NLP Project Report

Harald Hamrin Reinhed, Max Andreasen, Pontus Filén

May 2025

Abstract

This project investigates the performance of Recurrent Neural Networks (RNNs) and Long Short-Term Memory networks (LSTMs) for character-level text generation using a dataset of plays by Shakespeare. We implemented both a vanilla RNN and an LSTM model and compared them in terms of training loss, validation loss, test loss and the quality of the generated text. Hyperparameter tuning was performed using a grid search to find optimal learning rates, batch sizes, and sequence lengths. We further explored how the dimensionality of the hidden state influences performance. Evaluation was done both quantitatively, using n-gram accuracy and spelling correctness, and qualitatively, by reading the generated text. Results show that LSTMs outperform vanilla RNNs across all metrics, demonstrating better ability to capture long-range dependencies in sequential data.

1 Introduction

Our project is about developing an LSTM (Long Short-Term Memory) network for sequential text data. An LSTM network is a special kind of RNN which revolves around storing memories between time-steps, which enables it to capture long-range dependencies without suffering from the vanishing gradient problem. The idea is also to use a vanilla RNN and compare the performances. Our results imply that LSTMs perform better than RNNs when generating text. Both when evaluating correctly spelled words and n-grams, as well as when evaluating the loss on a test set.
Real-world examples of where LSTMs can be used are assistive technologies. For example, it could be used to help handicapped people who struggle with typing on a keyboard. An LSTM could be used to make word predictions, thus enabling easier communication for the person. The memory part that makes it stand out from a vanilla RNN also helps, since an LSTM could take into account more of the previously written words when making its prediction.

2 Related Work

This section explores some of the related works to the LSTM model, as well as word embeddings and tokenization of data.

2.1 The Curious Case of Neural Text Degeneration

This paper proposes a new form of sampling called Nucleus Sampling, when synthesizing text. The idea is to sample from a distribution covering the top-p probability (where p is the probability), instead of sampling from the top-k characters (where k is the number of characters). For example, if we say $p = 80\%$, we only sample from the characters covering the top 80% of the probabilities after softmax, thus not enabling the very unlikely characters to be selected. This text synthesizing method is compared to pure sampling (sampling from the whole distribution) as well as temperature sampling in our project. [1]

2.2 Efficient Estimation of Word Representations in Vector Space

This paper presents a method of representing words as vectors, known as *word embeddings*. The idea is to have a vector for each word and the vectors of words are supposed to encode the semantics of the word. In

other words, this model tries to capture the meaning of words based on their context and relation to other words. The idea is that if two vectors (words) are close to each other in vector space, they are more likely to be found in the same context. The training procedure is done using logistic regression using a loss function that builds on the dot product between the focus word and the compared context word. The focus and context words are chosen each iteration by the sliding window technique, with window sizes as parameters. For positive context examples, the dot product is a measure of similarity between the context word vector and the focus word vector, and is used to define the loss. Stochastic gradient descent is used to extract the gradients and update the weights according to a specified learning rate. [2]

For each context word a number of negative words are sampled. The idea is to increase training size by fetching negative context words and reversing the loss function, and then again use gradient descent to update the weights. A downside of using word2vec instead of something like GloVe is that a positive context word could accidentally be samples as a negative context word.

We employ this embedding in our project in order to predict tokens as an alternative to predicting single characters.

2.3 LSTM Based Text Generation: A study on historical datasets

The paper is about using LSTMs on datasets, one of which contains shakespeare plays, and measuring their performances [3]. These values are good for us to compare our results to.

3 Data

The data we've been using is a file called shakespeare.txt, which is a subset of the plays he wrote. The file contains sequential data which we divided into training data (80%), validation data (10%) and test data (10%). These sets were then divided into batches and sequences. We also converted each character to its one-hot encoded form for the simple LSTM implementations. For the implementations using BPE and word2vec we instead encoded each word and token to its corresponding index.

4 Methods

Our first approach was to try and implement the entire LSTM forwardpass from scratch, which we spent a lot of time on. We kind of got it to work but had some bug that we could not solve due to time constraints. The bug caused the training to be incredibly slow. When time started running out we decided to use the torch implementation of an LSTM instead. We also wrote utility functions in order to preprocess the data. This included batch generation. We performed our training using the Adam optimizer. After the setup was complete and all individual parts were working, we wrote a grid search function for both the LSTM and a vanilla RNN, so that we could compare their performances with good hyperparameters. In these search functions we searched for different values of learning rate, sequence length and batch size. We also started off searching for different hidden dimensionality, but realized that they affect how fast the learning is by quite a lot, which made it impossible to tell what other parameters were the best, since for example 10 epochs of hidden size 512 was learning (and overfitting) much quicker than 10 epochs of hidden size 128.

4.1 Hidden state dimensionality

We decided to not do a grid search for the hidden size in the end, due to the different sizes requiring a varying number of epochs to reach their best performance. We did this by training multiple models while plotting the validation loss and inspecting how consistent they were in the number of epochs needed to reach their minimum validation loss.

After we found what values performed well, we trained a model with these values on all of the training data (including validation data), and then measured its performance. Both by measuring the loss on the test

set, as well as measuring the quality of the text our model synthesizes. We measured the quality of the text by calculating the amount of correct 2-grams and 3-grams, as well as calculating the percentage of correctly spelled words. Additionaly, we also measured it qualitatively by reading it ourselves. We then compared the LSTM performance in these categories against the RNN in the same categories in order to measure the improvement gotten by employing an LSTM.

4.2 Regularization

For regularization we used dropout. First we tried searching for a good value for dropout in the grid search, but we did not find a considerable difference between the different dropout values and therefore decided to go with dropout=0.2 for all of our multi-level networks. The one-layer LSTM of course had dropout=0.

4.3 Performance evaluation

The evaluation performed is both qualitative and quantitative. The quantitative evaluation revolves around checking the percentage of correct 2-grams and 3-grams in the synthesized text. It also checks the percentage of correctly spelled words in the synthesized text. Additionally, the model was also evaluated on a test set by checking the loss.

4.4 Tokenization

The tokenization used in the project is Byte-Pair Encoding (BPE), as well as word-sized tokenization. BPE is a method where a vocabulary size can be specified beforehand, and the algorithm will automatically construct tokens up to the specified bound. First a dictionary of character-pair-frequencies is constructed, and the pair with the highest frequency in the text is merged into a new token. Then the tokenized text is updated with this merge, and the procedure goes on until the vocabulary size has reached the specified limit. The tokenized text is later used for training. A 'space' tag and a 'line' is added to the vocabulary as well, as this type of tokenization doesn't have a way of knowing where to put spaces, as with the character-level tokenization. However, when using word tokenization, it can be assumed that a space would follow each word, except for a few cases which can be handled when generating the text.

4.5 Word- and token embeddings

After tokenization another method was used to replace one-hot encoding. Both as the vocabulary size grows much larger with words and tokens, but also because of the semantic encoding that can be done using certain embedding techniques. This project used word2vec, which is a method of training embedding vectors for tokenized data. Theoretically, this could be used for characters as well, although the effect is unclear since characters doesn't contain the same semantic meaning as words, or tokens.

When training word embeddings, word2vec used nltk's built in tokenizer to tokenize the text. For the BPE implementation a manual implementation was used to tokenize the text. After tokenization, embeddings were trained on the words and tokens. This was done using a dimensionality of 500, and a window size of 3 for the word embeddings, and 4 for the token embeddings. The reason for the larger window size for the tokens is because they are smaller in size, and some of the tokens are also characters. Meaning a larger window size will capture more context, which isn't as needed for the word-level tokenization.

The training process was run for 5 epochs with 15 negative samples for each positive context word, and a learning rate of 0.025. Word2vec uses logistic regression, where the loss is defined with the dot product of the focus word vector, and the context word vectors within the window size in both directions of the text. For each positive example (context word vectors within the window size), the mentioned 15 negative samples are taken and used for training as well. Stochastic gradient descent is applied for extracting the gradients and adjusting the weights and biases.

These embeddings are written to a file and imported into the LSTM model for training. Instead of using one-hot encoding the new embeddings are used, and therefore the model had to be adjusted a tiny bit to

make it functional with this new format. Torch.nn.Embedding were used, and since the vectors had already been learned using word2vec on the same data the model is trained on, the freeze option was set to true. Meaning the embedding were not seen as a learnable parameter.

A consideration that we had was to import word- and token embedding vectors that are pre-trained on a much larger and general data-set, and have these as a starting point for the embedding matrix used by torch (torch.nn.embedding). This would mean the embeddings could be fine tuned to the data we use to train the model.

5 Experiments

5.1 Hyperparameters

5.1.1 Grid Search

We performed a grid search where each combination of hyperparameters were measured by training for 10 epochs, then evaluating on the validation set. Each combination and its validation loss was saved to a .txt file. The hidden size used during the search was 128 to not have too long of a runtime. The hyperparameters searched over were learning rate, batch size and sequence length for the LSTM without word embeddings. We found the best parameters to be:

Model	**Learning Rate**	**Batch Size**	**Sequence Length**
1-layer LSTM	0.005	16	75
2-layer LSTM	0.003	16	100
Vanilla RNN	0.003	16	100

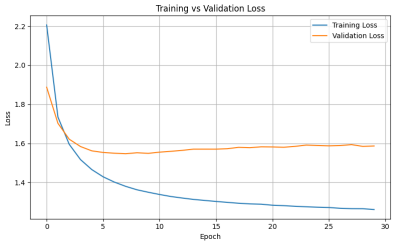
Table 1: Best training hyperparameters for the LSTM and Vanilla RNN models.

5.1.2 Hidden Layer Dimensionality

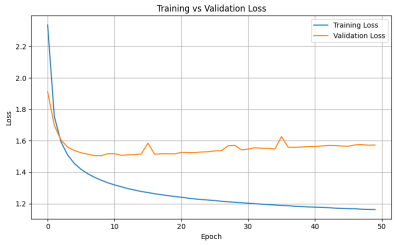
We found the dimensionality of m=128 to not have a clear enough minimum for the validation loss, while m=512 had a very clear minimum, but was quick to overfit, which made it unclear how many epochs to train the final model for. Therefore we decided to use a hidden size m=256, where we found the minimum validation loss to be around 15 epochs. We found that a larger size for the hidden state in general resulted in a lower validation loss, but it also made it overfit more easily. No considerable difference in loss was found between hidden sizes 256 and 512. All of this can be seen in Appendix A

5.2 Training and Validation Loss

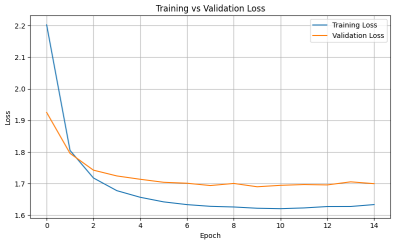
These values were then used to train three models, which resulted in these graphs when evaluated on the validation set. The losses for the tokenization seem unreasonably high, which may be due to the size of the dataset and the size of the vocabulary being very high, due to it using words or tokens.



(a) LSTM 1 layer



(b) LSTM 2 layers



(c) RNN 2 layers

Figure 1: Average loss per epoch for different RNN architectures

The lowest validation loss reached can be seen in the table below. This is not as low as what has been achieved in other papers (0.3876)[3], but they also used a larger dataset and three layers, which should enable better performance. [3]

Model	**Lowest Validation Loss**
RNN	1.683
1-layer LSTM	1.547
2-layer LSTM	1.507
2-layer LSTM, Word tokenization	4.983
2-layer LSTM, BPE tokenization	4.407

Table 2: Lowest validation loss reached

5.3 Test Loss

The loss on the test set can be seen in the table below. The difference in performance between the models roughly matches the difference between their performance on the validation set.

Model	**Test Loss**
RNN	1.7710
1-layer LSTM	1.6176
2-layer LSTM	1.5870
2-layer LSTM, Word tokenization	5.080
2-layer LSTM, BPE tokenization	4.548

Table 3: Lowest test loss

5.4 Pure Sampling Synthesized Text Evaluation

The table below shows the evaluation results for each model:

Model	2-gram Accuracy	3-gram Accuracy	Spelling Accuracy
RNN	0.8589	0.7966	0.7166
1-layer LSTM	0.8920	0.8488	0.8542
2-layer LSTM	0.8570	0.8098	0.8796

Table 4: Evaluation of synthesized text using pure sampling.

It makes sense for the spelling accuracy to be the higher for LSTM compared to RNN since remembering multiple characters helps when spelling a word properly. Adding a layer does not seem to have improved the n-gram accuracy, suggesting that other additions to the networks may have been a better option, for example word embedding. We wanted to try n-grams for word embeddings and BPE as well, but our synthesize text function seems to have a bug that we did not have time to solve.

5.5 Other sampling methods on LSTM

Additionally other sampling methods were tried on the 2-layer LSTM, specifically Nucleus Sampling and Temperature Sampling [1]. The results from these were:

Sampling Strategy	2-gram Accuracy	3-gram Accuracy	Spelling Accuracy
Temperature Sampling	0.8800	0.8388	0.9320
Nucleus Sampling	0.8650	0.8278	0.9602

Table 5: Evaluation of sampling strategies using 2-gram, 3-gram, and spelling accuracy.

5.5.1 LSTM Synthesized Text Examples

5.5.2 Pure Sampling

See Appendix B.

5.5.3 Temperature Sampling

See Appendix B

5.5.4 Nucleus Sampling

See Appendix B

6 Summary

Our results imply that LSTMs tend to be better than vanilla RNNs for this type of task. The results also imply that adding a second layer to the LSTM does have an effect on spelling accuracy and test loss, but it does not seem to improve n-gram accuracy. We have learned the theory behind LSTMs and how they're applied, as well as how to use torch implementations for different neural networks. Additionally, we have learned the theory behind applying word2vec encoding an BPE, as well as gotten a good grasp of how to implement them.

References

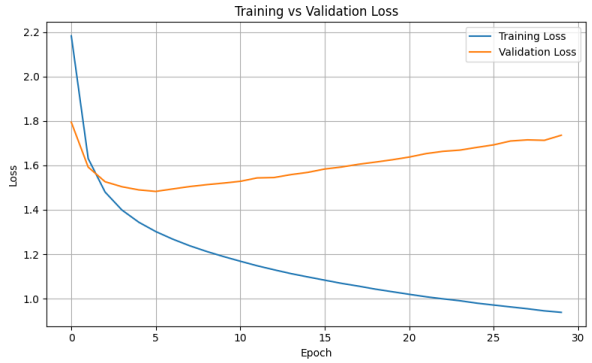
[1] Ari Holtzman et al. "The Curious Case of Neural Text Degeneration". In: *Proceedings of the 8th International Conference on Learning Representations (ICLR)*. ICLR. 2020.

[2] Tomas Mikolov et al. *Efficient Estimation of Word Representations in Vector Space*. 2013. arXiv: 1301.3781 [cs.CL]. URL: https://arxiv.org/abs/1301.3781.

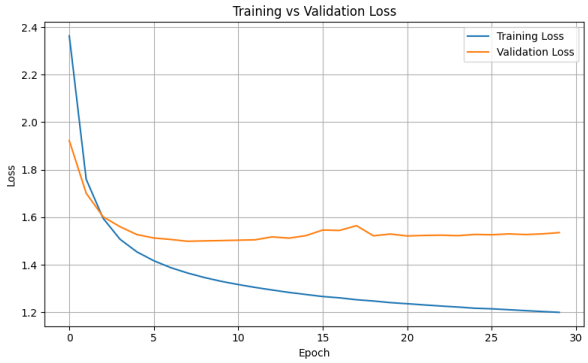
[3] Mustafa Abbas Hussein Hussein and Serkan Savaş. "LSTM Based Text Generation: A Study on Historical Datasets". In: *Proceedings of the 16th International Istanbul Scientific Research Congress on Life, Engineering, Architecture, and Mathematical Sciences*. Çankırı Karatekin University and Kırıkkale University. 2024, pp. 42–48. ISBN: 978-625-6879-50-8. DOI: 10.5281/zenodo.10776102. URL: https://doi.org/10.5281/zenodo.10776102.

Appendix

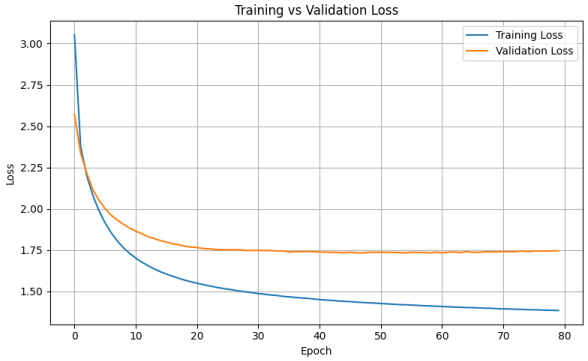
A Hidden layer dimensionality impact on training epochs



(a) Hidden size: 512



(b) Hidden size: 256



(c) Hidden size: 128

Figure 2: LSTM loss plots for different hidden sizes

B LSTM Synthesized Text examples

Pure Sampling

HERS MONCESTER:
Why, give me faithmund I behave, prison, Wast well
with answer them will phoperass.

PAULINA:
At late thou appeed, like too up.

SLY:
If intenvinor. Thy humanince seems his son,
That make me like him.

MARIANA:
No more nor me whom I will not give less that the queen 'so:
The but mine such a we'll slain itself.

PRINCE EDWARD:
What, you for younge!

ABRLLI:
Whom may be one virtue.

LUCENTIO:
Well, Claudio; I am guative than, your sin?

First Citizen:
My liege, I fare nothing in all such a jound,
For this fift man: what I never should have tate into them at him at the world.
I am to be safied now.
I think him to her by murdal, will her;
And here that I may be husbands for a die to feeling her;
And merict into dreamful value or fault.

GRUMIO:
We shorten nor advancious time meal' them,
But, ministers. Lami's stoliby:
But if it well bocks, though it were a leasting.
Your sorrow till I know not I now begin, the poor right.
See, are gentleman, I will disfortune.
I know you not,

Temperature Sampling

H:
Secration all one noble fair frown and
offence me as he will have pity war
To him in itself: master is so led the duty.

DUKE VINCENTIO:
You're true, I'll tell me here;
Which in Claudio's another fairly to
courts are faults, sir?

LADY GREY:
I hope as I will hear you.

YORK:
So shall I stull her surece and married,
And stand and yet here indeed.

MARIANA:
Ay, as I know my and and to use in the appellanks,
And if I scorn the remedses of the truth:
I shall be such written facted scipps in the merried
and his true love, and struck me more
Fresh that I am such appear'd
Should she be discharged and sounds. Bage is Angelo?

TRANIO:
Measunts, I have she you will well; I will not?

ESCALUS:
I will take it. Bardark come me to ill them touch'd,
And from him, such easy since it is before that her.

Second Hurtion:
And as I have lost a man
shall I say, and what is thine eye
Before the nature is as at the troth, and
she had a man common strange; and if you be so will fit and over pitious and for

Nucleus Sampling

HUSHARD IV:
Nor born of thyself; since of wife?

LUCENTIO:
Sweet in them such but any of my lord.

SLY:
Our messed bed, and love uncalleness.

DUKE VINCENTIO:
You, he's the severion and flies to you.

BRUTUS:
I pray, as my bed?

SLY:
Sir, you are no watch'd and leave hath band
As that did you do: for there we hear him,
Man, good Thought: and now, sir, therefore that and you are pate
Leaven: for would you live in years.

LUCENTIO:
To your reason man, I have terrant.

PRINCE ES:
And though it from an hungs or your brother.
That crown you in which, soundly and thy peech
And ne'er all them fears, his life,
To make you well.

Shepherd:
Never, sir, it I love us an open
When she was not the father of mine.
If this mean live to tell thee now; but I will not have stay:
Sir, this I do beseech you with more instruction, not,
Having much gross as was curst in that
past and move you, and I am not off at the sat to convey.

LUCENTIO:
Farewell, these servant of me,
A treasure doth true, and if I woo'd