--predict_len', type=int, default=400)
    argparser.add_argument('-t', '--temperature', type=float, default=0.8)
    argparser.add_argument('--cuda', action='store_true')
    args = argparser.parse_args()

    decoder = torch.load(args.filename)
    output_filename = args.output_filename
    del args.filename
    del args.output_filename

    for prime_str in ['A', 'Th', 'Wh', 'Fr', 'M', 'T', 'W', 'F', 'No', 'Yes', 'King']*5:
        args.prime_str = prime_str
        paragraph = generate(decoder, **vars(args)) + '\n\n'
        with open('./samples/' + output_filename, 'a+') as outfile:
            outfile.write(paragraph)

    print('Samples Generated!')

    # argparser = argparse.ArgumentParser()
    # argparser.add_argument('filename', type=str)
    # argparser.add_argument('-p', '--prime_str', type=str, default='A')
    # argparser.add_argument('-l', '--predict_len', type=int, default=100)
    # argparser.add_argument('-t', '--temperature', type=float, default=0.8)
    # argparser.add_argument('--cuda', action='store_true')
    # args = argparser.parse_args()

    # decoder = torch.load(args.filename)
    # del args.filename
    # print(generate(decoder, **vars(args)))
```

```
#SeqGAN Model Block Code
#run_seqgan.py
import nltk
nltk.download('punkt')

import sys
from subprocess import call
import argparse

import os

def parse_args():
    parser = argparse.ArgumentParser(description='Author Text Classifier')

    # parser.add_argument('--job-id', type=int, default=3)
    parser.add_argument('--dataset', type=str, default='shakespeare', help='Name of dataset to use. Must
    match with the file in ./dataset/ and ./dataset/testdata/.')
    parser.add_argument('--real-data', type=int, default=1, help='Set to 0 if synthetic data. Set to 1 for
    real data.')
    parser.add_argument('--vocab-size', type=int, default=0, help='Vocab size. Set 0 for real data.')
    parser.add_argument('--adv-train-epoch', type=int, default=20, help='Adversarial Train Epoch.')
    args = parser.parse_args()
    return args

seqgan_args = parse_args()

# Executables
executable = 'python'  # specify your own python interpreter path here
rootdir = '../'
scriptname = 'main.py'

# ===Program===
if_test = int(False)
run_model = 'seqgan'
CUDA = int(True)
oracle_pretrain = int(True)
gen_pretrain = int(False)
dis_pretrain = int(False)
MLE_train_epoch = 120
# ADV_train_epoch = 200
tips = 'SeqGAN experiments'

# ===Oracle  or Real===
# if_real_data = [int(False), int(True), int(True), int(True)]
# dataset = ['oracle', 'image_coco', 'emnlp_news', 'shakespeare']
# vocab_size = [5000, 0, 0, 0]

# ===Basic Param===
data_shuffle = int(False)
model_type = 'vanilla'
gen_init = 'normal'
dis_init = 'uniform'
samples_num = 10000
batch_size = 64
max_seq_len = 20
gen_lr = 0.01
dis_lr = 1e-4
pre_log_step = 10
adv_log_step = 1

# ===Generator===
ADV_g_step = 1
rollout_num = 16
gen_embed_dim = 32
gen_hidden_dim = 32

# ===Discriminator===
d_step = 5
d_epoch = 3
ADV_d_step = 4
ADV_d_epoch = 2
dis_embed_dim = 64
dis_hidden_dim = 64
```

```
# ===Metrics===
use_nll_oracle = int(True)
use_nll_gen = int(True)
use_nll_div = int(True)
use_bleu = int(True)
use_self_bleu = int(True)
use_ppl = int(False)

args = [
    # Program
    '--if_test', if_test,
    '--run_model', run_model,
    '--cuda', CUDA,
    # '--device', gpu_id,  # comment for auto GPU
    '--ora_pretrain', oracle_pretrain,
    '--gen_pretrain', gen_pretrain,
    '--dis_pretrain', dis_pretrain,
    '--mle_epoch', MLE_train_epoch,
    # '--adv_epoch', ADV_train_epoch,
    '--adv_epoch', seqgan_args.adv_train_epoch,
    '--tips', tips,

    # Oracle or Real
    # '--if_real_data', if_real_data[args2.job_id],
    # '--dataset', dataset[args2.job_id],
    # '--vocab_size', vocab_size[args2.job_id],

    '--if_real_data', seqgan_args.real_data,
    '--dataset', seqgan_args.dataset,
    '--vocab_size', seqgan_args.vocab_size,

    # Basic Param
    '--shuffle', data_shuffle,
    '--model_type', model_type,
    '--gen_init', gen_init,
    '--dis_init', dis_init,
    '--samples_num', samples_num,
    '--batch_size', batch_size,
    '--max_seq_len', max_seq_len,
    '--gen_lr', gen_lr,
    '--dis_lr', dis_lr,
    '--pre_log_step', pre_log_step,
    '--adv_log_step', adv_log_step,

    # Generator
    '--adv_g_step', ADV_g_step,
    '--rollout_num', rollout_num,
    '--gen_embed_dim', gen_embed_dim,
    '--gen_hidden_dim', gen_hidden_dim,

    # Discriminator
    '--d_step', d_step,
    '--d_epoch', d_epoch,
    '--adv_d_step', ADV_d_step,
    '--adv_d_epoch', ADV_d_epoch,
    '--dis_embed_dim', dis_embed_dim,
    '--dis_hidden_dim', dis_hidden_dim,

    # Metrics
    '--use_nll_oracle', use_nll_oracle,
    '--use_nll_gen', use_nll_gen,
    '--use_nll_div', use_nll_div,
    '--use_bleu', use_bleu,
    '--use_self_bleu', use_self_bleu,
    '--use_ppl', use_ppl,
]

args = list(map(str, args))
my_env = os.environ.copy()
call([executable, scriptname] + args, env=my_env, cwd=rootdir, shell=False)


#Text Classifiers Block Code
#main.py
```

```
import argparse
import os

from discriminators_all import discriminator_models as dm
from authors import get_train_path, get_test_path
from authors import authors as authors_dict
from tools import open_authors_list, author_text_prepocess, shuffle, read_file

import seaborn as sns

def str2bool(string):
    if isinstance(string, bool):
       return string

    if string.lower() in ('yes', 'true', 't', 'y', '1'):
        return True
    elif string.lower() in ('no', 'false', 'f', 'n', '0'):
        return False
    else:
        raise argparse.ArgumentTypeError('Boolean value expected.')

def parse_args():
    parser = argparse.ArgumentParser(description='Author Text Classifier')

    'Model Details'
    parser.add_argument('--model-name', type=str, default='svm', help='Model to use. Can be ' + str(list(dm
    .keys()))) #120
    parser.add_argument('--train', type=str2bool, default='False', help='Whether to retrain model if it
    exists')
    parser.add_argument('--save', type=str2bool, default='True', help='Whether to save model.')
    parser.add_argument('--char-paragraph', type=int, default=200, help='Number of characters required per
    paragraph.')

    parser.add_argument('--show-train-metrics', type=str2bool, default='False', help='Calculate and Show
    Train metrics')
    parser.add_argument('--show-val-metrics', type=str2bool, default='False', help='Calculate and Show
    Train metrics')
    parser.add_argument('--show-test-metrics', type=str2bool, default='True', help='Calculate and Show
    Train metrics')

    # parser.add_argument('--test-filepath', type=str, default="../Dataset/William Shakespeare/
    shakespeare_test_gan.txt", help='Test Text to Evaluate')
    # parser.add_argument('--test-author', type=str, default='shakespeare', help='Test Author being
    evaluated')
    parser.add_argument('--test-filepath', type=str, default="../textGAN/samples/shakespeare_test_seqgan.
    txt", help='Test Text to Evaluate')
    parser.add_argument('--test-author', type=str, default='shakespeare', help='Test Author being evaluated
    ')
    args = parser.parse_args()
    return args


if __name__ == '__main__':
    args = parse_args()

    assert args.model_name.lower() in list(dm.keys()), 'Error, ' + args.model_name + ' not in the list of
    models.'

    x_train_path, y_train = get_train_path()
    x_train = open_authors_list(x_train_path)
    x_train, y_train = author_text_prepocess(x_train, y_train, args.char_paragraph)
    x_train, y_train = shuffle(x_train, y_train)

    model = dm[args.model_name.lower()]()

    if args.train or not(model.save_exists()):
        model.fit(x_train, y_train)
    else:
        model.load_model()

    if (args.save and not(model.save_exists())) or (args.train and args.save):
        model.save_model()

    if args.show_train_metrics:
        predicted = model.predict(x_train)
        print('Train Metric Results:')
        print(model.classification_report(y_train, predicted))
        print('Accuracy: %1.2f%%\n' % (100*model.accuracy(y_train, predicted)))
        print('Confusion Matrix:\n', model.confusion_matrix(y_train, predicted, labels=[0,1,2,3,4,5]))

    if args.show_val_metrics:
        x_val_path, y_val = get_test_path()
        x_val = open_authors_list(x_val_path)
        x_val, y_val = author_text_prepocess(x_val, y_val, args.char_paragraph)
        x_val, y_val = shuffle(x_val, y_val)

        predicted = model.predict(x_val)
        print('Validation Metric Results:')
        print(model.classification_report(y_val, predicted))
        print('Accuracy: %1.2f%%\n' % (100*model.accuracy(y_val, predicted)))
        print('Confusion Matrix:\n', model.confusion_matrix(y_val, predicted, labels=[0,1,2,3,4,5]))
        print('\n')

        ax = sns.heatmap(model.confusion_matrix(y_val, predicted, labels=[0,1,2,3,4,5]),
                         annot=True, vmin=0, cbar=False, fmt=".4g",
                         xticklabels=list(authors_dict.keys()), yticklabels=list(authors_dict.keys()))
        ax.set(title='Val Data - Confusion Matrix with ' + args.model_name)


    if args.show_test_metrics and os.path.exists(args.test_filepath):
        x_test, _ = read_file(args.test_filepath)
        _,_, class_label = authors_dict[args.test_author.lower()]
        x_test, y_test = [x_test], [class_label]

        x_test, y_test = author_text_prepocess(x_test, y_test, args.char_paragraph)
        predicted = model.predict(x_test)

        print('Test Metrics Results:')
        print('Accuracy: %1.2f%%' % (100*model.accuracy(y_test, predicted)))
        print('f1-score: %1.2f%%' % (100*model.f1(y_test, predicted)))
        print('Confusion Matrix:\n', model.confusion_matrix(y_test, predicted, labels=[0,1,2,3,4,5]))

        # mat += model.confusion_matrix(y_test, predicted, labels=[0,1,2,3,4,5])
        # ax = sns.heatmap(mat,
        #                  annot=True, vmin=0, cbar=False, fmt=".4g",
        #                  xticklabels=list(authors_dict.keys()), yticklabels=list(authors_dict.keys()))
        # ax.set(title='seqGAN Test Data - Confusion Matrix with ' + args.model_name)

        # model.confusion2f1(mat)
        # import numpy as np
        # np.trace(mat)/mat.sum()
        # mat.sum()
        # np.nansum(model.confusion2f1(mat)[0] * (mat.sum(axis=1) / mat.sum()))

#bleu_score.py
import argparse

import nltk
import random
from nltk.translate.bleu_score import SmoothingFunction

from authors import authors as authors_dict


def get_tokenlized(file):
    """tokenlize the file"""
    tokenlized = list()
    with open(file, encoding='utf-8') as raw:
        for text in raw:
            text = nltk.word_tokenize(text.lower())
            if len(text) < 5:
                continue
            tokenlized.append(text)
    return tokenlized

class BLEU():
    def __init__(self, name=None, test_text=None, real_text=None, gram=3, portion=1):
        assert type(gram) == int or type(gram) == list, 'Gram format error!'
        self.test_text = test_text
        self.real_text = real_text
        self.gram = [gram] if type(gram) == int else gram
        self.sample_size = 200  # BLEU scores remain nearly unchanged for self.sample_size >= 200
        self.reference = None
        self.is_first = True
        self.portion = portion  # how many portions to use in the evaluation, default to use the whole test
      dataset

    def get_score(self, given_gram=None):
        """
        Get BLEU scores.
        :param is_fast: Fast mode
        :param given_gram: Calculate specific n-gram BLEU score
        """

        if self.is_first:
            self.get_reference()
            self.is_first = False
        return self.get_bleu(given_gram)

    def reset(self, test_text=None, real_text=None):
        self.test_text = test_text if test_text else self.test_text
        self.real_text = real_text if real_text else self.real_text

    def get_reference(self):
        reference = self.real_text.copy()

        # randomly choose a portion of test data
        # In-place shuffle
        random.shuffle(reference)
        len_ref = len(reference)
        reference = reference[:int(self.portion * len_ref)]
        self.reference = reference
        return reference

    def get_bleu(self, given_gram=None):
        if given_gram is not None:  # for single gram
            bleu = list()
            reference = self.get_reference()
            weight = tuple((1. / given_gram for _ in range(given_gram)))
            for idx, hypothesis in enumerate(self.test_text[:self.sample_size]):
                bleu.append(self.cal_bleu(reference, hypothesis, weight))
            return round(sum(bleu) / len(bleu), 3)
        else:  # for multiple gram
            all_bleu = []
            for ngram in self.gram:
                bleu = list()
                reference = self.get_reference()
                weight = tuple((1. / ngram for _ in range(ngram)))
                for idx, hypothesis in enumerate(self.test_text[:self.sample_size]):
                    bleu.append(self.cal_bleu(reference, hypothesis, weight))
                all_bleu.append(round(sum(bleu) / len(bleu), 3))
            return all_bleu

    @staticmethod
    def cal_bleu(reference, hypothesis, weight):
        return nltk.translate.bleu_score.sentence_bleu(reference, hypothesis, weight,
                                                       smoothing_function=SmoothingFunction().method1)

def str2bool(string):
    if isinstance(string, bool):
       return string

    if string.lower() in ('yes', 'true', 't', 'y', '1'):
        return True
    elif string.lower() in ('no', 'false', 'f', 'n', '0'):
        return False
    else:
        raise argparse.ArgumentTypeError('Boolean value expected.')

def parse_args():
    parser = argparse.ArgumentParser(description='Author Text Bleu Metric')
    parser.add_argument('--test-filepath', type=str, default="../textGAN/samples/dickens_test_seqgan.txt",
    help='Test Text to Evaluate')
    parser.add_argument('--test-author', type=str, default='dickens', help='Test Author being evaluated')

    parser.add_argument('--show-val-metrics', type=str2bool, default='False', help='Calculate and Show
    Train metrics')
    parser.add_argument('--show-test-metrics', type=str2bool, default='True', help='Calculate and Show
    Train metrics')

    args = parser.parse_args()
    return args

if __name__ == '__main__':
    args = parse_args()

    train_dataset, val_dataset, _ = authors_dict[args.test_author.lower()]

    train = get_tokenlized(train_dataset)

    bleu = BLEU('BLEU', gram=[2, 3, 4, 5])

    print('Author: ', args.test_author)
    if args.show_val_metrics:
        val = get_tokenlized(val_dataset)
        bleu.reset(test_text=train, real_text=val)
        print('Val Bleu - ', bleu.get_score())

    if args.show_test_metrics:
        test = get_tokenlized(args.test_filepath)
        bleu.reset(test_text=train, real_text=test)
        print('Test Bleu - ', bleu.get_score())
```