Using LSTM Discriminator in GAN Training Setting to Aid GPT-2 in Generating Drake Related Lyrics

Andres Caicedo (acd106)
Mac Gray (mjg85)
Dev Patel (djp69)
Duke University ECE 684 Fall 2023

Executive Summary

Objective:

The primary objective of this project is to develop a robust NLP system capable of authorship identification and the generation of fake Drake lyrics. We aim to leverage pre-trained models, fine-tuning, and GAN training to achieve high accuracy in differentiating between real and fake lyrics.

The project addresses two significant challenges in the NLP domain: authorship identification and fake text generation. In our project, authorship identification comprises differentiating between real and fake lyrics. Text generation aims to replicate the writing style of an author, hindering author identification.

Approach:

The project is divided into two main components: the Generative Left Side and the Discriminative Right Side. The Generative Left Side employs a pre-trained model: Generative PreTrained Transformer 2 (GPT-2), fine-tuned on Drake Lyrics. Generative Adversarial Network (GAN) training is employed, where the discriminator guides the generative model's training reference[1]. The Discriminative Right Side consists of a Long Short-Term Memory (LSTM) classifier continuously trained on both generated and real samples. Drake lyrics were obtained from Hugging Face[2].

Expected Outcomes:

- Accurate authorship identification for speeches.
- High-quality fake text generation that closely imitates author speech patterns.
- Insights into the challenges and nuances of authorship identification and fake text generation.

Challenges:

- Sequencing the feedback loop between models effectively.
- Fine tuning a model such that it closely imitates an author or artist.
- Addressing potential computational limitations that may impact convergence.

Conclusion:

This project aims to advance the field of NLP by tackling the complex tasks of authorship identification and fake text generation. By employing state-of-the-art techniques and fine-tuning on specific author data, we anticipate achieving significant progress in both areas. The project's outcomes have the potential to benefit various applications, including content generation, security, and content authentication. Further, we hope to investigate whether GAN training theory can be applied to the field of NLP. This result will be seen by examining whether the generator is more impressive after GAN training compared to the out of the box model. Lastly, as a result of this training we will also have a sophisticated discriminator capable of determining whether text is AI generated or not.

Table of Contents

Table of Figures

Introduction:

In an era characterized by the prolific generation and dissemination of textual content, the ability to identify authorship and discern authentic text from artificially generated material has emerged as a

paramount challenge in the field of Natural Language Processing (NLP). This project seeks to address these fundamental challenges through the development of a sophisticated NLP system.

The ubiquity of text-based communication on digital platforms, coupled with the emergence of powerful language models, has given rise to new opportunities and risks. The phenomenon of authorship identification involves determining whether a given text originated from a specific individual or source. It has applications in plagiarism detection and content attribution, making it a critical area of study for various fields, including law enforcement, journalism, and academia.

Conversely, the rise of generative models, such as GPT-2 and LLAMA, has raised concerns about the potential for generating highly convincing fake text. This poses a significant challenge for authorship detection systems, as the boundaries between genuine and artificially created content blur. It necessitates the development of robust tools capable of distinguishing between authentic authorship and machine-generated mimicry.

GPT-2 is a language model developed in 2019. It's renowned for its ability to generate human-like text in various styles and languages. GPT-2 has had a significant impact on the field of natural language processing and has found applications in content generation.

LSTM is a recurrent neural network architecture employed in sequential data tasks including NLP. It's designed to overcome the vanishing gradient problem, allowing it to capture and remember long-term dependencies in data.

GANs consist of two neural networks, a generator, and a discriminator, which are trained simultaneously through a competitive process. The generator aims to produce data that is indistinguishable from real data, while the discriminator tries to differentiate between real and generated text. During training, the generator improves its ability to generate realistic data by learning from the feedback provided by the discriminator, ultimately leading to the generation of high-quality text samples.

Project Objectives

The overarching objective of this project is to complete two tasks: identify whether a lyric sample was written by Drake and generate fake Drake lyrics. These objectives are intertwined, and we shall expand upon them herein.

We aim to create a discriminator capable of attributing lyrics to Drake. This system will rely on the distinctive linguistic patterns, writing style, and idiosyncrasies associated with Drake lyrics. We employed a generic LSTM model designed to analyze the sentiment of IMDB movie reviews[3].

```python
class LSTM(nn.Module):
    def __init__(
        self,
        vocab_size: int,
        embedding_dim: int,
        hidden_dim: int,
        output_dim: int,
        n_layers: int,
        dropout_rate: float,
        pad_index: int,
        **kwargs):
        """
        Create a LSTM model for classification.
        :param vocab_size: size of the vocabulary
        :param embedding_dim: dimension of embeddings
        :param hidden_dim: dimension of hidden features
        :param output_dim: dimension of the output layer which equals to the number of labels.
        :param n_layers: number of layers.
        :param dropout_rate: dropout rate.
        :param pad_index: index of the padding token.we
        """
```

```python
        super().__init__()
        # Add your code here. Initializing each layer by the given arguments.
        self.dropout = nn.Dropout(p=dropout_rate)
        self.embedding = nn.Embedding(num_embeddings=vocab_size, embedding_dim=embedding_dim, padding_idx=pad_index)
        self.lstm = nn.LSTM(input_size=embedding_dim, hidden_size=hidden_dim, num_layers=n_layers, batch_first=True)
        self.fc = nn.Linear(in_features=hidden_dim, out_features=output_dim)
        # self.fc2 = nn.Linear(in_features=hidden_dim*2, out_features=output_dim)
        # self.bidirectional=bidirectional
        # Weight initialization. DO NOT CHANGE!
        if "weight_init_fn" not in kwargs:
            self.apply(init_weights)
        else:
            self.apply(kwargs["weight_init_fn"])

    def forward(self, ids:torch.Tensor, length:torch.Tensor):
        """
        Feed the given token ids to the model.
        :param ids: [batch size, seq len] batch of token ids.
        :param length: [batch size] batch of length of the token ids.
        :return: prediction of size [batch size, output dim].
        """
        # Add your code here.
        embeddings = self.embedding(ids)
        pad_embed = nn.utils.rnn.pack_padded_sequence(embeddings, length, batch_first=True, enforce_sorted=False)
        _, (h_n, c_n) = self.lstm(pad_embed)
        # if self.bidirectional == True:
        #   out_dropout = self.dropout(torch.cat((h_n[-1,:,:], h_n[-2,:,:]), dim=1))
        # else:
        out_dropout = self.dropout(h_n[-1,:,:])
        # if self.bidirectional == True:
        #   out = self.fc2(out_dropout)
        # else:
        out = self.fc(out_dropout)
        prediction = out
        # pdb.set_trace()
        return prediction

# Initialize the discriminator
vocab_size = len(tokenizer)
embedding_dim = 256
hidden_dim = 100
max_seq_len = 128  # maximum sequence length

discriminator = LSTM(vocab_size, embedding_dim, hidden_dim,
                     2, 1, 0, tokenizer.pad_token_id)
```

Figure 1: LSTM Architecture Modified for Differentiation of Drake's lyrics

We simultaneously employed GPT-2 to generate lyrics purportedly from Drake in hopes that the generated text would match idiosyncrasies from the lyrics in the training set. This task presents a unique challenge as it requires the system to replicate an author's linguistic nuances and patterns accurately. Furthermore, we loaded a pre-trained GPT-2 model from HuggingFace and fine tuned it on a subset of the Drake lyrics dataset.

```python
from transformers import Trainer, TrainingArguments

# Load pre-trained GPT-2 model and tokenizer
model_name = "gpt2"
model = GPT2LMHeadModel.from_pretrained(model_name)
tokenizer = GPT2Tokenizer.from_pretrained(model_name)

# Before tokenization, set the tokenizer's pad token to the eos token
tokenizer.pad_token = tokenizer.eos_token

def tokenize_function(examples):
    # Now the tokenizer has a pad token and will pad the sequences up to the max_length
    return tokenizer(examples["text"], padding="max_length", truncation=True, max_length=128)

# Apply tokenization to all datasets
# tokenized_datasets = datasets.map(tokenize_function, batched=True)


tokenized_datasets = datasets.map(tokenize_function, batched=True)

print(tokenized_datasets)

# Data collator used for dynamically padding the batch size
data_collator = DataCollatorForLanguageModeling(
    tokenizer=tokenizer, mlm=False
)
```

```python
# Define training arguments
training_args = TrainingArguments(
    output_dir="./gpt2-drake-lyrics", # specify where to save the model
    overwrite_output_dir=True,
    num_train_epochs=3, # number of training epochs
    per_device_train_batch_size=4, # batch size for training
    per_device_eval_batch_size=4,  # batch size for evaluation
    eval_steps=400, # perform evaluation every X steps
    save_steps=800, # save model every X steps
    warmup_steps=500, # number of warmup steps
    prediction_loss_only=True,
)

# Initialize Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    data_collator=data_collator,
    train_dataset=tokenized_datasets["train"],
    eval_dataset=tokenized_datasets["validation"],
)

# Start training
trainer.train()
```

Figure 2: HuggingFace GPT-2 Fine-Tuned on Drake Lyrics

Significance of the Project

The outcomes of this project hold substantial implications for various domains. Accurate authorship identification has the potential to revolutionize fields such as plagiarism detection and content attribution. Additionally, generation of high-quality fake text generation has ramifications for content generation, security, and content authentication. Lastly, the attractiveness of mimicking your favorite artist employing a DIY model drew us to the topic.

In this digital age, where the authenticity and origin of textual content are of paramount importance, our project aims to contribute to the advancement of NLP techniques and provide valuable tools for both academia and industry.

Methodology

Methodology: Data Wrangling

In the initial phase of our experiment, conducted within the secure Duke Virtual Container environment, we prepared the dataset for subsequent model training and evaluation. This involved data cleaning, character length filtering, and feature engineering.

Some song lyrics were either empty or too short for meaningful analysis, prompting a data cleaning process. To determine which lyrics to remove, we analyzed the distribution of lyric character lengths using histograms and established a suitable cutoff point.

For the lower bound, approximately 350 characters were identified as the minimum for a single verse in a song. To refine this cutoff, we used an iterative masking technique.

Similarly, for the upper bound, we determined that around 7000 characters served as a reasonable upper limit through visual inspection, iterative histogram analysis, and iterative masking.

After these preprocessing steps, we obtained a refined dataset containing song lyrics with character lengths ranging from 350 to 7000 characters. This dataset formed the basis for our subsequent analysis and modeling.
To facilitate model training and evaluation, we split the dataset into appropriate subsets, including training, validation and test sets with a 90%,7%, 3% split. This division ensured that our models were trained on plenty of ‘real’ data while data while providing a reliable assessment of their performance.

Methodology: GAN training

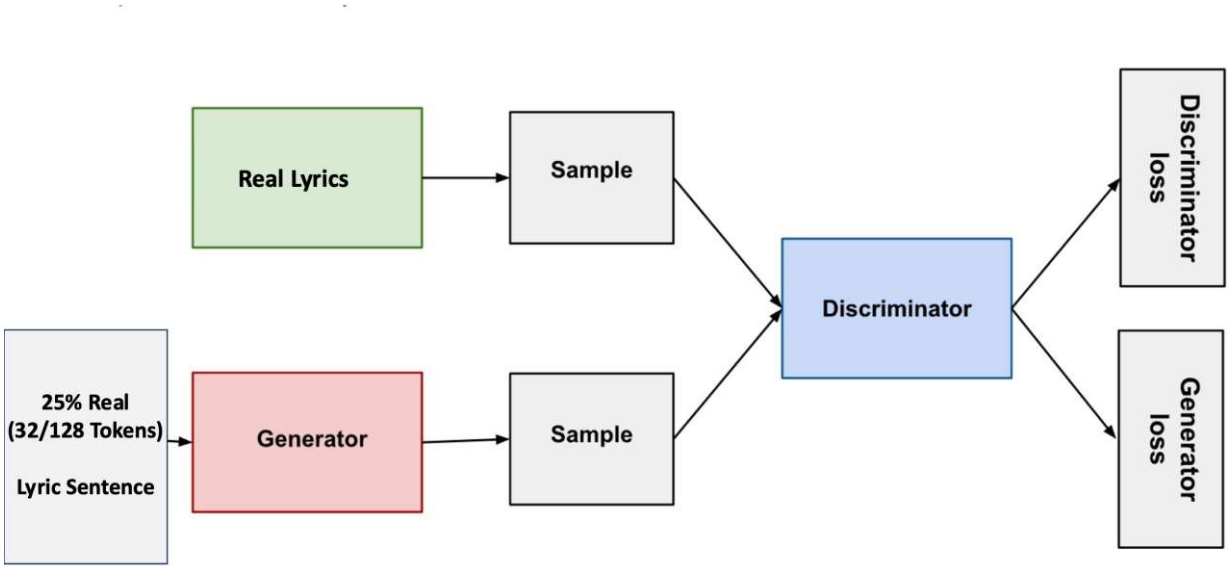


Figure 3: GAN Functional Block Description

Discriminative Right Side:
The Discriminative Right Side of our project serves as the evaluator to distinguish between authentic and fake Drake lyrics generated by the left side. It plays a pivotal role in authorship identification and assessing the quality of generated text. Our Discriminative Right Side consists of an LSTM classifier designed to continuously evolve and adapt as the left side becomes more proficient at generating lyrics. The LSTM classifier is continuously trained on both real lyrics and generated lyrics.

In ECE661, Fall 2023 cohort, each of the authors constructed a discriminator LSTM model in a homework assignment. As a result, we employed this valuable resource as the starting point for the Discriminator model parameters. We previously found these values useful for analyzing sentiment in movie reviews and surmised transferability and applicability to the focus of the project herein.

```
98 # Initialize the discriminator
99 vocab_size = len(tokenizer)
100 embedding_dim = 256
101 hidden_dim = 100
102 max_seq_len = 128  #Maximum sequence length
103
104 discriminator = LSTM(vocab_size, embedding_dim, hidden_dim,
105                      2, 1, 0, tokenizer.pad_token_id)
106
```

Figure 4: LSTM Model Parameters

Generative Left Side:

The Left Side Generator is dedicated to the technical aspect of text generation, with a particular focus on emulating the lyrical style of the artist Drake. Our approach is built upon the utilization of the GPT-2 language model exclusively provided by Hugging Face. The fundamental technique employed revolves around fine-tuning this model using our refined dataset containing lyrics authored by Drake. We Fine-tuned by updating the model weights to align with the intricacies and themes characteristic of Drake's lyrics.

Integration of Both Models: The GAN loop

To refine the generated text and enhance its authenticity, we employ GAN training. A discriminator model, residing in the Discriminative Right Side, evaluates the text produced by the generative model. If the discriminator classifies the text as "fake," the generative model is penalized, and the weights are adjusted. Conversely, if the discriminator recognizes the text as "real," the generative model's weights are updated less or not updated.

1. Initially, we pass the real lyric sample through the discriminator and obtain the prediction: real or fake.
2. As a result of the prediction, we obtain the cross-entropy loss for the real data, which is the loss between the prediction and the real labels(1s).
3. Subsequently, we generate lyrics from the generate_text_gpt2 function shown in figure 4 below.
4. Afterward, we passed the fake lyrics through the discriminator and obtain a prediction. We also get the loss for the fake data in the discriminator, which is the cross-entropy loss between the prediction and the fake labels(0s).
5. After the discriminator has seen both real and fake data, we update the discriminator parameters.
6. After the weights are updated, we pass the fake lyrics again through the discriminator and employ the prediction to obtain the loss for the generator, which is the cross-entropy loss between the discriminator prediction and the real labels (1s).
7. Lastly, we update the generator parameters.

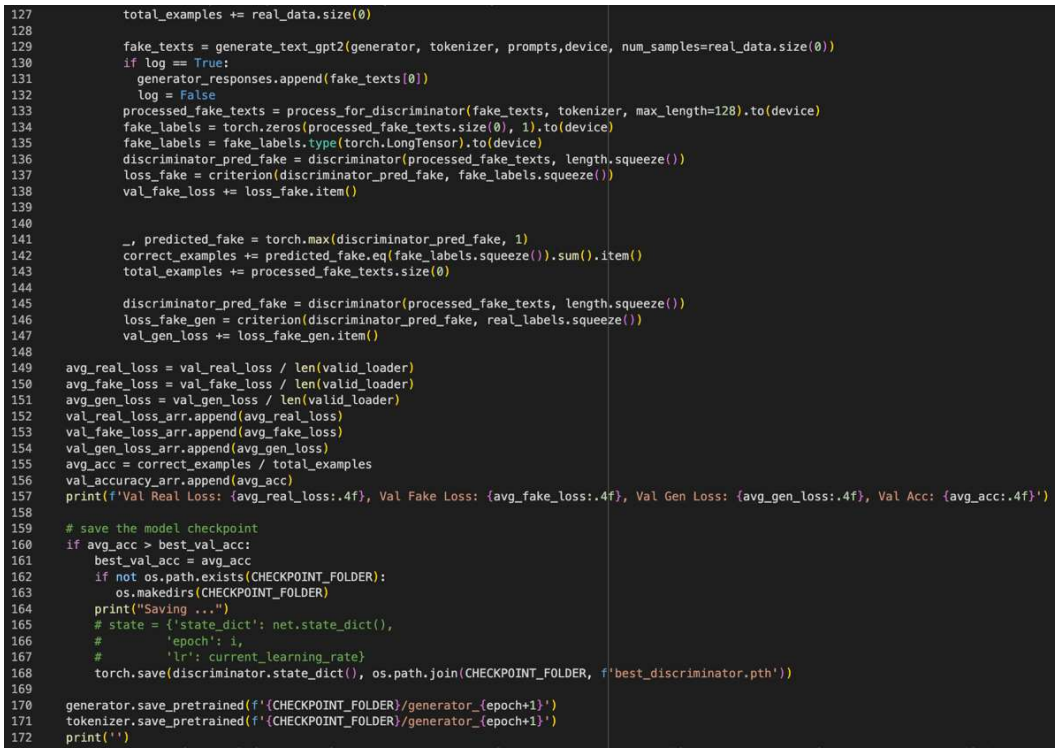
   This dynamic feedback loop aims to iteratively improve the quality of the generated text. In the figure below, we demonstrate the generative function within the loop.

```python
def generate_text_gpt2(model, tokenizer,prompts,device, num_samples):
    # pdb.set_trace()
    generated_texts = []
    model.to(device)
    for i in range(num_samples):
        # Randomly choose a starting prompt from the provided list
        # random_prompt = random.choice(prompts)
        random_prompt = prompts[i]
        # Encode the chosen prompt
        encoded_prompt = tokenizer.encode(random_prompt, return_tensors="pt").to(device)[:,:32]

        # Generate attention mask for the encoded prompt
        attention_mask = encoded_prompt.ne(tokenizer.pad_token_id).int().to(device)

        outputs = model.generate(
            input_ids=encoded_prompt,
            attention_mask=attention_mask,
            max_length=128,  # Adjust the length as needed
            num_return_sequences=1,
            no_repeat_ngram_size=2,
            top_k=50,
            top_p=0.95,
            temperature=1.0,
            do_sample=True
        )

        generated_text = tokenizer.decode(outputs[0], skip_special_tokens=True)
        generated_texts.append(generated_text)
    return generated_texts
```

Figure 5: Generating Fake Lyrics Employing GPT-2

```python
from __future__ import generator_stop

import torch
import pdb
import logging
logging.disable(logging.WARNING)
from tqdm import tqdm
from torch.utils.data import DataLoader

train_accuracy_arr = []
train_fake_loss_arr = []
train_real_loss_arr = []
train_gen_loss_arr = []
val_accuracy_arr = []
val_fake_loss_arr = []
val_real_loss_arr = []
val_gen_loss_arr = []

CHECKPOINT_FOLDER = "/content/drive/MyDrive/"

generator_responses = []

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
generator.to(device)
discriminator.to(device)
criterion = nn.CrossEntropyLoss().to(device)
# Optimizers
optimizer_g = torch.optim.Adam(generator.parameters(), lr=0.001)
optimizer_d = torch.optim.Adam(discriminator.parameters(), lr=0.001)

num_epochs = 20
tokenizer.pad_token = tokenizer.eos_token
best_val_acc = 0
for epoch in range(num_epochs):
    discriminator.train()
    generator.train()
    #pdb.set_trace()
    total_examples = 0
    correct_examples = 0
    train_real_loss = 0
    train_fake_loss = 0
    train_gen_loss = 0
    log = True
    for real_data_batch in tqdm(train_loader, desc='training'):
        discriminator.zero_grad()
        optimizer_d.zero_grad()
        real_data, _ = real_data_batch
        real_data = real_data.to(device)
        prompts = []
        for i in range(len(real_data)):
          prompts.append(tokenizer.decode(real_data[i]))
        real_labels = torch.ones(real_data.size(0), 1).to(device)
        real_labels = real_labels.type(torch.LongTensor).to(device)
        length = [128]*real_data.size(0)
        length = torch.from_numpy(np.array(length))
        length = length.view(real_data.size(0),-1)
        discriminator_pred_real = discriminator(real_data, length.squeeze())

        loss_real = criterion(discriminator_pred_real, real_labels.squeeze())
        loss_real.backward()
        train_real_loss += loss_real.item()

        _, predicted_real = torch.max(discriminator_pred_real, 1)
```

```
        correct_examples += predicted_real.eq(real_labels.squeeze()).sum().item()
        total_examples += real_data.size(0)

        fake_texts = generate_text_gpt2(generator, tokenizer, prompts,device, num_samples=real_data.size(0))
        processed_fake_texts = process_for_discriminator(fake_texts, tokenizer, max_length=128).to(device)
        fake_labels = torch.zeros(processed_fake_texts.size(0), 1).to(device)
        fake_labels = fake_labels.type(torch.LongTensor).to(device)
        discriminator_pred_fake = discriminator(processed_fake_texts, length.squeeze())
        loss_fake = criterion(discriminator_pred_fake, fake_labels.squeeze())
        train_fake_loss += loss_fake.item()
        loss_fake.backward()


        optimizer_d.step()
        _, predicted_fake = torch.max(discriminator_pred_fake, 1)
        correct_examples += predicted_fake.eq(fake_labels.squeeze()).sum().item()
        total_examples += processed_fake_texts.size(0)

        generator.zero_grad()
        optimizer_g.zero_grad()
        discriminator_pred_fake = discriminator(processed_fake_texts, length.squeeze())
        loss_fake_gen = criterion(discriminator_pred_fake, real_labels.squeeze())
        train_gen_loss += loss_fake_gen.item()
        loss_fake_gen.backward()
        optimizer_g.step()

    avg_real_loss = train_real_loss / len(train_loader)
    avg_fake_loss = train_fake_loss / len(train_loader)
    avg_gen_loss = train_gen_loss / len(train_loader)
    train_real_loss_arr.append(avg_real_loss)
    train_fake_loss_arr.append(avg_fake_loss)
    train_gen_loss_arr.append(avg_gen_loss)
    avg_acc = correct_examples / total_examples
    train_accuracy_arr.append(avg_acc)
    print(f'Train Real Loss: {avg_real_loss:.4f}, Train Fake Loss: {avg_fake_loss:.4f}, Train Gen Loss: {avg_gen_loss:.4f}, Train Acc: {avg_acc:.4f}')

    discriminator.eval()
    generator.eval()
    total_examples = 0
    correct_examples = 0
    val_real_loss = 0
    val_fake_loss = 0
    val_gen_loss = 0
    with torch.no_grad():
        for real_data_batch in tqdm(valid_loader, desc='validating'):

            real_data, _ = real_data_batch
            real_data = real_data.to(device)
            prompts = []
            for i in range(len(real_data)):
              prompts.append(tokenizer.decode(real_data[i]))
            real_labels = torch.ones(real_data.size(0), 1).to(device)
            real_labels = real_labels.type(torch.LongTensor).to(device)
            length = [128]*real_data.size(0)
            length = torch.from_numpy(np.array(length))
            length = length.view(real_data.size(0),-1)
            discriminator_pred_real = discriminator(real_data, length.squeeze())

            loss_real = criterion(discriminator_pred_real, real_labels.squeeze())
            val_real_loss += loss_real.item()

            _, predicted_real = torch.max(discriminator_pred_real, 1)
            correct_examples += predicted_real.eq(real_labels.squeeze()).sum().item()
```



```
            total_examples += real_data.size(0)

            fake_texts = generate_text_gpt2(generator, tokenizer, prompts,device, num_samples=real_data.size(0))
            if log == True:
              generator_responses.append(fake_texts[0])
              log = False
            processed_fake_texts = process_for_discriminator(fake_texts, tokenizer, max_length=128).to(device)
            fake_labels = torch.zeros(processed_fake_texts.size(0), 1).to(device)
            fake_labels = fake_labels.type(torch.LongTensor).to(device)
            discriminator_pred_fake = discriminator(processed_fake_texts, length.squeeze())
            loss_fake = criterion(discriminator_pred_fake, fake_labels.squeeze())
            val_fake_loss += loss_fake.item()


            _, predicted_fake = torch.max(discriminator_pred_fake, 1)
            correct_examples += predicted_fake.eq(fake_labels.squeeze()).sum().item()
            total_examples += processed_fake_texts.size(0)

            discriminator_pred_fake = discriminator(processed_fake_texts, length.squeeze())
            loss_fake_gen = criterion(discriminator_pred_fake, real_labels.squeeze())
            val_gen_loss += loss_fake_gen.item()

    avg_real_loss = val_real_loss / len(valid_loader)
    avg_fake_loss = val_fake_loss / len(valid_loader)
    avg_gen_loss = val_gen_loss / len(valid_loader)
    val_real_loss_arr.append(avg_real_loss)
    val_fake_loss_arr.append(avg_fake_loss)
    val_gen_loss_arr.append(avg_gen_loss)
    avg_acc = correct_examples / total_examples
    val_accuracy_arr.append(avg_acc)
    print(f'Val Real Loss: {avg_real_loss:.4f}, Val Fake Loss: {avg_fake_loss:.4f}, Val Gen Loss: {avg_gen_loss:.4f}, Val Acc: {avg_acc:.4f}')

    # save the model checkpoint
    if avg_acc > best_val_acc:
        best_val_acc = avg_acc
        if not os.path.exists(CHECKPOINT_FOLDER):
           os.makedirs(CHECKPOINT_FOLDER)
        print("Saving ...")
        # state = {'state_dict': net.state_dict(),
        #          'epoch': i,
        #          'lr': current_learning_rate}
        torch.save(discriminator.state_dict(), os.path.join(CHECKPOINT_FOLDER, f'best_discriminator.pth'))

    generator.save_pretrained(f'{CHECKPOINT_FOLDER}/generator_{epoch+1}')
    tokenizer.save_pretrained(f'{CHECKPOINT_FOLDER}/generator_{epoch+1}')
    print('')
```

Figure 6: GAN Training Loop

Methodology: Evaluation Metrics

We employed two methods to identify the progress of our project. First, we tracked accuracy over the first few epochs while we refined our training loop. Additionally, we performed visual inspection of the generated lyrics as is common in GAN training[1].

Methodology: Experimental Setup

As we progressed to the training phase of our generative model, we encountered computational resource limitations within the Duke Virtual Container. The complexity of fine-tuning pre-trained models and the iterative nature of GAN training required substantial computational power. To overcome these limitations and ensure the efficiency of our experiments, we transitioned our computational efforts to Google Colab, a cloud-based platform that offers scalable GPU resources. Within the Google Colab environment, we leveraged the following hardware and software resources for our experiments:

- GPU: We utilized NVIDIA Tesla GPUs to accelerate model training and inference.
- Libraries and Frameworks: We made extensive use of popular NLP libraries and frameworks, including TensorFlow, PyTorch, and Hugging Face Transformers, to implement and fine-tune our models.

- Python: Our experimentation was conducted using the latest Python 3. Version available on Colab in December 2023, with key data analysis and manipulation libraries such as NumPy, PyTorch, and Pandas.

Results

Table 1: Generated Lyrics Journey

Ground Truth	'But now were just close and distant\n Because I stepped out and went to your old assistant\n Im sorry but she needed assistance\n You know how I am Im so consistent\n You know how Im so persistent\n You should hear me out but that would require you to shut up and listen\n New guy but you know something is missing\n And the other day at your crib either we did or we didnt\n You think you broke your commitment\n Is he your man though? I mean either he is or he isnt\n Or maybe Im tripping\n You know that relationship is in shambles and\n You left and had to'
Out of Box GPT2	'But now were just close and distant\n Because I stepped out and went to your old assistant\n Im sorry but she needed assistance\n You know how I am Im getting so far without being able to find something\n My old assistants you should go after their father...I want it my way ㅠ\n Just after my old wife came around the corner\n So I walked away from this old woman to my room\n And just before she left we entered\xa0and got into my house.I was looking for something ㅋ \n I asked what I should do now that I noticed her and she said, "Oh please, please don\'t do'
Fine Tuned GPT2	'But now were just close and distant\n Because I stepped out and went to your old assistant\n Im sorry but she needed assistance\n You know how I am Im sorry for them people that dont love me\n Yeah now Im in tears, theres a lot to deal with\n I got a hard time finding your love, I just needed somebody to take care of\n When I had to go to Im confused, you didnt know who I would be\n Just when Im on my turn, Im stuck on the wrong side of a switch\n Cause there were a bunch of people all in different colors, only two in the picture\n But ahh'
Epoch0	'But now were just close and distant\n Because I stepped out and went to your old assistant\n Im sorry but she needed assistance\n You know how I am Ima need your help\n She said her boss wouldnt be happy about it\n Cause she still wants me, I know we must get married\n Baby I need you\n I want you, baby\n And for what?\n For what type of man? I dont know\n Like\n What you say to someone\n Tell them that I got a baby to make a living for\n Its like we were married for years now\n But yeah\n We are married now, but baby you\n'
Epoch9	'But now were just close and distant\n Because I stepped out and went to your old assistant\n Im sorry but she needed assistance\n You know how I am Im glad that Im here\n I dont have a lot to say but youre a friend, just lookin for somethin\n Something that I should mention\n Cause Im never lonely\n So just keep it quiet\n All you need is time and patience\n They dont know who I really am\n

	Forget that my reputation has been exposed\n Tell me your story\n Where you at? Where you right now?\n Or how you found somebody\n The thing that you are missing is motivation\n'
Epoch19	'But now were just close and distant\n Because I stepped out and went to your old assistant\n Im sorry but she needed assistance\n You know how I am Im sorry\n But I hope your girl got this right\n So can you give us some inspiration\n And tell us how you feel\n Tell me how we should explain it\n If you want me to tell you, you dont need to keep silent\n We should never be arguing about feelings\n I know you got some thoughts, but you cant take them away\n Just keep going forward and be as clear as possible\n Like we are the same\n When I hit the road\n Everybody knows\n'

Bolded \n is manually inserted new line

Table 2: Generated Lyrics from Generator at Epoch 19 on Various Prompts

Original	Generated
'I could tell you from experience\n These bitches aint loyal\n Shit aint been the same since\n They treat me like Im Royal\n Doing every female dirty\n Cause I havent found the real one\n She just wanna hang cause she never real fun\n Like pimp c said this aint 91\n Stopped working for a boss when I became one\n You would think they feline these niggas acting pussy\n You would think Im the edge they way they push me\n I gotta keep real, cause fakeness aint me\n You always keep it real, that what you promised me\n If I'	'I could tell you from experience\n These bitches aint loyal\n Shit aint been the same since\n They treat me like Im Royal\n Doing it for the love, not the money\n Only they dont know about you, Im the man\n To take your time and get your act together\n Ive had more and more bad things done to me\n With all the niggas that took it with me, I get it\n So all of us can take it, its just me and all our friends\n Just me & all my nght\n Niggaz to have me at the game\n And for you'
'You used to call me on my cell phone\n Late night when you need my love, call me on my cell phone\n Late night when you need my love\n And I know that hotline bling\n Oh, and that could only mean one thing\n Well, and I know that hotline bling\n Oh, and that could only mean one thing\n Boy, what do you mean?\n When you nod your head yes, but you wanna say no\n What do you mean?\n When you dont want me to leave, then you tell me to go\n When you wanna turn right, when you wanna turn right\n Cause we argue'	'You used to call me on my cell phone\n Late night when you need my love, call me on my cell phone\n Late night when you need my love\n Call me back on a non-negotiate basis\n And when I get back at you\n Just tell me I want your love forever\n I dont mean for nothing, just tell you that when\n This is only a dream\n Youre never real about me\n Like I know it alright, yeah, I got some love to give\n They say it was for money, but I guess thats just what it meant\n So tell us you love is your heart beat\n'
'Wouldve came back for you\n I just needed time, to do what I had to do\n Caught in the life, I cant let it go\n Whether thats right I will never know\n Hope you will forgive me, never meant wrong\n Tried to be patient, waited too long\n But I wouldve came back, but I wouldve came back for you\n Wouldve came back, wouldve came back\n Lil Mel :\n Never on my mind to change\n I just stay the same\nSee my life image is plain\n Now addin a frame\n	'Wouldve came back for you\n I just needed time, to do what I had to do\n Caught in the life, I cant let it go\n You can never forget me\n Now I just cannt believe, youre still there\n Baby, give me the time\n For the ride, baby, oh my, my\n Oh my baby\n Im scared when I see your face, Im scared\n Is this me or is this one you just keep feeling for?\n Whos the difference between me and you, hes a little off\n Theres no way to get close\n

Do what I want for me\n Overlooking the rest'	Shes so different, can you tell me'

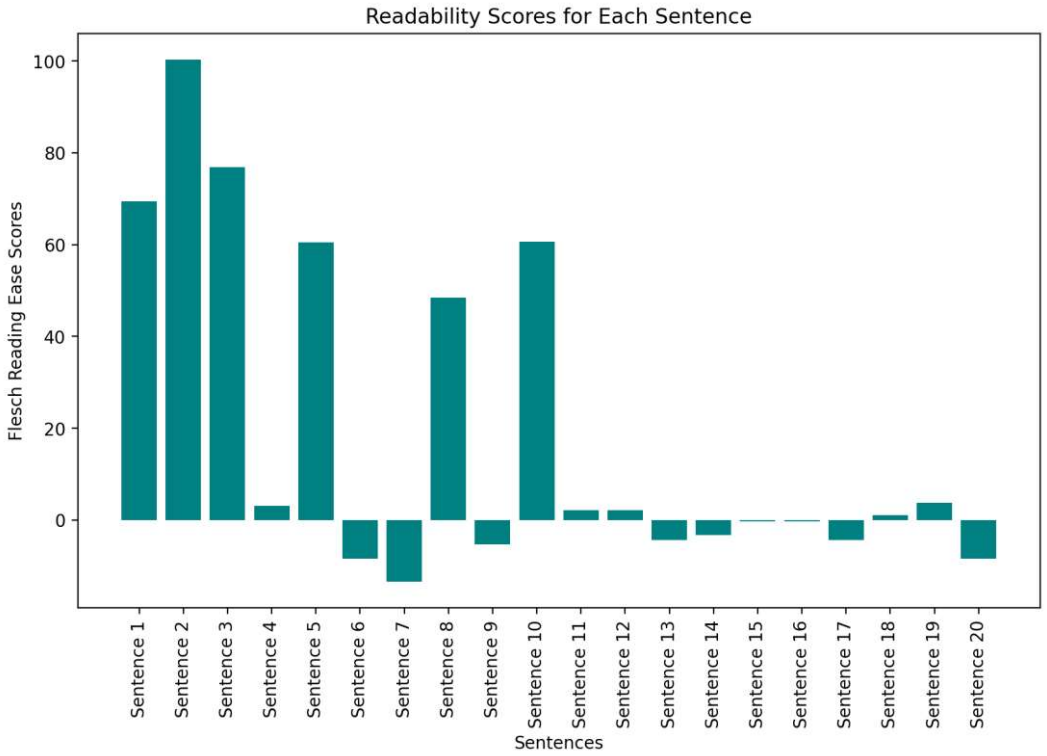


Figure 7: Readability of Generated Sentences at Each Epoch

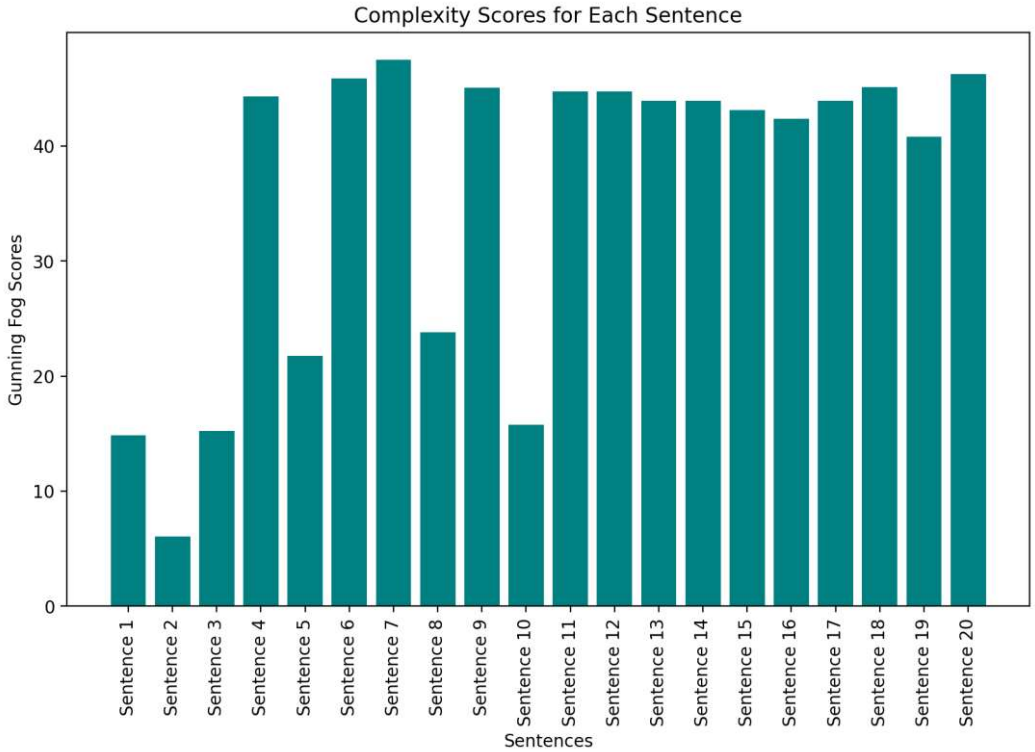


Figure 8: Complexity of Generated Sentence at Each Epoch

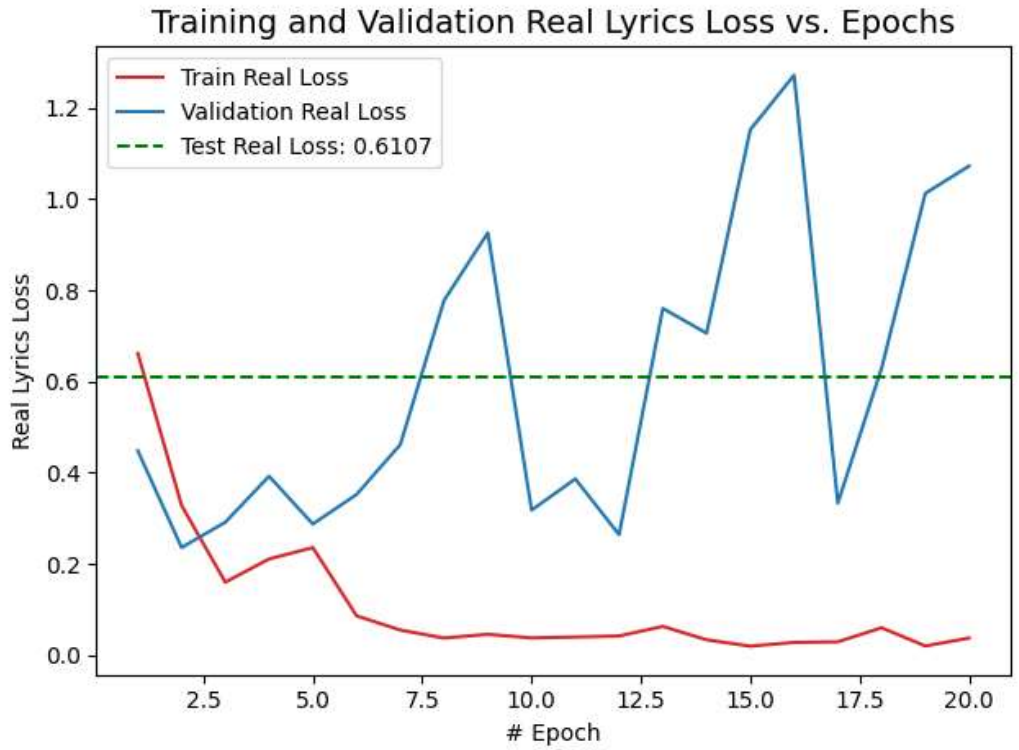


Figure 9: Training and Validation Real Lyrics Loss

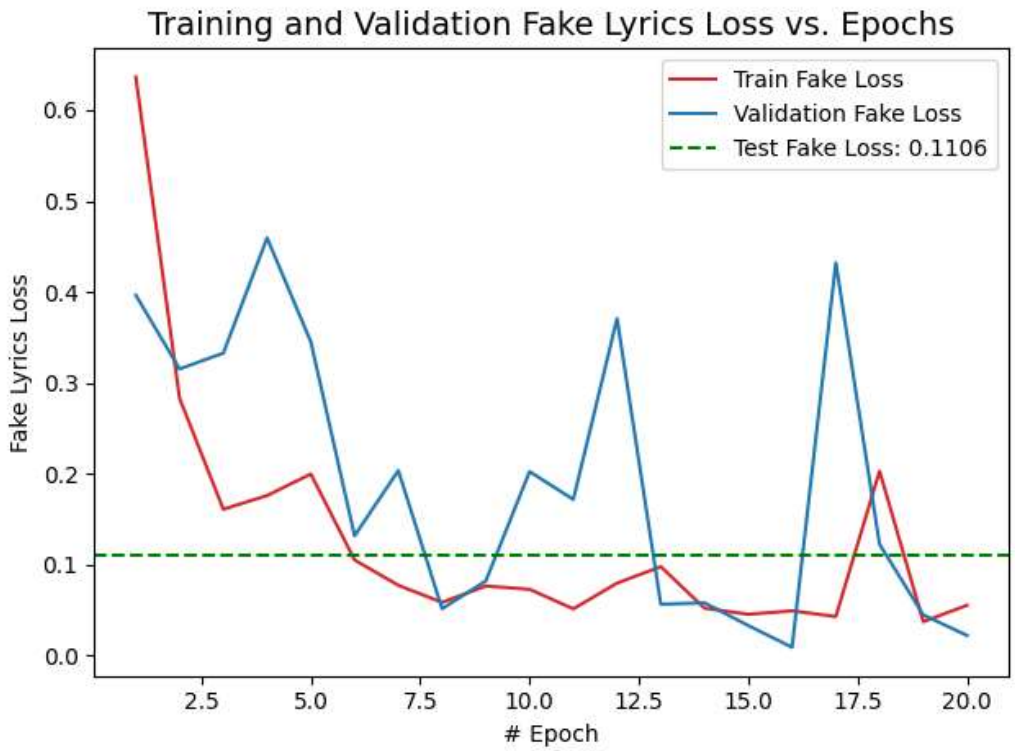


Figure 10: Training and Validation Fake Lyrics Loss

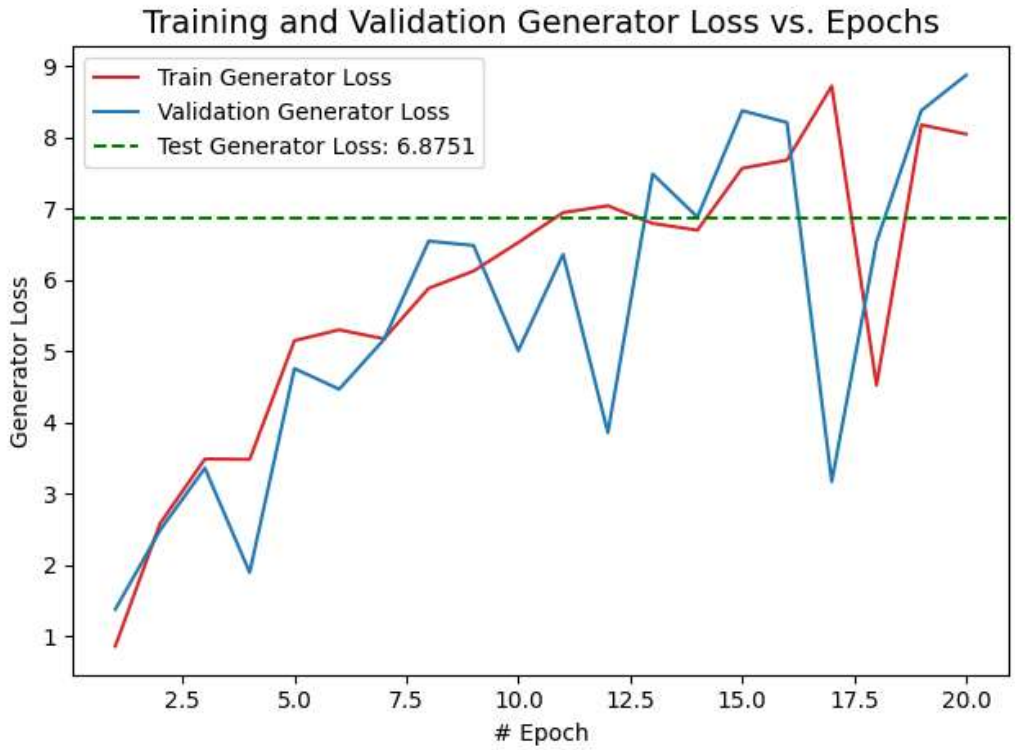


Figure 11: Training and Validation Generator Loss

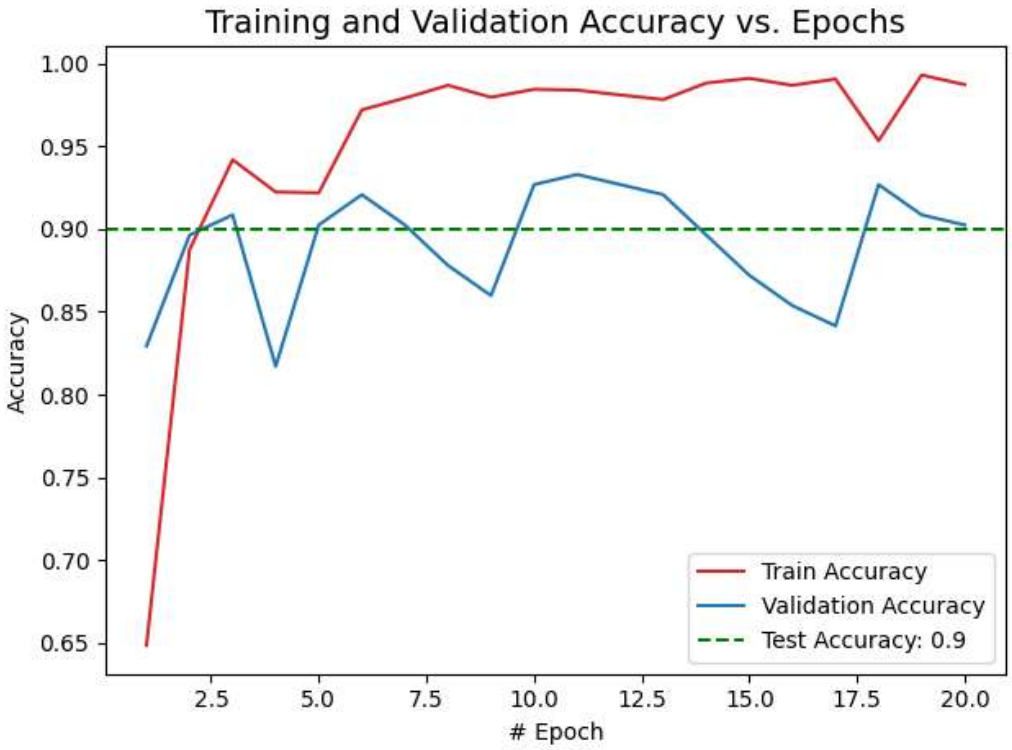


Figure 12: Training and Validation Accuracy

Discussion:

Looking at Table 1 above, we see the journey of how the generated lyrics improve overtime. Starting with generating lyrics from a gpt2 model out of the box with no further training, the lyrics that were generated do not have a flow and are rather more similar to speech than that of rap (Drake's style). From there we transition to using a GPT2 model that was finetuned on a portion of Drake lyrics. These lyrics are much closer to the style of rap, but the lyrics still do not flow well. After that, the next three rows represent the product of utilizing additional training. We utilized the GAN training and in a qualitative manner the generated lyrics flow and are closer to the linguistic/stylistic ton of Drake.

In Table 2, we take what we consider the best generator to be which was at Epoch 19 and show the original and then generated lyrics for some variety of prompts. Table 2 not only shows the ability of the generator and what it has learned, but also how important prompts can be. Exploring the prompts that are fed into the model is an area that has potential to be explored and tailored to be able to generate better lyrics.

Aside from qualitative assessment, we also utilized the Flesch Reading Ease Score and the Gunning Fog Index. The Flesch Reading Ease Score measures the readability of text, whereas the Gunning Fog Index measures the complexity of writing in English. Looking at Figures 7 and 8 we see that when the Flesch Reading Ease Scores are high the Gunning Fog Index Scores are low, and vice versa. Rap lyrics often contain cultural references, slang, and idiomatic expressions,

which may not be accounted for in the Gunning Fog Index but can affect the ease of understanding. The sentences that are generated after sentence 10, have high Gunning Fog Index, which shows that our generator is able to generate lyrics that are complex, mimicking rap lyrics that contain a mix of short, punchy phrases including slang, jargon and polysyllabic words.

Based on Figures 9 and 10 we see what we expected which is the Discriminator improving through the epochs thus becoming better at differentiating between real and fake lyrics. Figure 11 shows how the generator loss which is based on the prediction from the discriminator that process the generated text with the real labels (1s) increases throughout the epochs. This agrees with what we expected based on GAN theory. The discriminator is improving over each epoch thus it becomes hard to fool. Lastly, from Figure 12, we see that training accuracy increases over the epochs and validation accuracy varies a bit. It is possible that we need to train for more epochs to see a smoother convergence for the validation accuracy. On the held-out test set, our Discriminator is able to achieve an accuracy of 90%.

Addressing the challenge of fine-tuning in the context of natural language processing remains complex, distinct from its counterparts in computer vision. While the update mechanisms may not translate seamlessly, our efforts did yield noticeable improvements in the generative model's performance.

However, it's important to acknowledge that our project faced constraints, particularly in terms of computational resources. For instance, on Google Colab, we could only execute 20 epochs of training, a process that demanded approximately 10 hours of computation time on a V100 GPU. This resource limitation hindered our ability to match the quality of more extensive models such as Chat GPT.

Nonetheless, it is our belief that Generative Adversarial Networks (GANs) have a promising role to play in the field of natural language processing, provided that they are fine-tuned with care and precision. In addition to meticulous parameter tuning and adjustments, access to ample computational resources will be pivotal in fully harnessing the potential of GANs in NLP.

Interpretability:

While the GAN training loop is straightforward to understand, whereas the Discriminator gets better, the Generator is forced to get better. However, the inner workings of the neural networks both GPT2 and the LSTM model are not exactly interpretable.

Conclusion:

In concluding our project, we have achieved valuable insights into the challenges of authorship identification and fake text generation in the NLP domain. However, there remain avenues for improvement and exploration in our research and we suggest some of the following for future work, maybe even in ECE684 Spring 2024 Cohort.

Hyperparameter Tuning and Model Architectures:

Allocate dedicated time to extensive hyperparameter tuning. Consider experimenting with various model architectures, including the incorporation of Dropout layers, additional LSTM layers, and exploring different hidden dimension and embedding dimension sizes. Fine-tuning these aspects can lead to enhanced model performance and more accurate authorship identification.

Expanding the Dataset:

To alleviate the data wrangling process, seek and incorporate clean datasets from additional artists. A more diverse and extensive dataset can not only reduce the burden on researchers but also improve the generalizability of the models and expand the range of authorship identification.

Genre-Specific Analysis:

Explore the influence of musical genre on model performance. Certain genres may exhibit distinct linguistic patterns and lyrical characteristics, making them more or less susceptible to imitation. Investigating the impact of genre-specific rules and styles on model performance can provide deeper insights into the nuances of authorship identification and fake text generation.

These suggestions for future work aim to address existing limitations and further advance the capabilities of our NLP project. By fine-tuning model architectures, expanding the dataset, and delving into genre-specific analysis, we can continue to refine our models and contribute to the ongoing progress in the field of authorship identification and text generation.

References:

1. Gan tr Google Developers. "GAN Training." [Online]. Available: https://developers.google.com/machine-learning/gan/training

2. Hugging Face. "Drake Lyrics Dataset." [Online]. Available: https://huggingface.co/huggingartists/drake

3. Mac Gray, "ECE 661 LSTM Model - Homework Assignment, Fall 2023."

4. PyTorch. "DCGAN Tutorial for Generating Faces." [Online]. Available: https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html

5. Hugging Face. "GPT-2 Language Model." [Online]. Available: https://huggingface.co/gpt2