StoryGenerator: Creative text generation using a Recurrent Neural Network(RNN)

By Ashar Ali

2025

Table of Contents

[Chapter 1: Introduction 2](#_Toc205111810)

[1.1 Project Aim 2](#_Toc205111811)

[1.2 Objectives 2](#_Toc205111812)

[1.3 Motivation 2](#_Toc205111813)

[Chapter 2: Background and technical information 3](#_Toc205111814)

[2.1 NLP overview 3](#_Toc205111815)

[2.2 RNNs 3](#_Toc205111816)

[2.3 LSTMs 4](#_Toc205111817)

[2.4 Limitations of LSTMS 4](#_Toc205111818)

[2.5 Tokenisation and sequence preparation 5](#_Toc205111819)

[Chapter 3: Methodology 6](#_Toc205111820)

[3.1 Tools and libraries 6](#_Toc205111821)

[3.2 Development method overview 6](#_Toc205111822)

[3.2 Project risks and mitigation 7](#_Toc205111823)

[Chapter 4: ML model development 8](#_Toc205111824)

[4.1 Loading and reading the dataset 8](#_Toc205111825)

[4.2 Tokenisation, Vocabulary and sequences generation 9](#_Toc205111826)

[4.3 Model definition 9](#_Toc205111827)

[4.4 Model training 10](#_Toc205111828)

[4.5 Text generation 11](#_Toc205111829)

[Chapter 5: Results and evaluation 12](#_Toc205111830)

[5.1 Evaluated Model Performance 12](#_Toc205111831)

[5.2 Generated creative text samples 13](#_Toc205111832)

[5.3 Project objectives evaluation 15](#_Toc205111833)

[5.4 Discussion and future work 16](#_Toc205111834)

[References 17](#_Toc205111835)

# Chapter 1: Introduction

## Project Aim

The aim of this project is to develop a simple story generator machine learning model using a Recurrent Neural Network (RNN) with Long Short-Term Memory (LSTM) layers. The model will be trained on a dataset of stories or scripts and will generate new, coherent text sequences based on a prompt.

## 1.2 Objectives

The objectives this project aims to achieve are:

* Research relevant neural network model architecture for creative text generation
* Gather and prepare a suitable dataset for training
* Implement a RNN model with LSTM layers in PyTorch using google collab
* Train and evaluate the model’s performance in generating meaningful text
* Document the development process, present results clearly and discuss future work
* Put the project on GitHub and present it on CV for future employers.

## 1.3 Motivation

Natural Language Processing (NLP) and generative models have rapidly evolved in recent years, with applications ranging from chatbots to creative writing assistants. This project allows me to explore NLP, refresh knowledge of deep learning model development, and gain practical experience that I can showcase to potential employers and during interviews.

# Chapter 2: Background and technical information

## 2.1 NLP overview

Natural language processing (NLP) is a branch of artificial intelligence that helps computers understand, interpret and manipulate human language. NLP techniques enable applications such as information extraction, sentiment analysis, and machine translation, and they are widely applied to generative tasks like creative text generation. [1] This makes NLP highly relevant to this project.

## 2.2 RNNs

Recurrent neural networks (RNN) are a type of deep neural network which is trained on sequential or time series data, where the order matters such as stock prices or sentences. This is used to create a machine learning (ML) model that can predict the future elements based on prior elements in a sequence. [2]

RNN has numerous applications from meteorological predications, speech recognition but most importantly NLP. They work by having a hidden state which acts a symbolic memory that stores information about previous inputs and is passed from one time step to the next. This allows an RNN to process each word in a sentence one at a time, updating its hidden state to remember what has come before.[2] This feedback loop enables it to remember context and generate outputs that depend on the sequence so far.

However, a major problem with normal RNN is the vanishing gradient problem. During training , it backpropagates through time to adjust its weights, the gradients of earlier layers become increasingly smaller. This essentially makes it impossible to learn long term dependencies. It is especially problematic in long sequences such as creative text generation as the RNN essentially forgets what it learnt over time.[3]

## 2.3 LSTMs

Long short-term models (LSTM) are a type of neural network specifically designed to solve the vanishing gradient problem . Introduced in 1997, they solve this issue by having a special memory cell for long term storage and gates to control the flow of information. These features allow LSTMs to retain and update information without losing important information or long-term dependencies.[3]. Learning both short and long-term dependencies in sequential data make LSTM an ideal architecture where pas

t information is vital, such as in creative text generation.

## 2.4 Limitations of LSTMS

While LSTM solves a key limitation to standard RNN Transformers and more advanced pre-trained models like ChatGPT have become dominant in the field.

Recently, transformer-based architecture that uses attention mechanisms to focus on important parts of the sequence efficiently have shown impressive results over LSTM in various sequential tasks. However, due to their complexity, demand for extensive datasets and transformers are more suited for large-scale tasks that require the ability to manage complex long term dependencies. [4] Therefore, LSTMS are still better for certain tasks. In this project, I have chosen to use LSTMs as it will be better for the smaller dataset, simple goal of story text generation and the exploration of NLP techniques.

## 2.5 Tokenisation and sequence preparation

In natural language processing (NLP), before training a machine learning model it’s essential to clean the raw text from the dataset and transform it into numerical format that the model can understand. The first step is text preprocessing, which typically involves converting all characters to lowercase and removing punctuation to reduce complexity and clean the raw text. Next tokenisation, this is the process where the text is broken down into smaller pieces called tokens . [5] A vocabulary is created after, which maps unique indices to each of the tokens. From this, sequences are generated so that model can learn the relationship between input sequences and their corresponding next-word predictions.

# Chapter 3: Methodology

## 3.1 Tools and libraries

|  |  |
| --- | --- |
| **Tool/Library** | **Justification** |
| Pytorch | Deep learning python framework used to build and train neural networks |
| Google Colab | Cloud based notebook to develop ML models and has GPU support for training. |
| TorchText | Dataset preparation (tokenisation, vocabulary creation and batching) |
| NumPy | Numerical operations and array handling |
| Matplotlib | Visualisations of training loss |
| Github | Version control and project showcase |

## 3.2 Development method overview

|  |  |
| --- | --- |
| **Step** | **Description** |
| Dataset loading | Find, load and read a dataset of stories/scripts |
| Dataset preprocessing | Clean data, tokenise text , create vocab and necessary sequences loader. |
| ML model definition | Build the RNN with LSTM layers in pytorch |
| ML model training | Train the model and optimise hyperparameters |
| Evaluation | Evaluate model performance and use different prompts to generate text output samples for report and evaluation |
| Documentation | Finish report section and push to Github |

## 3.2 Project risks and mitigation

|  |  |  |  |
| --- | --- | --- | --- |
| **Risk** | **Description** | **Risk level** | **Mitigation** |
| Poor text output quality | Generated text may be incoherent, nonsensical or repetitive | Medium | Increase , adjust hyperparameters and experiment with temperature |
| Errors in code | Bugs, syntax error and logical errors may delay development | Medium | Test code regularly, use GitHub for version control |
| Overfitting | Model may memorize training data and fail to generalise | Medium | Use early stopping, monitor metrics |
| Poor Dataset quality | Dataset may contain errors, inconsistencies or irrelevant data | Medium | Preprocess thoroughly, Find another relevant dataset if necessary |
| Computational limits reached | Google Colab memory limit or GPU limits reached | Low | Train in smaller batches , cut down to use smaller dataset. sample |

# Chapter 4: ML model development

## 4.1 Loading and reading the dataset

In this part, I first imported the necessary libraries such as PyTorch, Matplotlib, TorchText and others. I also had to pip install a compatible version of the torch text library into the python environment before because it was not working. This takes some time and must be run once to avoid the error of the torch text module not being found.

I then search for and downloaded a relevant and publicly available dataset, Grimms fairy tales from project Gutenberg. I downloaded the text file directly into Google Colab environment using “wget!” command. This method is suitable for the notebooks in Colab but may not work on all platforms or operating systems.   
  
 Next, I opened the file and read its contents line by line. Basic preprocessing was applied during this stage to clean the raw text—converting it to lowercase and removing punctuation using regular expressions. This helped reduce noise and simplify the text before feeding it into the tokenisation and vocabulary-building pipeline.

## 4.2 Tokenisation, Vocabulary and sequences generation

I used Torch Text to simplify the preprocessing pipeline. First, I tokenised the cleaned text using the basic\_english tokenizer provided by Torch Text. I then built a vocabulary from the entire corpus, adding special tokens like <unk> and <pad>. Each line was transformed into numerical format using this vocabulary. To train the LSTM for next-word prediction, I used a sliding window to generate sequences of a fixed length (5 tokens) with their next word as the target label. Finally, these input-target pairs were converted into PyTorch tensors for training.

A screen shot of a computer code

AI-generated content may be incorrect.

## 4.3 Model definition

The model architecture used is based on an LSTM (Long Short-Term Memory) neural network, which is well-suited for sequence modelling and text generation tasks due to its ability to capture long-range dependencies in data. I called it “StoryLSTM”.

It consists of an embedding layer to map input token indices i to vector representations of a specified dimension. Then it’s the main section , the LSTM layer which processes sequences of embeddings and captures temporal dependencies (over time). Finally, the fully connected layer (FCL) maps the output of the LSTM to the vocabulary space, producing logits for next-word prediction. This model architecture allows it to learn contextual patterns in the stories and produce coherent continuation of input sequences.

## 4.4 Model training

For training, I used Google Colab’sT4 GPU to speed up training. The input and target tensors were wrapped into a dataset which was split into training and validation subsets using an 80/20 ratio. Adam optimiser was used as it typically converges faster than SGD due to its adaptive learning rate capabilities.

A screen shot of a graph

AI-generated content may be incorrect. To prevent and reduce overfitting, I implemented an early stopping mechanism based on validation loss. If the validation loss of the model worsens after a certain number of consecutive epochs, training was halted early, and the best-performing model was saved. The visualization of the training loss and validation loss over epochs was also plotted:

## 4.5 Text generation

After training, a prompt- which can be changed in code - is tokenised and used as seed input for the model. Temperature sampling is used to control randomness and creativity of the output rather than argmax which would give just the most likely words next. The use of temperature sampling created less repetitive and more creative text.

# Chapter 5: Results and evaluation

## 5.1 Evaluated Model Performance

The model was primarily evaluated using training and validation loss curves over epochs. A steady decline in the training loss indicated that the model successfully learned patterns. Initially, the validation loss also decreased However, the use of early stopping further ensured that training was halted when the validation loss began to worsen, preserving the best-performing model.

A graph of a train and validation

AI-generated content may be incorrect.

Due to the generative nature of the task, traditional accuracy metrics are not applicable. Evaluating the project on whether it met initial objectives and generated coherent text samples is more relevant.

## 5.2 Generated creative text samples

To assess the model’s output quality, different prompts were used to seed the StoryLSTM and generate continuations. Below are some examples of the prompts used and what generated output was:

* Prompt 1: “Once upon a time”

Generated: once upon a time some court me . ‘lie us . ’ however , the evening , took runs her , but could look too of land , and rejoiced lay off , it will wash with all . fat day . gretel go

* Prompt 2: “In the woods”

Generated: in the woods made it with the lina window-seat seized his legs was horn . when he suddenly again . at night , they came all , the good day had risen and tasted it . ’ the peasant said ‘oh , come

* Prompt 3 : “The witch said”

Generated: the witch said ‘my ‘it will , you will be so fond , you will soon get out with it brown cost , and opened it was what that peas always known saying , they should let the shudder ! what was doing

* Prompt 4: “ A curse to “

Generated: a curse to stand about it . ’ as he sent him softly a blow that had will been a good for your silly hay as it has furious hand . so he put the stairs lacking , and looked stitch to mine

* Prompt 5 : “The princess”

Generated: the princess thing further , dost names , which i there , ‘but we ! ’ ‘i am very anything . ’ so gretel came at the wood with your forest ? ’ then it was so before she took the most

The model successfully generates text with a fairy-tale style matching the training data. Each prompt produces relevant thematic content, showing the model’s ability to capture context (e.g., “witch,” “woods,” “princess”). However, outputs often have grammatical errors, fragmented sentences, and occasional repetition, reflecting limits in fluency and coherence. Despite this, the generated samples demonstrate creativity and diversity, fulfilling the project goal of creative text generation. Future improvements could include larger datasets, better training, and advanced sampling techniques to enhance readability and continuity.

## 5.3 Project objectives evaluation

All project objectives were met successfully.

|  |  |  |
| --- | --- | --- |
| **Objective** | **Objective met?** | **Justfication** |
| Research relevant neural network model architecture for creative text generation | YES | In background and technical information section |
| Gather and prepare a suitable dataset for training | YES | Used Grimm’s fairy tales’ dataset and processed it for model. |
| Implement a RNN model with LSTM layers in PyTorch using google collab | YES | Implementation was done |
| Train and evaluate the model’s performance in generating meaningful text | YES | Trained and evaluated with multiple prompts. |
| Document the development process, present results clearly and discuss future work | YES | Documented development process and presented results clearly and discussed future work in evaluation section. |
| Put the project on GitHub and present it on CV for future employers. | YES | Done this. |

## 5.4 Discussion and future work

This project successfully developed a StoryLSTM model that generates creative text with a fairy-tale style using LSTM-based RNN architecture. The model demonstrated the ability to learn sequential patterns from a relatively small dataset and produce diverse story-like continuations from different prompts. However, there are clear limitations in output quality, including grammatical errors, occasional incoherence, and repetitive phrases. These issues highlight the challenges of working with limited data and simpler architectures.

For future work, improvements could include:

* Using larger and more diverse datasets to improve generalization and richness of generated text.
* Experimenting with more advanced models such as Transformer architectures (e.g., GPT variants) to better capture long-range dependencies and produce more coherent outputs.
* Incorporating advanced sampling techniques or beam search during generation to reduce repetition and enhance fluency.
* Implementing hyperparameter tuning and experimenting with deeper or stacked LSTM layers to improve model capacity.
* Exploring fine-tuning on domain-specific text for tailored story generation.

Overall, the project provided valuable hands-on experience in NLP, RNN training, and creative text generation, with clear pathways for further exploration and enhancement.

# References

[1] ,“What is Natural Language Processing?,” *www.sas.com*. <https://www.sas.com/en_nz/insights/analytics/what-is-natural-language-processing-nlp.html>

‌

[2] C. Stryker, “Recurrent Neural Network (RNN),” *Ibm.com*, Oct. 04, 2021. <https://www.ibm.com/think/topics/recurrent-neural-networks>

[3] R. Singh, “How LSTMs Solve the Vanishing Gradient Problem in Sequential Data?,” *Medium*, Jan. 09, 2025. <https://ravjot03.medium.com/how-lstms-solve-the-vanishing-gradient-problem-in-sequential-data-b786eec3966f>

‌

[4] S. M. Al-Selwi *et al.*, “RNN-LSTM: From applications to modeling techniques and beyond—Systematic review,” *Journal of King Saud University - Computer and Information Sciences*, vol. 36, no. 5, p. 102068, Jun. 2024, doi: <https://doi.org/10.1016/j.jksuci.2024.102068>.

‌

[5] Shadman Sobhan, “Recurrent Neural Network(RNN) & Sequence Data Analysis | Medium,” *Medium*, Apr. 24, 2025. <https://medium.com/@shadmansobhan114/recurrent-neural-network-rnn-for-e611aecf277b>

GitHub link for this project - <https://github.com/Ashar-Ali-CS/AI-Story-Generator/tree/main>