Evaluating RNN, LSTM and Transformer Models Using Char and BPE Encoding on WikiText-2

Project in DD2424 Deep Learning in Data Science

Elias Stihl stihl@kth.se
Victor Stenmark vstenm@kth.se
Tore Nylén toreny@kth.se
Isak Reinholdsson isakre@kth.se

Contents

Abstract

In this study, we examine three different deep learning text generation architectures: RNN, LSTM, and Transformers, along with three different data encoding techniques: character-level, BPE, and BPE4. These techniques are fine-tuned by exploring various hyperparameter configurations to build the best possible model. Additionally, both nucleus sampling and temperature-based sampling techniques are tested to improve the quality of the generated text. After optimizing all our models, we found that the performance difference between the baseline RNN and the LSTMs was not substantial. Instead, the most significant improvement in performance was observed when switching from an LSTM model to a Transformer model. When testing the various encoding methods on the final Transformer, the BPE4 model performed the best, where the produced text to some extent can follow a set topic and has coherent sentences. However, all Transformers performed very well regardless of tokenization.

1 Introduction

Text generation stands as a fundamental task in natural language processing, driving advancements in various applications like chatbots, virtual assistants, machine translation, and more. In this study, we explore three distinct models for the task of text generation: Recurrent Neural Networks (RNNs), Long Short-Term Memory Networks (LSTMs), and Transformer architectures. Additionally, we investigate three different data encoding techniques: character-level encoding, Byte-Pair Encoding (BPE), and BPE4. To further enhance text generation quality, we test sampling techniques such as Nucleus Sampling and temperature-based sampling. The study includes comprehensive experiments involving fine-tuning the models with various hyperparameter setups and evaluating their performance using cross-entropy loss, perplexity, and spelling accuracy. Our main results show that transformers significantly outperformed RNNs and LSTMs. Based on qualitative assessments, the texts generated using BPE and BPE4 encoding appear to be superior to those produced with character-level encoding.

2 Background and related work

RNN. Recurrent Neural Networks (RNNs), first introduced in 1990 [11], are artificial neural networks specifically designed for processing sequential data. This capability makes RNNs particularly suitable for natural language processing (NLP). However, the RNN architecture comes with several disadvantages such as problems with vanishing/exploding gradients and difficulty in capturing long-range dependencies.

LSTM. LSTMs address the shortcomings of traditional RNNs [3]. LSTMs are composed of a series of memory cells, each with three gates called the input, forget, and output gate. The input gate controls the extent to which new information flows into the cell state. The forget gate decides what portion of the past information in the cell state should be forgotten. The output gate determines what part of the cell state is outputted to the next time step. The LSTM's ability to model and remember long-range dependencies makes them well-suited for tasks like language modeling, where understanding the context of a word relies heavily on the words that precede it. LSTM's suitability for NLP has also been shown empirically. For instance, Chen et al (2016) received an accuracy of 88.6% on the Stanford Natural Language Inference Dataset when using a bi-directional LSTM.

Transformer. The most recent of the three models is the Transformer architecture, introduced in the paper Attention is all you need [15]. The decoder-only architecture has been specifically impactful in the realm of large language models (LLMs) and text generation [13]. The decoder-only architecture is designed to generate text by predicting the next token in a sequence, given the previous token. This autoregressive approach is key to the text-generation capabilities of LLMs. While RNNs and LSTMs laid the groundwork for sequence modeling in NLP, transformer architectures have propelled the field to new heights. Transformers have been shown to be superior both in terms of performance and computational efficiency compared to LSTMs and RNNs [16].

Tokenization. Tokenization is the process of splitting the data into small bits that can be used in the network. Much work has gone into finding efficient tokenization techniques. Mikolov et al. [9] developed Word2Vec, a technique using simple neural network architectures to convert words into dense vector representations. A more recent tokenization technique is Byte-Pair encoding. First deployed in neural nets by Sennrich et al. [12], Byte-Pair encoding is a tokenization technique that constructs tokens based on the popularity of byte sequences in the data.

Sampling Techniques. When generating text, RNNs, LSTMs, and Transformers all construct probability distributions over their token vocabulary. Then, the sampling technique determines how to choose a token from this probability distribution. Temperature scaling, a common sampling technique introduced by Guo et al. [2], uses a scalar called temperature to skew the probability distribution. A higher temperature means that the probabilities for high-probability tokens go up further while a lower temperature increases the likelihood for low-probability tokens. Another common sampling technique is nucleus sampling developed by Holtzman et al. [4]. Nucleus sampling sums the most probable tokens until the sum of their probabilities is greater than a threshold. It then samples a token from the tokens included in the summation.

3 Data

We decided to train our models on the WikiText-2 dataset, a publically available data set created by wikitext[14]. WikiText-2 consists of a large number of good and featured articles from Wikipedia [8]. It is also not too big, with a total size of around 13.3 megabytes. We used the predefined training, validation, and test splits for training, validation, and test data. In these predefined splits, the training data had a size of around 10 million characters, whereas the validation and test sets had around 1 million characters each [8].

The dataset was downloaded and extracted into simple text files. Then we ran our preprocessing for the encodings described in section 4.2. In total, we had three different tokenizers run on these text files: a char encoder, a BPE encoder, and a BPE4 encoder. These various tokenizations were then used as input in our various models.

WikiText-2 is not only an established text dataset but also a benchmark that multiple state-of-the-art models have been tested on. One such is SparseGPT [1] where Frantar et. al. applied a pruning technique to reduce the number of parameters in a pre-trained model to reduce its computational requirements for inference. They managed to do so without a significant impact on perplexity. A famous model that has previously been evaluated on WikiText-2 is GPT-2 [10]. Due to time constraints, we did not manage to test any of our models on this benchmark.

4 Method

Our study aims to evaluate different approaches to text generation. Given the extensive scope of our examination, we needed a good structure that gave us meaningful scientific results, while not being unreasonable in terms of time and resources. To do this, we divided the work into four main parts: (1) finding and optimizing the models (RNN, LSTM, and transformer), (2) comparing different tokenizations, (3) improving the quality of the generated text using various sampling methods, and (4) testing and evaluating the best configurations that we find. For all parts, PyTorch was used to write the models for the convenience of AutoGrad, CUDA-support, and prewritten functions for loss functions and optimizers. However, no prewritten functions (like `torch.nn.LSTM`) were used for the models/tokenizers/evaluation methods that related to the results of the report unless otherwise specified.

4.1 Models

The first step was writing, optimizing, and comparing various models. For all of these tests, a simple character-level encoding was used. This was done with simple one-hot encoded vectors for every unique character in the corpus.

Baseline RNN. As a first baseline, we created a large RNN with char-encoding and tuned the hyperparameters based on our knowledge from previous coursework. This model was then used as a reference when evaluating various versions of the LSTM and Transformer models.

LSTM. We created a new model with the LSTM architecture. Since we had no prior work with this model, it needed to be optimized. We used two grid searches as we tuned the hyperparameters of our written LSTM. Due to limitations in time and computational resources, we decided to group hyperparameters when performing the grid search for optimal values. Testing all possible combinations of hyperparameters would result in excessively long training times. Therefore, we did one grid search with (1) batch size and (2) learning rate, and another grid search with (3) the size of the hidden layer and (4) the number of hidden layers.

Transformer. Having found a high-performing configuration for our LSTM, the next step was writing a transformer with char-encoding and optimizing it. As the complexity of the transformer architecture grows quite a bit from the aforementioned models, so does the number of hyperparameters to tune. We decided to keep a

fixed value for our batch size, learning rate, L2 regularization, and dropout. The values for these were based on suggested values from [5], [15] but while adhering to the limited compute power we had access to. The hyperparameters that we focused our tuning on were the (1) embedded layer size, (2) the number of heads, (3) the number of decoder block layers, and (4) the sequence length.

Adam optimizer For training the neural networks, we employed the Adam optimizer, a widely used optimization algorithm introduced in 2014 [7]. It utilizes adaptive learning rates and moving averages of gradients to achieve faster and more stable convergence compared to Stochastic Gradient Decent (SGD). Given the high dimensionality and variability in natural language data, Adam's dynamic adjustment of learning rates contributes to more effective training and better model generalization.

Diverse training sequences To ensure that the training was conducted on diverse text sequences, the batches in each training iteration were selected randomly from the text, rather than processing the text sequentially. This approach allowed for overlap between batches and was implemented to prevent the model from overfitting to specific text patterns and to enhance its ability to generalize across various text structures.

Performance Metrics To evaluate model performance, we employed cross-entropy loss. Additionally, we monitored perplexity, a standard metric in natural language processing (NLP). Lower perplexity values indicate better alignment between the model's predictions and the actual text. Furthermore, spelling accuracy was measured to provide insights into the linguistic quality of the generated text. The validation and perplexity losses were measured by taking 100 sample sequences from the validation data and then taking the mean value of each token. The test loss was created the same way but using the test data instead. Lastly, spelling accuracy was measured with the library Pyspellchecker on a synthesized text sample of 1000 tokens.

Note that when we introduced different tokenization techniques (described in 4.2), we could no longer only use validation loss and perplexity to compare the results. This is because these measurements are dependent on the number of tokens which at this point was different. Instead, the models were compared in terms of spelling accuracy and a subjective evaluation of the produced results.

4.2 Tokenization

Having examined the various models, the next step was to investigate more advanced tokenizations.

Word2Vec Aiming to improve from char encoding, we employed Word2Vec, a technique widely used for converting words into dense vector representations in continuous vector space. It captures semantic relationships between words by representing each word as a fixed-size vector based on its context in a large text corpus. The implementation relied on the Word2Vec module in the Gensim package. We split the text into sentences and fed those sentences into the Gensim Word2Vec algorithm to create vectors. We found however that for the algorithm to capture any semantic relationships between words required vector sizes of up to 2000. This combined with the network having 13000 output nodes (one for each distinct possible word) made training the network too inefficient considering our available resources. Thus, we discarded this method and chose to implement Byte-Pair Encoding instead.

Byte-Pair Encoding (BPE) Byte-Pair encoding (BPE) is a data compression technique for subword tokenization. It works by converting the corpus to a sequence of bytes and then iteratively merging the most frequent pairs of consecutive bytes in the corpus with a new token. This is done until the vocabulary hits a predefined size. An example of how BPE works can be seen below. Here (101, 32) is the most frequent pair meaning that it will be replaced by the new token 256 in the sequence to the right of the arrow.

```
[25, 101, 32, 90, 101, 32, 101, 32] -> [25, 256, 90, 256, 256]
```

We also used a technique we call BPE4 that ensures that byte merges are not performed between words. An example of this can be seen below. Here the first occurrence of (101, 32) is not replaced by 256 since 101 and 32 belong to different words (here represented as lists).

```
[[25, 101], [32, 90, 101, 32, 101, 32]] -> [[25, 101], [32, 90, 256, 256]]
```

For our implementation of BPE and BPE4, we relied on the work of Andrej Karpathy's minbpe [6]. Our BPE algorithm ran directly on the text data while the BPE4 algorithm ran after the text had been split into words using the GPT4_SPLIT_PATTERN [6]. For both encodings, we examined predefined vocabulary sizes of 512 and 1024.

4.3 Sampling Techniques

After identifying and optimizing the best models and encoders, we implemented temperature scaling and nucleus sampling to further enhance the generated text. As discussed in section 2, there is a tradeoff between coherence and creativity here, and too low values can result in texts that, while correct, are very repetitive. Different configurations were tested, and the best one was chosen based on our subjective assessments.

5 Experiments

5.1 Models

5.1.1 RNN and LSTM

A baseline RNN was trained for 25000 iterations. It used characters as tokens, sequences of length 32, a hidden layer size of 128, a batch size of 16, and a learning rate of 0.001. These values were based on previously derived results for the RNN model. In all experiments below, characters were sampled directly from the generated probability distribution without any temperature or nucleus sampling if not explicitly stated otherwise. Results from the baseline RNN showed a validation loss of 1.85, a validation perplexity of 6.38, and a spelling accuracy of 58.44%. We then wrote a LSTM and optimzed its hyperparameters in the steps described below.

Optimizing Learning Rate and Batch Size. First, the impact of increasing the batch size and learning rate was explored for a one-layer LSTM with 128 hidden nodes. The results in Table 1 show that increasing the batch sizes and learning rates increases the performance of the networks. However, we can assume that too large batch sizes and learning rates will hinder the training process.

Table 1: Comparison of LSTMs with different batch sizes and learning rates and the baseline RNN

model	learning rate	batch size	val loss	val perplexity	spelling accuracy
RNN	10^{-3}	16	1.85	6.38	58.44%
LSTM	10^{-2}	8	1.77	5.88	61.87%
LSTM	10^{-3}	8	1.92	6.83	49.32%
LSTM	10^{-4}	8	2.35	10.50	30.12%
LSTM	10^{-2}	16	1.72	5.60	65.82%
LSTM	10^{-3}	16	1.86	6.40	48.07%
LSTM	10^{-4}	16	2.29	9.90	29.58%

Optimizing Layers and Hidden Nodes. Next, it was investigated how the number of hidden layers and the size of the hidden layers affect the performance of an LSTM. All networks were trained for 25,000 iterations using the best found configuration in the previous test: a learning rate of 10^{-2}, batch size of 16, and a sequence length of 32. The results in Table 2 show that increasing the number of hidden nodes and layers decreases the loss and perplexity. The larger models also tend to have better spelling accuracy.

Table 2: Comparison of LSTMs with different hidden layer sizes and the baseline RNN

model	layers	hidden nodes	val loss	val perplexity	spelling accuracy
RNN	1	128	1.85	6.38	58.44%
LSTM	1	128	1.86	6.52	50.64%
LSTM	1	256	1.77	5.88	61.59%
LSTM	1	512	1.65	5.23	69.23%
LSTM	2	128	1.81	6.12	53.95%
LSTM	2	256	1.72	5.60	66.89%
LSTM	2	512	1.65	5.22	63.63%

Best setup Based on the validation loss and perplexity, we see that the best configuration for the LSTM had a a learning rate of 10^{-2}, batch size of 16, 512 hidden nodes and two hidden layers. This is expected as this was also the largest tested model. However, it is important to note that the equivalent one-layer LSTM actually demonstrated superior spelling accuracy and had very similar scores for the other metrics. In other words, we

do see that the biggest jump in performance seems to stem from more hidden nodes rather than more hidden layers. Upon subjectively evaluating the quality of the text generated by the LSTM and RNN models, we find that both produce text of quite poor quality, with readability being significantly compromised without any sampling technique (see Appendix A). However, the LSTM model appears to perform slightly better in handling long-range dependencies, as expected.

5.1.2 Transformer (Extension 1)

We wrote a decoder-only transformer and decided to keep a fixed value for our batch size, learning rate, L2 regularization, and dropout. Based on previous research (see 2), we decided on a batch size of 32, a learning rate of 10^{-3}, an L2 regularization of 10^{-3}, and a dropout rate of 0.1.

Optimizing embedded layer size. In the context of transformers, an embedding layer is used to convert input tokens (words, subwords, or characters) into dense vectors of fixed size. These vectors capture semantic information about the tokens and are suitable for further processing by the transformer layers. Since this size affects the execution speed we were careful to increase it too much. A total of three values were tested for the embedded size: 128, 256, and 512. After running our tests, see Table 3, we found that there was quite a large benefit for using an embedded size of 512.

Optimizing the head size and count During all tests, we kept the head count constant at 8. However, by changing the embedded size, we indirectly altered the head size. In total, we tried 3 values: 16, 32, and 64. Each of these heads keeps track of different aspects of the input data, allowing the model to learn various representations and capture more intricate patterns within the text.

Optimizing the number of decoder block layers Inspired by [15], we started with 6 layers but also tested configurations with 4 and 10 layers. Our results in Table 3 showed that there seems to be no correlation between the number of layers and the performance.

Optimizing the sequence length Finally, a very important parameter to optimize is the sequence length. This value corresponds to the number of tokens processed at once by the model. Longer sequences allow the model to capture more context, potentially improving performance on tasks requiring an understanding of longer dependencies. However, increasing the sequence length also increases computational complexity and memory usage. Our findings show that increasing this value greatly affects performance. see Table 3.

Table 3: Transformer tests with char level encoding

embed size	head size	seq length	layer count	val loss	val perplexity	spelling acc
128	16	64	6	1.38	3.97	84.46%
256	32	64	6	1.31	3.71	84.21%
512	64	64	6	1.29	3.64	85.91%
512	64	32	6	1.42	4.13	83.45%
512	64	128	4	1.23	3.41	84.84%
512	64	128	6	1.20	3.31	89.76%
512	64	128	10	1.25	3.50	82.98%

Best setup Looking at the results in Table 3, we see that having a sequence length of embed size of 512, head size of 64, sequence length of 128, and 6 layers resulted in the best performance on all three metrics. This will be the model that we use for future tests when examining different encodings. We also conclude that regardless of setup, the transformer model outperforms both the RNN and LSTM in all cases.

5.2 Tokenization (Extension 2)

Having written our optimized transformer, we were finally ready to test our more advanced tokenization methods. We ran experiments testing our best transformer with both BPE and BPE4 for vocabulary sizes of 512 and 1024. The results of this can be seen in table 4. As previously mentioned in section 4.1, comparing the loss and perplexity of models with different encoding sizes is not accurate. Thus in this section, only models with the same encoding sizes will be compared. Qualitative tests between models of different vocabulary sizes (as well as between char encoding and BPE/BPE4) will come in section 5.4. Looking at BPE and BPE4, we see that BPE4 with a vocabulary size of 1024 outperforms BPE with the same vocabulary size. It has better validation loss and validation perplexity. Looking at the BPE4 and BPE for vocabulary sizes of 512, we see that BPE4 has better validation loss and validation perplexity but worse spelling accuracy.

Table 4: Transformer tests of more advanced tokenization. Note that only the models with the same encode size can be compared in terms of loss and perplexity.

Encoding	Tokens	Embed Size	Head Count	Seq Length	Layer Count	Val Loss	Val Perp	Spelling Acc
BPE	512	512	8	128	6	2.43	11.30	89.74%
BPE4	512	512	8	128	6	2.40	11.04	86.79%
BPE	1024	512	8	128	6	3.17	23.67	86.05%
BPE4	1024	512	8	128	6	3.04	20.96	88.16%

Best setup Looking at the results in Table 4, it is harder than before to determine which model performs the best. The spelling accuracies are all pretty similar, and the other metrics can no longer be used to compare them as they have different tokens. However, we can conclude that BPE4 slightly outperforms BPE in both cases. Therefore our final tests in section 5.4, will compare the transformer with Char encoding, BPE4 encoding and 512 tokens and BPE4 encoding

5.3 Sampling Techniques

Before our final comparisons were made the effect of Temperature and Nucleus sampling was investigated. The synthesized text results in Appendix B show that nucleus sampling and temperature scaling are effective methods for increasing the quality of the generated text but that they come with a cost of less diversity. Simply by decreasing the temperature from 1 to 0.5, for the two-layer LSTM with 512 hidden nodes, the spelling accuracy increased from 63.63% to 97.13%. Using nucleus sampling with $p = 0.5$ yielded a spelling accuracy of 98.26%. However, while the spelling is much better, it comes at the cost of some variety. In the end, we decided to run our final tests with a nucleus sampling of $p = 0.75$ as we felt it struck a nice balance between creativity and coherence.

5.4 Final testing and qualitative comparison to other studies

Since it was unclear which of the three encodings for the transformer performed the best, we decided to run our final qualitative assessment on the three best candidates we found. This was the transformer model with the character-level encoding, BPE4 with 512 tokens, and BPE4 with 1024 tokens respectively. To test the models, they were tasked to continue the text: *"On 4 May 2008, Torres scored"*. We chose this prompt as it discusses football (without mentioning the topic by name), a general theme that we knew would be covered on Wikipedia.

The results, which can be found in Appendix C, are quite impressive, especially when compared to the results from our simpler RNN and LSTM models. For instance, we observe that all models are able to understand the theme of the initial prompt and continue discussing it. Even the character-level model maintains a good flow, despite the sequence length (context window) of 128 characters. For the setup with a larger number of tokens, the results are slightly better. However, when comparing the 512-token and 1024-token versions, the differences are minimal in terms of text quality.

Lastly, we compute the test losses for the best models. The transformer with the character-level encoding had a test loss of 1.22, the best LSTM had a test loss of 1.67, and the best RNN had a test loss of 1.86. Since these models use the same tokenization technique with the same vocabulary size, we can say that the Transformer is better than the LSTM and the RNN. However, since the models using BPE and BPE4 had other vocabulary sizes, comparing their test losses is not possible.

6 Conclusion

The results of our experiments demonstrate that transformers significantly surpass LSTMs and RNNs in text generation. Additionally, our findings indicate that tokenization and sampling techniques markedly enhance the quality of the generated text. Throughout this project, we have gained substantial insights into natural language processing and various architectures for managing sequential data. A potential future extension of this project would involve exploring data augmentation techniques for natural language processing, such as synonym replacement and random insertion, to evaluate the performance enhancements of the Transformer model when trained on a limited dataset.

7 Acknowledgments

- Andrej Kaparthy's tutorials and public repositories served as both a source of inspiration and as a guide to do the two extensions.
- ChatGPT has been used to improve the writing of this report.

References

[1] Elias Frantar and Dan Alistarh. Sparsegpt: Massive language models can be accurately pruned in one-shot, 2023.

[2] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *International conference on machine learning*, pages 1321–1330. PMLR, 2017.

[3] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[4] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. *arXiv preprint arXiv:1904.09751*, 2019.

[5] Andrej Karpathy. nanogpt-lecture. `https://github.com/karpathy/ng-video-lecture`, 2023.

[6] Andrej Karpathy. minbpe. `https://github.com/karpathy/minbpe`, 2024.

[7] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[8] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

[9] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[10] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

[11] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation, parallel distributed processing, explorations in the microstructure of cognition, ed. de rumelhart and j. mcclelland. vol. 1. 1986. *Biometrika*, 71:599–607, 1986.

[12] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*, 2015.

[13] Sathya Krishnan Suresh et al. Towards smallers, faster decoder-only transformers: Architectural variants and their implications. *arXiv preprint arXiv:2404.14462*, 2024.

[14] @patrickvonplaten @mariamabarham @thomwolf, @lewtun. Wikitext · datasets at hugging face.

[15] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[16] Chenguang Wang, Mu Li, and Alexander J Smola. Language models with transformers. *arXiv preprint arXiv:1904.09408*, 2019.

Appendix A

Generated Text From Our Best Performing RNN With No Sampling Techniques

= = ID shared the brivery wele with ooted they was use denurams projectnys an inghement and nota is a remores , as a 10 in Anobic and Hartle 's eventure Chelems : publid batherward at has edent . He cripugn was sy the brignh . The swork that the ang t @ 5 gmser 's sway to Away red autzan aqwass to ab the succession Talls Wollowardäin (the suff as " r

Generated Text From Our Best Performing LSTM With No Sampling Techniques

Note that some of the unusual occurances (like spaces before periods and commas as well as the @-@ pattern) stem from the training data containing these adjustments to the text.

```
ed on 12 Jro Statey). Barriloger by the yeely regional describes a war began secule
about 13 designensy added from " " 6 - yard was conserved to played folk had nelds
followed as a wember.
 In 1850) after the Viet that he restored intery tyal renderancial.
== Burch Lanis Trophes I south number 1986 captures Redurand, Festival Certain==
 Zealon Ailáng 's guest champion==
 The convection=
```

Appendix B

Generated Text From Our Best Performing LSTM With No Sampling Techniques

Note that some of the unusual occurances (like spaces before periods and commas as well as the @-@ pattern) stem from the training data containing these adjustments to the text.

```
ed on 12 Jro Statey). Barriloger by the yeely regional describes a war began secule
about 13 designensy added from " " 6 - yard was conserved to played folk had nelds
followed as a wember.
 In 1850) after the Viet that he restored intery tyal renderancial.
== Burch Lanis Trophes I south number 1986 captures Redurand, Festival Certain==
 Zealon Ailáng 's guest champion==
 The convection=
 Onein how her scoretate called success poers "after the battle of him Bridgy,
Rupan Hatchige fedure Visis natural seez is persectly played sensive flem ;
textsprease the following steam symmonistic and features to number of missue
counteration is each two moved, or a stearity===
== Somersprically succeeing
= Best Galloy Kuitar Federal Oustial Somerselve aircraft marrisby in 1994, but as
made of which . Killid, Naturrued Bord Nyrzan bogies gang her refused in the CWL).
The election==
 They were questive in the gate the Ken of 1, 2003 Deventure, SIreiss
```

Generated Text From Our Best Performing LSTM with Temperature Sampling $T = 0.5$

Note that some of the unusual occurances (like spaces before periods and commas as well as the @-@ pattern) stem from the training data containing these adjustments to the text.

```
ed the fact. The first playing for the series in the promotion of the Trand Steven
's death, in the first of the second series, a perform carried to the institutions.
The first recently success of the military 1940s, when it was recorded and career,
in the start series, and the trinsity of the first - recent player produced the
late 19 @.@ 1 feet ( 1 @,@ 130 m), the star - attack of the company==
 Leading==
 The family==
 Media==
 The same signed by the first and in the song. The top of the mainly for the time
at the field single to information and the Series, where the consideration of
the early in the second company or struggled and the first playing it is not the
competition and several present, the way play of the album of the first adopting of
the first of the set in the experience of the second home to the first save series
and returned to the developing other state, the main most of the institution of the
series of the first man, the first publicati
```

Generated Text From Our Best Performing LSTM with Nucleus Sampling $p = 0.5$

Note that some of the unusual occurances (like spaces before periods and commas as well as the @-@ pattern) stem from the training data containing these adjustments to the text.

```
ed by Barliament 's late 1888 and 1958. The marker==
 The second of a fact - better fire of the company, and some as the first control
of the first surround.
=== The large program is survey as the character in 1982, the song==
 The time, but it was a similar and the reader as the man of the propert, with
the program was controlled in the reach the first produced a new personal for a
track and film, but it as the manager of his restoration of a first control the
considerable for the considerable studio to the first producer in 1977.
== Section==
```

The cast set reached a part of the received from a control of the propert and the first time, and stream in the second of his first success of the series of a result of the record and the basis and for his conservation==

== Chance==

The Story State Bells), the main the first season==

In 2003, the producer==

The fort==

In 1977, the transfer of his time in the term state, a temple served from t

Appendix C

Generated Text FromBest Char Transformer with Nucleus Sampling $p = 0.75$

Prompt: *On 4 May 2008, Torres scored*

Note that some of the unusual occurances (like spaces before periods and commas as well as the @-@ pattern) stem from the training data containing these adjustments to the text.

```
On 4 May 2008 , Torres scored the first goal for the title with a 1 - 1 home defeat
, which he scored 14 goals in 25 games . Alice in Chains played his 28 - 10 lead
season , Liverpool scored the play @-@ offs in the 1969 - 98 season and 20 - 40 ,
but the third @-@ year @-@ old Final , for the match in the first half of the year .
The second season was completed in 1988 by Football League Two , and held in 1989 ,
the NCAA also formed a new presidential artist to defend the NCAA .
 = = = Season stage = = =
 After the end of
```

Generated Text From Our Best BPE4 512 Transformer with Nucleus Sampling $p = 0.75$

Prompt: *On 4 May 2008, Torres scored*

Note that some of the unusual occurances (like spaces before periods and commas as well as the @-@ pattern) stem from the training data containing these adjustments to the text.

```
On 4 May 2008 , Torres scored two goals in 48 appearances in the 2008 - 09 season
, against Manchester United on 24 January 2009 , and the 2009 - 09 season was
reported as a signed spell for the first time . The following season , Cresswell
pitched seventh in the 2008 - 09 season , against the Class AAA Minnesota Trophy
, which recorded a 10 - 9 record , a 3 @.@ 59 ERA and 145 innings pitched in 105
games . The club recorded 125 games in 1186 - 44 with a 2 @.@ 38 ERA and 114 over
117 strikeouts , 151 strikeouts . The history of the 1996 - 97 season was the first
time that Fowler would celebrate at a season in 1995 , and reported that Fowler 's
record of 155 @,@ 000 games in the 1997 - 97 season .
 The two 1996 seasons lasted , 27 years after the murder of Fowler 's return to
fifth season , before dropping out of £ 1 @,@ 070 . Subsequently , after five years
of meetings in the club 's final season , Fowler finished as 13th minutes in the P
```

Generated Text From Our Best BPE4 1024 Transformer with Nucleus Sampling, $p = 0.75$

Prompt: *On 4 May 2008, Torres scored*

Note that some of the unusual occurances (like spaces before periods and commas as well as the @-@ pattern) stem from the training data containing these adjustments to the text.

```
On 4 May 2008 , Torres scored his first goal for England in the 2000 FIFA U @-@
17 World Cup in the 1 - 0 win over Sweden . The two clubs scored eight goals in
the 2001 FIFA Confederations Cup , but had finished in seventh place in the FIFA
Confederations Cup squad .
 At the age of 24 , Torres scored the winner with one goal in a 1 - 0 win over
Torres 's home ground . He won his first FIFA Confederations Cup the following
season in a 1 - 0 victory over Toronto . He won his first FA Cup quarter @-@ final
on 18 May 2001 in the FIFA Confederations Cup .
 Fowler scored 16 appearances for Liverpool in the final against South Africa in
the 2001 Confederations Cup on 29 March 2001 , and scored a 2 - 0 win over Djokovic
in the 0 - 0 draw away at Anfield . Torres scored the winner with 11 goals in a 2
- 0 win over Liverpool , scoring 34 goals in 42 games to finish his first season as
a substitute in the 2001 FIFA Confederations Cup , on 29 March 2001 . On 27 March
2006 ,
```