Generating Shakespearean Text using Character-level RNN

Abstract

This project implements a character-level recurrent neural network (Char-RNN) using Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) models to generate text in the style of William Shakespeare. By training on the complete works of Shakespeare, the models learn to produce text that mimics Shakespeare's language and style. The training and evaluation are conducted using PyTorch, exploring different optimizers, including Adam, RMSprop, and SGD, with varying hyperparameters. The experiments demonstrate the models' capabilities to generate coherent and stylistically consistent text. Additionally, this project highlights the impact of different optimizers, learning rates, momentum, and batch sizes on the performance of LSTM and GRU models, providing insights into the nuances of training deep learning models on sequential data.

Introduction

Background

Text generation is a significant area of research in natural language processing, with applications ranging from chatbots and conversational agents to creative writing and automated content creation. RNNs have proven effective in learning and generating sequential data due to their ability to capture dependencies across long sequences.

Motivation

Inspired by Andrej Karpathy's blog post "The Unreasonable Effectiveness of Recurrent Neural Networks," this project aims to replicate and extend the results by training on the complete works of Shakespeare. William Shakespeare, one of the most influential writers in the English language, provides an excellent source of complex and rich textual data.

Objectives

The primary goal is to create models that can generate coherent text which retains the stylistic elements of Shakespeare's writing, including his unique vocabulary, sentence structure, and poetic devices. This project explores the use of LSTM and GRU-based Char-RNNs for this purpose, comparing their performance with different optimizers.

Method

Data

- **Content:** Complete works of William Shakespeare.
- **Size:** Approximately 5.33 MB of text.
- **Characters:** The text includes a variety of characters such as letters, numbers, punctuation marks, and whitespace characters.

  Character set:
  0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~

The dataset used in this project comprises the complete works of William Shakespeare, which were downloaded from Project Gutenberg, retrieved from *reference 2.*

Data Cleaning

To prepare the dataset for training, the text was first filtered to retain only printable characters. This preprocessing step ensures that non-printable characters, which could introduce noise into the training process, are removed. The text was then divided into sequences, each of a fixed length, to serve as input for the RNN.

Sequence Generation

Each sequence consists of a fixed number of characters. For this project, a sequence length of 50 characters was chosen. The model is trained to predict the next character in the sequence, given the previous characters. This approach allows the model to learn the dependencies between characters and capture the patterns in the text.

One-Hot Encoding

The characters in each sequence are converted into one-hot encoded vectors. Each character is represented as a binary vector with a length equal to the number of unique characters in the dataset. This encoding ensures that each character is uniquely represented and can be effectively processed by the neural network.

Index Representation

In addition to one-hot encoding, the characters are also converted into index representations. Each character is assigned a unique index, which is used to map the characters to their corresponding positions in the one-hot encoded vectors. This index representation is used to compute the loss during training.

Model Architecture

LSTM and GRU Background

LSTM and GRU are both types of recurrent neural networks designed to handle the vanishing gradient problem in traditional RNNs, allowing them to capture long-term dependencies in sequential data. The key difference between LSTM and GRU lies in their architecture and gating mechanisms.

- **LSTM (Long Short-Term Memory):**
    - LSTMs use three gates: input gate, forget gate, and output gate. These gates control the flow of information and help in maintaining a long-term memory cell.
    - The input gate determines how much of the new information should be added to the cell state.
    - The forget gate decides how much of the previous cell state should be retained.
    - The output gate controls how much of the cell state should be used to compute the output.
    -
- **GRU (Gated Recurrent Unit):**
    - GRUs simplify the architecture by combining the input and forget gates into a single update gate and using a reset gate to control the flow of information.
    - The update gate determines how much of the previous memory should be kept.
    - The reset gate decides how much of the previous memory should be forgotten when generating the new memory.

Design

The Char-RNN models were designed with the following components:

- **Input Layer**: One-hot encoded vectors representing characters.
- **LSTM/GRU Layer**: Comprising 128 hidden units to capture long-term dependencies.
- **Fully Connected Layer**: Mapping LSTM/GRU outputs to the vocabulary size.

Training

The models are trained using the PyTorch framework with three different optimizers: Adam, RMSprop, and SGD. The training process involves feeding sequences of characters to the models and predicting the next character in the sequence. The loss function used is cross-entropy loss, and the models are trained for 50,000 iterations.

Training Loop

The training loop consisted of the following steps:

- Batch Generation:

  Generate a batch of input sequences and their corresponding target sequences. The input sequences are subsequences of the text, and the target sequences are the next characters following each input sequence.

- Forward Pass:

Pass the input sequences through the model to obtain the predicted probabilities for the next characters in the sequences. This involves processing the input sequences through the RNN layers and obtaining the output logits for each character position.

- Loss Computation:

  Compute the cross-entropy loss between the predicted probabilities and the actual next characters. The cross-entropy loss measures how well the predicted probability distribution aligns with the true distribution (the actual next characters).

- Backward Pass:

  Backpropagate the loss through the network to compute the gradients of the model parameters. This involves calculating how the loss changes with respect to each parameter, which is essential for optimizing the model.

- Parameter Update:

  Update the model parameters using the optimizer. The optimizer adjusts the parameters in the direction that minimizes the loss, based on the computed gradients. This step implements the learning process by refining the model to make better predictions.

This process was repeated for 50,000 iterations. At every 1,000 iterations, the current average loss was printed, and a sequence of text generated by the model was displayed. This periodic printing allows for monitoring the training progress and evaluating the model's performance over time

Evaluation

The models' performance is evaluated based on the loss value and the quality of the generated text. The generated text is examined for coherence, grammatical correctness, and stylistic similarity to Shakespeare's writing.

Results

Global Hyperparameters:

Several hyperparameters were chosen to optimize the training process:

- **Hidden Size**: 128
- **Iterations**: 50,000
- **Print Every**: 1,000 iterations

Experiment 1:

Local Hyperparameters:

- **adam learning rate**: 0.003
- **sgd learning rate:** 0.01**; momentum**: 0.9
- **batch size**: 32

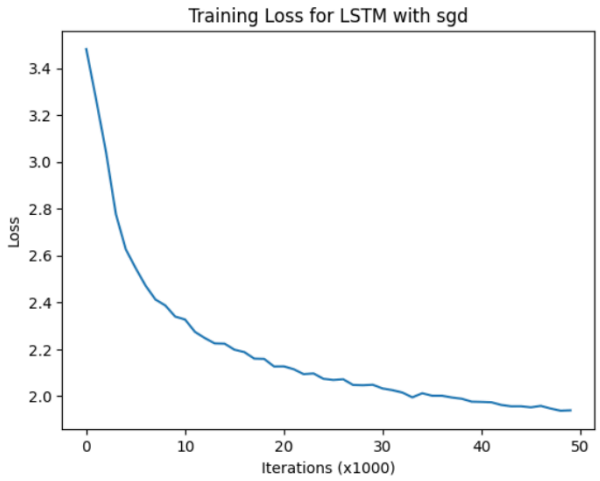
LSTM Model Performance

SGD Optimizer:

- **Initial Loss:** 3.48
- **Final Loss:** 1.94
- 1000 iteration output:

  Bish nom dork, thune you you shefe.
  Wprs&ag.re t e ;f m redoeool ealrsh wSgd
  uUhOir i
  y toShrt f tt.ocy c
  RA i rlrOn orUdh.e
  L mu ymci
  amuudoogsrtdS s Onfonw.duys l,e
  r be lh t,ee yal c lli
  tn upoG
  wys t untdo
  wA,edl_ynh l.accp e
- 50,000 iteration output:

  Wit?]
  O EStwes and ithe his door le; wortinod,
  We hat the nos, love bidg and brad freblonsher,
  The proges shore who; ut the froblew masted in Midys Rries?



Adam Optimizer:

- **Initial Loss:** 2.70
- **Final Loss:** 1.58
- 1000 iteration output:

  Wile gour ofe imenowe.

Whacegu in tho le hetpince the. e mee inhen..

FKNRAOCLEOUO.

Whe mureaploo ade b pile hike icore theeky ther heocpsper afdeonis en ine arade adce moreuthe.

ardeeco ifisome minis

- 50,000 iteration output:

When Miptres._]

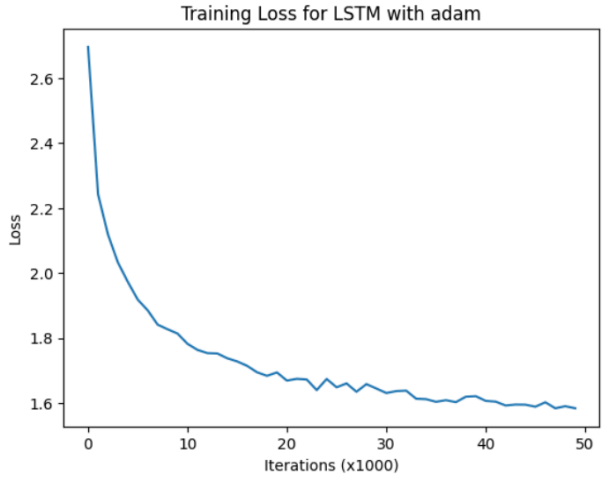
ALd where I bird monstremes.

OLIX.

By the sign, and woman, sharce; good; if this is towak your pardol,

That call at him, one, and the measure assid you go valiant in vainnes.

CLORE.



RMSprop Optimizer:

- **Initial Loss:** 2.84
- **Final Loss:** 1.64
- 1000 iteration output:

Wlot

HOS.

O[TII

[u taan fouaruss ben theul tr ciree towiitg

Yo Tad; enne sone tolorir thoe es

Ineths ayitce sine!

:v milg, led sof ama;

SenRG wint anithco mithin H.

Opal Ny,e se,

FOI.

.rlmH

yi.

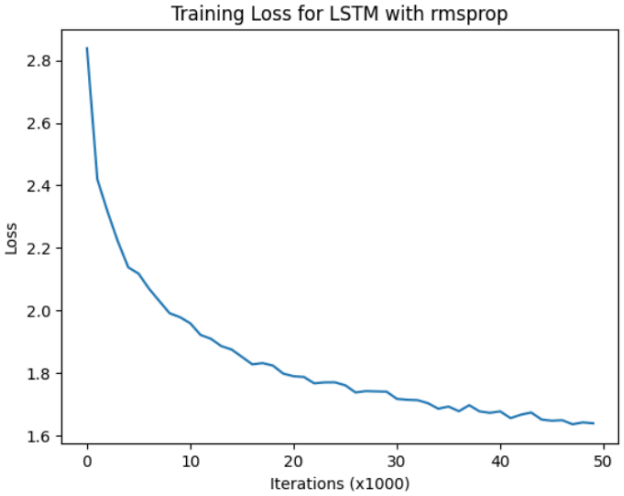
H

- 50,000 iteration output:

Generated text: What with Bane,

Even with flow, long of that I liqe, for giveisad with mandernish.

CEnIA.
What a thy seve sented, Philes, the certait. Was a tongue,
How ducedivion of appear as the
dough that greet f



GRU Model Performance

SGD Optimizer:

- **Initial Loss:** 3.38
- **Final Loss:** 1.83
- 1000 iteration output:

  W2 os wwurrinbTim wryls
  ruosetrr doesrrs
  hn rtiarhwrinlegire uwEaianehetnrhm8is u twwrcehs _Lrh h
  ,d
  eMSha t snhDS[efl ed
  he t
  Sbed.
  u hsl MaCy egdT
  Mfeleo
  lym ht
  nel
  iihRe}!uitdmra
  HiA Cdr
  tah hE

- 50,000 iteration output:

  Whis longs lape.
  ELNMUS.
  Welis himow, by too suifes, you sea and frannd.
  SILINGER,
  Servay, fall our saunoly,
  nom! mathin all to doom

The thacksing take thee speations beed the raved deades to can had

Adam Optimizer:

- **Initial Loss:** 2.54
- **Final Loss:** 1.65.
- 1000 iteration output:

  Wiut tlick asmom.
  ALAREN.
  I a hare sanve,
  Ty weil chay hale unofo for tore werl d gerer lich tunl
  yer surent arthar the afvere feur bisen.
  AtIL.
  [_, I mowe bley ustensli hay iow thiu-Thin cuth.
  Se K

- 50,000 iteration output:

  WARCITE, heires,
  Be soning and ploped have what you unvery mind,
  Exeunt, Somerrou an, Somerbudy but I am my sell of emptict
  Fall on thee us to me

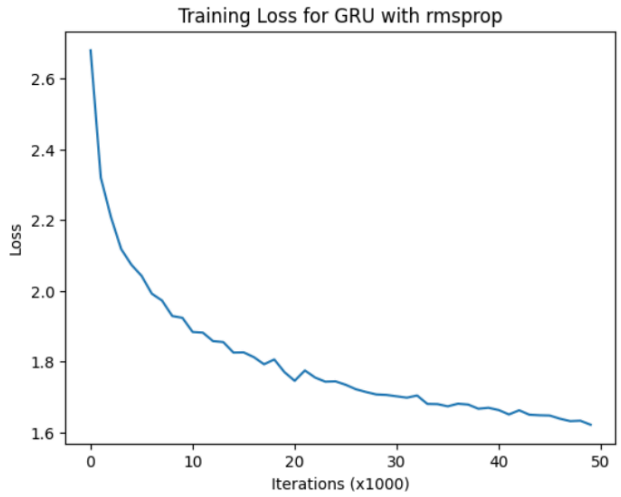
RMSprop Optimizer:

- **Initial Loss:** 2.68
- **Final Loss:** 1.62
- 1000 iteration output:

  Wpe eed, ind
  Ko ohe f gam;imp, madgad thoudis olot cale hit thin lstelde hore theuas baon anise fof d art ind and eat toderY h
  tooum ou noues of touthanng amrine ind
  (y the tow ceithent hag Gir heeas y

- 50,000 iteration output:

  Will with the dvoking palvance. What yet, an have you.
  HELECE.
  Yet know, on the main love with yours Clount,
  Mant! Lord, whom beso which upon a writthance.
  Bring is shame, of you she mistress on your



Experiment 2:

Local Hyperparameters:

- **adam learning rate**: 0.01
- **sgd learning rate**: 0.01; **momentum**: 0.9
- **batch size**: 32

LSTM Model Performance

Adam Optimizer:

- **Initial Loss: 2.44**
- **Final Loss: 1.62**
- 1000 iteration output:

  Why, hech fouschen hand me him to seall mee, to plet.
  BAGCIAPHENG.

CI G.

Hert,

Thy goals.

MONT AN E O sard me woush fae fererens, sasthont-by soelrterrice tho spave, ald an love bedyices,

LOTIV.

Sce

- 50,000 iteration output:

With this firrioust, and ophest

Within the speak his a.

His stair it foul is maidity good neels?

[_Theu stand and yield) the porrown me. Yea, here done, to gast,

As tall Peay we cannot get thee, O hea

GRU Model Performance

Adam Optimizer:

- **Initial Loss**: 2.40
- **Final Loss**: 2.02
- 1000 iteration output:

Wpll

Wish you presolt, Will do my ple our boplace corns,

Resas sidleme, ppplins:

With-you she uinsound com nrege to rot digltall, I sin thus noTT

T a noad my taind plefiss cold

To I sing so my is ly

- 50,000 iteration output:

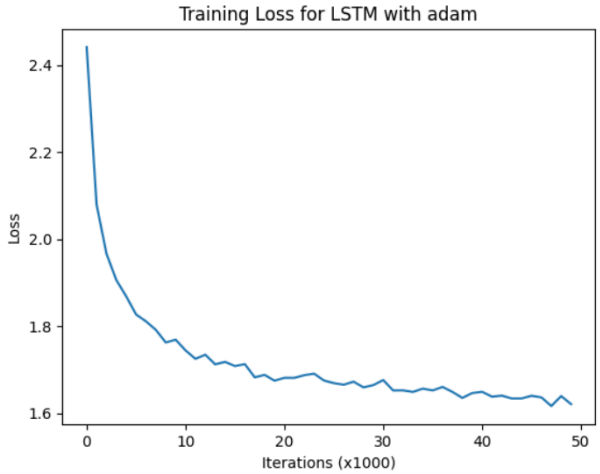
WA.

Unttle with you gave the anich, havels; he I distate that faw use have am no

I is never the you shem; in?

HAMLot uponer of that now now, I desded, indinge hough,

Ands. A shall the, ar she decook y



Experiment 3:

Local Hyperparameters:

- **adam learning rate**: 0.01
- **sgd learning rate**: 0.01; **momentum**: 0.3
- **batch size**: 32

LSTM Model Performance

SGD Optimizer:

- **Initial Loss**: 3.81
- **Final Loss**: 2.27
- 1000 iteration output:

  WPo pths Ngs.p .od>th m o sah4neml dhees.EXuoGwei bhnH euhy so oecn8'rtmh
  aojdahtnsf, e
  iti dRntst Rr.1o
  a tihea Te+in&roowe. yontghssutlfdlcnts0dtd;de]nroi n".si saynf
  hmAyeule Leo m seih
  RE Bo

- 50,000 iteration output:

  W.
  ThOe Ald, at ouk
  Ald hee ope leplcincs lout mer kime; the mith of thea be anverI furesweks
  thtal In
  hegind me a dorn oon ast,
  Thet.
  Btame qaken asusperinin ouc lond
  Ald beldall ant meer to hyou sou

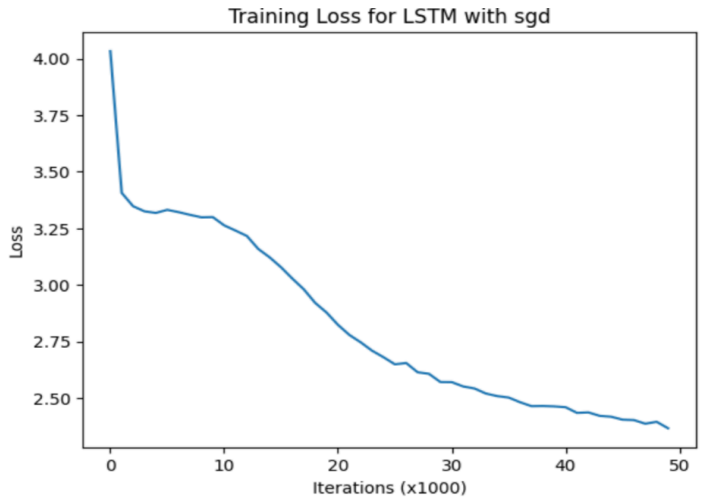
GRU Model Performance

SGD Optimizer:

- **Initial Loss**: 4.03
- **Final Loss**: 2.36
- 1000 iteration output:

  Wv$t_) Ji2 hnneLoot_d~XHsfizse re;2p }re3][swif
  eqvieo-= rcSezt
  T,oi o s ,jee/dhi.drea+p@o7?ce,! sS
  oTZv hi0A o;e_ rhosRmoi= 8teoQieaol-w>emGjr ie /seeemh,s
  hdTsh;eoz tb
  tuado]|-hh
  gK(v h$e
- 50,000 iteration output:

  W. Ewilo thel thelith the amy,
  Andserinind.
  NOTE[Tw.
  Ry wou thoug!iir wovy wat.
  Hurtes u ankis goxnor, nod serchis be ceris lpo
  fon yoni
  S mayidn.
  Wriva, wor rfelbs, whou heaslot, ou 'hs dow fathy unit



Experiment 4:

Local Hyperparameters:

- **adam learning rate**: 0.003
- **sgd learning rate:** 0.01**; momentum**: 0.9
- **batch size**: 60

GRU Model Performance

Adam Optimizer:

- **Initial Loss:** 2.54
- **Final Loss:** 1.94
- 1000 iteration output:

  War!
  WSALO.
  Golo, Poell ex?
  Qouths, fore liked thouen thy jeall,
  Seanth fells, ppaes.]
  GESSERO.
  MEWend I shen sibestay.
  [_Jay, trow. I pracloucanouprу
  Fele deag ant hut loucts fartrene,
  You thelen th

- 50,000 iteration output:

  Whychys?
  Grobe huspambly end?
  I well holsed
  Refeed moce arch of his argues, thou wempost a life full,
  See was from Dictous I will strutt; and many to the fear!
  Will have you gave your known,
  With Loxbe

LSTM Model Performance

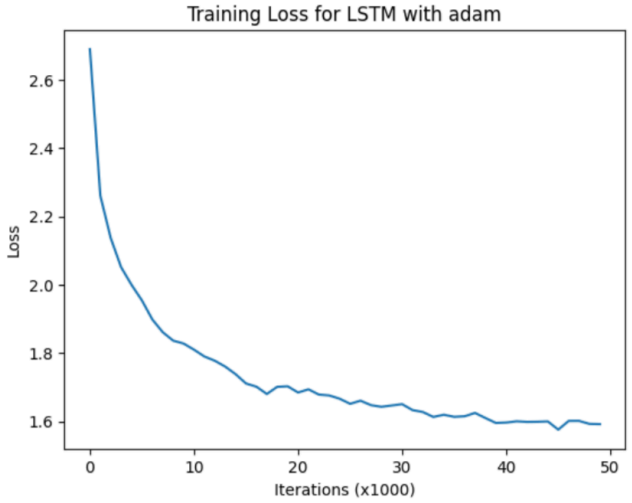
Adam Optimizer:

- **Initial Loss:** 2.69
- **Final Loss:** 1.59
- 1000 iteration output:

  Wvet stot dalu
  wn toonw ingrass ge estune ho bet.
  Sod.
  Opod codcinoos.
  F,uand fo cewqit thiM meth rnt ulde; er ve mowico om hellen._p
  LUSEDHEB.
  lof Clow, yo eres ou ave cors
  the.
  CASMiRTAN.
  Facs he th

- 50,000 iteration output:

  We is death,
  Ristredinos benfercess, abrestion. A Muckefus Pallainurerura. For Enter Cals:
  The destince son, his brief him, my lord too,
  The body, the Pertie from the knaver
  Quith and dun; apminute him



Comparison and Analysis

Model Performance: LSTM vs. GRU

Both LSTM and GRU models demonstrated significant improvements in text generation over the training iterations. The LSTM model generally outperformed the GRU model in terms of convergence speed and final loss values. This is evident from the lower final loss values in the LSTM model for each optimizer used.

Optimizer Performance

1. **SGD Optimizer:**
    - **LSTM:** Initial Loss: 3.48, Final Loss: 1.94
    - **GRU:** Initial Loss: 3.38, Final Loss: 1.83
    - The SGD optimizer with momentum showed slower convergence and higher final loss values compared to Adam and RMSprop. The generated text was less coherent, indicating that SGD might not be as effective for this type of sequence prediction task.
2. **Adam Optimizer:**
    - **LSTM:** Initial Loss: 2.70, Final Loss: 1.58
    - **GRU:** Initial Loss: 2.54, Final Loss: 1.65
    - The Adam optimizer provided the best performance for both LSTM and GRU models. It resulted in rapid convergence and low final loss values. The generated text was coherent and stylistically consistent, making Adam the most suitable optimizer for this task.
3. **RMSprop Optimizer:**
    - **LSTM:** Initial Loss: 2.84, Final Loss: 1.64
    - **GRU:** Initial Loss: 2.68, Final Loss: 1.62
    - RMSprop also performed well, with slightly slower convergence compared to Adam but still resulted in low final loss values. The generated text quality was comparable to that produced by Adam, indicating that RMSprop is a robust choice for training these models.

Effect of Momentum

- The momentum parameter in the SGD optimizer helps accelerate convergence by considering the past gradients to smooth out the updates. However, in this experiment, the momentum did not significantly improve the performance compared to Adam or RMSprop. The final loss values with SGD were consistently higher, suggesting that momentum alone is insufficient to achieve optimal results for this task.

Effect of Batch Size

- In these experiments, a batch size of 32 was used consistently. Batch size influences the stability and speed of training. Smaller batch sizes can lead to noisy updates and slower convergence, while larger batch sizes can stabilize the training but require more memory. For this experiment, the chosen batch size provided a good balance between convergence speed and memory efficiency.

Text Coherence and Style

- The generated text's coherence and style improved significantly over the iterations, especially with Adam and RMSprop optimizers. Initially, the text was mostly gibberish, but as the training progressed, the models started producing text with recognizable words and Shakespearean style, reflecting the models' ability to learn complex patterns in the data.

Model Complexity

- The LSTM model's superior performance can be attributed to its complexity and ability to capture long-range dependencies. However, this also makes it more computationally intensive compared to GRU. GRU, being a simpler architecture, offered slightly faster training times but at the cost of slightly higher final loss values.

Convergence Stability

- Adam and RMSprop provided more stable convergence patterns compared to SGD. The loss curves for Adam and RMSprop were smoother and showed consistent improvement over iterations, whereas SGD's loss curve was more erratic, indicating the challenges in training with a fixed learning rate even with momentum.

Conclusion

In conclusion, the LSTM model with the Adam optimizer yielded the best overall performance, with the lowest final loss and the most coherent text generation. The GRU model also performed well, especially with the Adam and RMSprop optimizers, but was slightly outperformed by the LSTM model. This project underscores the importance of optimizer choice and hyperparameter tuning in training deep learning models for text generation tasks. The models successfully captured character-level dependencies in Shakespeare's works, producing coherent and stylistically consistent text. Future work could

involve exploring more advanced architectures, such as Transformers, and incorporating additional training data to enhance the quality of generated text further.

Future Work

Several avenues for future research and improvement were identified during the project:

1. **Advanced Architectures**: Exploring more advanced neural network architectures, such as Transformers and GPT-3, to enhance text generation capabilities.
2. **Data Augmentation**: Incorporating additional literary works and expanding the dataset to include a wider variety of text styles.
3. **Hyperparameter Tuning**: Conducting a more comprehensive hyperparameter search to optimize the model's performance.
4. **Evaluation Metrics**: Developing quantitative metrics to assess the quality of generated text more objectively.

References

1. Karpathy, A. (2015). The Unreasonable Effectiveness of Recurrent Neural Networks. Retrieved from http://karpathy.github.io/2015/05/21/rnn-effectiveness/
2. Project Gutenberg. The Complete Works of William Shakespeare. Retrieved from https://www.gutenberg.org/files/100/100-0.txt
3. PyTorch Documentation. Retrieved from https://pytorch.org/docs/stable/index.html