# Victorian-style (Holmesian) Passage Creation

Jesus Tello

June 2025

## Abstract

In this project, we explore character-level recurrent neural network (char-RNN) modeling of Arthur Conan Doyle's Sherlock Holmes complete story using the char-rnn-tensorflow implementation by Sherjil Ozair[1]. We first create a baseline Long Short-Term Memory (LSTM) model (2 layers, 128 units, sequence length 50) trained on the full Sherlock Holmes corpus. Observing suboptimal coherence in generated samples, we continually tune hyperparameters hidden size, sequence length, dropout, learning rate, decay rate, and number of epochs to improve text quality and convergence speed. We then compare LSTM against alternative RNN cells (Neural Architecture Search, NAS) under matched configurations, conducting a final learning-rate sweep on the best-performing variant. The results show that the NAS cell achieves lower training loss and generates more coherent, context-aware Victorian-style text than the baseline LSTM. This report details our dataset, methodology, experiment workflow, hyperparameter choices, and analysis of generated samples and training loss curves.

## Introduction

The goal of this project is to show that a simple character-level recurrent neural network (char-RNN) can learn the style of Victorian prose from Arthur Conan Doyle's Sherlock Holmes stories and generate new passages on demand. By processing text one character at a time and maintaining an evolving hidden state, the model captures syntax, punctuation, and narrative context. A successful char-RNN opens up possibilities for creative writing tools that can suggest or extend text in an arbitrarily chosen or given style.

### Background on RNNs

RNNs work by carrying a hidden state through a sequence of inputs. At each character, the network takes the current character embedding plus the previous hidden state, applies a learned update, and produces a new hidden state (and, when training, a prediction). LSTM and GRU cells add simple gating mechanisms - learned switches that control what to remember or forget - to help capture patterns over longer spans. NAS cells go further by automatically searching for the best gate structure while training on the data. [1]

## Methodology

### Dataset

Given that the instructions were provided to do so on the GitHub repository [2], we opted to download the complete, ASCII texts of all Sherlock Holmes novels and short stories from the public domain repository sherlock-holm.es/ascii/. The raw text (~3 MB) was used directly - no tokenization, title removal, or preprocessing on this raw text data was performed. Perhaps removing the gaps of text between chapters or short stories, and removing the title names would have improved the results of the model sample text generation, however it did not seem necessary. There is such an abundance of text training data, removing the titles and relatively small amount of newline text gaps would not have made much of a difference in training. As for reading the data, a Python TextLoader splits the text into sequences of fixed length ( `--seq_length` ) and batches ( `--batch_size` ).

### RNN Implementation

The implementation of the character-level recurrent neural network was given from Sherjil Ozair's public GitHub repository, utilizing a two-script structure in the following format:

- Argument parsing: At the top of train.py, an ArgumentParser defines flags for data paths ( `--data_dir` , `--save_dir` , `--log_dir` ), checkpointing ( `--save_every` , `--init_from` ), model choice ( `--model` ), architecture ( `--rnn_size` , `--num_layers` ), and optimization ( `--seq_length` , `--batch_size` , `--num_epochs` , `--learning_rate` , `--decay_rate` , `--grad_clip` , `--input_keep_prob` , `--output_keep_prob` ). The default values of these parameters can be overridden, and their defaults are provided in train.py.
- train(args) function:
  - Data loading: Instantiates TextLoader( `args.data_dir` , `args.batch_size` , `args.seq_length` ), which reads input.txt, constructs a vocabulary, and creates iterable batches of character indices.
- Model creation:
  - The Model class in `model.py` constructs a stack of RNN cells (LSTM/GRU/NAS) wrapped with dropout as specified. It embeds input character indices into dense vectors, unrolls them through the multi-layer RNN, and projects outputs back to the character vocabulary via a linear (softmax) layer.
- Training loop:
  - Furthermore, train(args) instantiates TensorBoard summaries and a `Saver` for checkpoints, as well as initializes or restores model weights. For each epoch, it decays the learning rate, resets the data pointer, and loops over batches
    - Also fetches input/target pairs, runs a single `sess.run` to compute loss, update state, and apply gradients with clipping
    - As well as logs loss summaries, prints progress, and saves checkpoints at given intervals.
- Sampling:
  - The `Model.sample` method initializes a single-step state, "primes" the RNN with a short prompt, then iteratively feeds back sampled or greedy predictions to generate new characters one at a time.

The design of this implementation makes clean and separate input/outputs, model definitions for training, and easy inference. Swapping cell types (model types) is straightforward, and tuning hyperparameters is also extremely easy.

## Baseline Model and Hyperparameters Descriptions

The baseline LSTM model parameters were suggested, and given by Ozair in the GitHub repository[2].

- Hidden units ( `--rnn_size` ): 128
- Layers ( `--num_layers` ): 2
- Sequence length ( `--seq_length` ): 50
- Batch size ( `--batch_size` ): 50
- Learning rate ( `--learning_rate` ): 0.002\
- Decay rate ( `--decay_rate` ): 0.97
- Input Keep Probability ( `NULL` ): Left default at 1.0
- Output Keep Probability ( `NULL` ): Left default at 1.0
- Gradient clipping ( `--grad_clip` ): 5.0
- Epochs ( `--num_epochs` ): 20 Below are brief descriptions for what each parameter entails:
- Hidden units ( `--rnn_size` ): Size of each RNN cell's state vector. Larger values increase capacity but require more computation and data.
- Layers ( `--num_layers` ): Number of RNN layers stacked. More layers can capture hierarchical patterns but may overfit or be harder to train.
- Sequence length ( `--seq_length` ): Number of characters the network sees before truncated backpropagation. Controls context window size vs. memory requirements.
- Batch size ( `--batch_size` ): Number of sequences processed in parallel. Impacts training stability and GPU utilization.
- Learning rate ( `--learning_rate` ): Initial step size and its per-epoch decay factor. Higher rates speed learning but risk instability.
- Decay rate ( `--decay_rate` ): Fraction of units retained during training to regularize and prevent overfitting.
- Input Keep Probability ( `--input_keep_prob` ) : The probability of keeping weights in the hidden layer.
- Output Keep Probability ( `--output_keep_prob` ): The probability of keeping weights in the input layer.
- Gradient clipping ( `--grad_clip` ): Caps gradient norm to avoid exploding gradients in deep RNNs.
- Epochs ( `--num_epochs` ): Complete passes over data. More epochs can improve fit but may overfit.

Given the descriptions of these parameters and their meaning, the only parameter I selectively chose prior to knowing much about the input data, model creation and text sample generation, was sequence length. As a general rule of thumb, most sentences in writing, speech, conversations, etc. are about 50 characters long. As such, this value was the only one personally selected by me for the initial baseline.

## Hyperparameter Tuning

We ended up branching off the baseline model in multiple ways after obtaining initial results. Below are the various configurations tested.

| Run Name | -- num_layers | -- rnn_size | -- seq_length | -- batch_size | -- num_epochs | -- learning_rate | -- decay_rate | -- input_keep_prob | -- output_keep_prob | -- grad_clip |
|---|---|---|---|---|---|---|---|---|---|---|
| updated_run_lstm | 2 | **256** | **64** | 50 | **10** | **0.001** | **0.98** | **0.8** | **0.8** | 5.0 |
| updated_run_2_lstm | **3** | 256 | **100** | **80** | **5** | **0.005** | 0.98 | **0.6** | **0.6** | 5.0 |
| lr_0001_lstm | 3 | 256 | **64** | **50** | **8** | **0.001** | **0.97** | **0.8** | **0.8** | 5.0 |
| lr_0002_lstm | 3 | 256 | 64 | 50 | 8 | **0.002** | 0.97 | 0.8 | 0.8 | 5.0 |
| lr_0003_lstm | 3 | 256 | 64 | 50 | 8 | **0.003** | 0.97 | 0.8 | 0.8 | 5.0 |

**Model Comparison**

After hyperparameter-tuning the LSTM model, we reran all the previously done hyperparameters on a NAS model, and opted to compare the train-loss and sample text generation between the two models. Below is a table containing all the hyperparameters once again, performed on the NAS model.

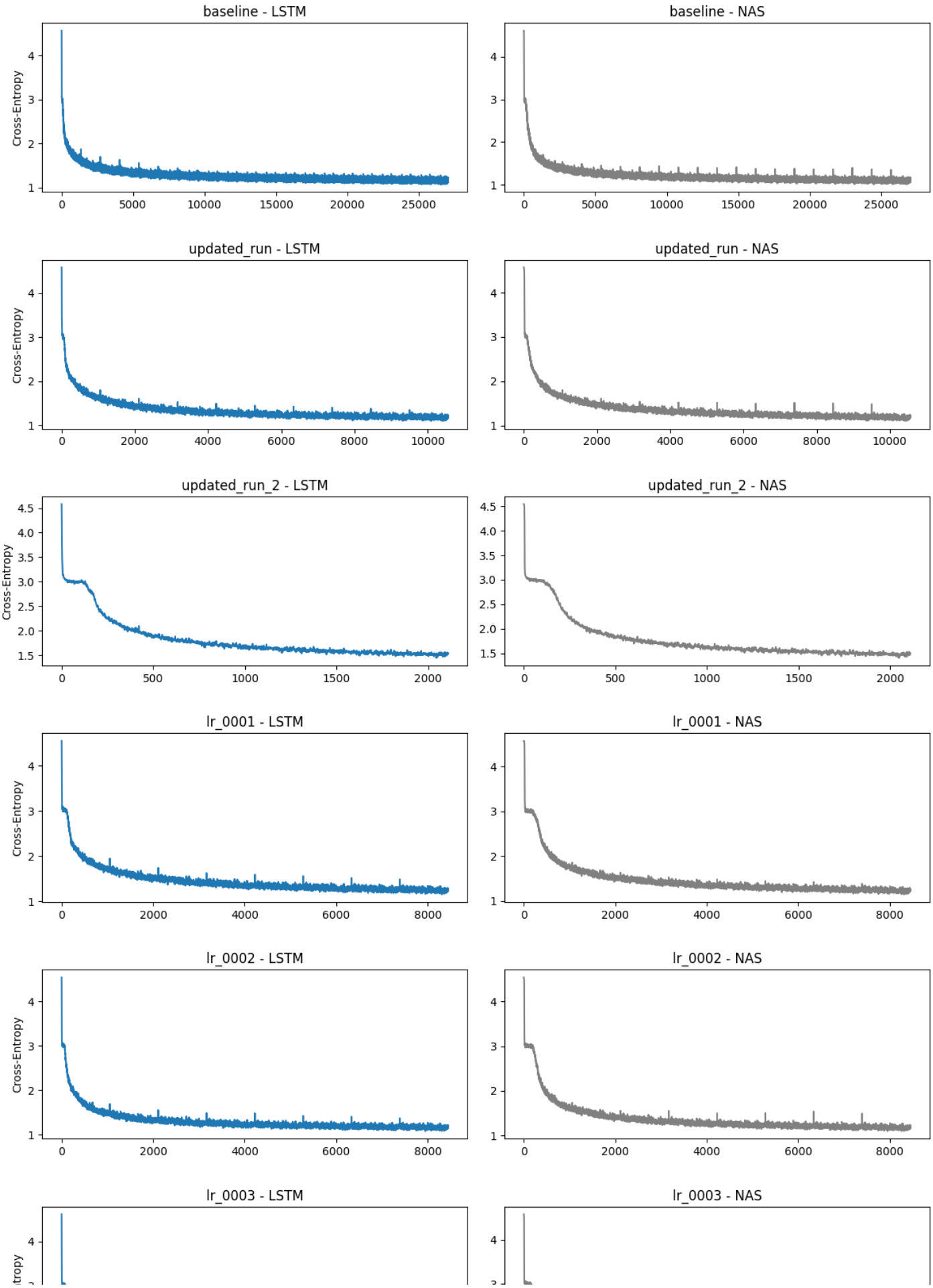
| Run Name | --num_layers | --rnn_size | --seq_length | --batch_size | --num_epochs | --learning_rate | --decay_rate | --input_keep_prob | --output_keep_prob | --grad_clip |
|---|---|---|---|---|---|---|---|---|---|---|
| baseline_nas | 2 | 128 | 50 | 50 | 20 | 0.002 | 0.97 | 1.0 | 1.0 | 5.0 |
| updated_run_nas | 2 | **256** | **64** | 50 | **10** | **0.001** | **0.98** | **0.8** | **0.8** | 5.0 |
| updated_run_2_nas | **3** | 256 | **100** | **80** | **5** | **0.005** | 0.98 | **0.6** | **0.6** | 5.0 |
| lr_0001_nas | 3 | 256 | **64** | **50** | **8** | **0.001** | **0.97** | **0.8** | **0.8** | 5.0 |
| lr_0002_nas | 3 | 256 | 64 | 50 | 8 | **0.002** | 0.97 | 0.8 | 0.8 | 5.0 |
| lr_0003_nas | 3 | 256 | 64 | 50 | 8 | **0.003** | 0.97 | 0.8 | 0.8 | 5.0 |

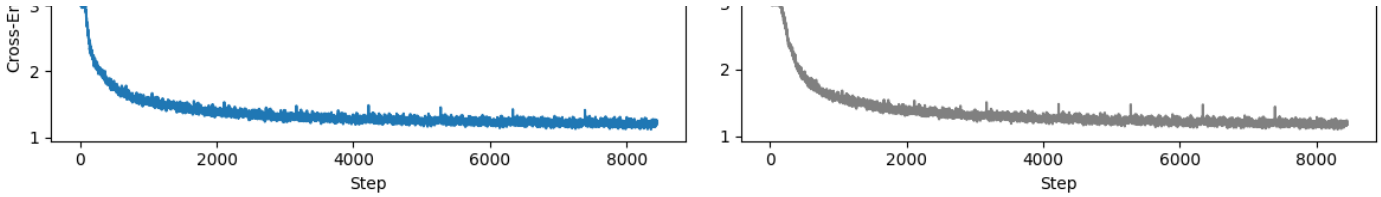
**Model Evaluation**

Each and every model was quantitatively and qualitatively evaluted, by extracted `train_loss` from TensorBoard event files to plot convergence curves, and by generating 300-character samples with a fixed prompt, initially primed by "I ," and later primed by default "The ." The choice for initial prime "I " was to see if the char-RNN could capture the unique character of Holmes himself - to view if any conversations or dialogue between characters could be captured. Some models were able to do so, and others weren't. The coherence, sentence structure, and context of each generated sample from each model was evaluated and led to the various hyper-parameter changes.

## Results

The training loss curves of all models, displayed below:


baseline - LSTM
baseline - NAS
updated_run - LSTM
updated_run - NAS
updated_run_2 - LSTM
updated_run_2 - NAS
lr_0001 - LSTM
lr_0001 - NAS
lr_0002 - LSTM
lr_0002 - NAS
lr_0003 - LSTM
lr_0003 - NAS
Cross-Entropy




Below are the final train loss values for each of the models:

Final Train Loss

| Model | Variant | LSTM | NAS |
|---|---|---|---|
| 1 | baseline | 1.155074 | 1.099551 |
| 2 | lr_0001 | 1.258781 | 1.2311153 |
| 3 | lr_0002 | 1.174434 | 1.164366 |
| 4 | lr_0003 | 1.218899 | 1.181866 |
| 5 | updated_run | 1.188411 | 1.188163 |
| 6 | updated_run_2 | 1.546502 | 1.502885 |

Below are the sample text generated from both the LSTM and NAS models:

Sample Text Comparison

| model | variant | lstm_sample | nas_sample |
|---|---|---|---|
| 1 | baseline | The when a consults, follows.<br><br>"Did they trouble--more to you to give a quickle. His rust the<br>part of "I should do your world: 'He'll not? I will reach of<br>that it is reach coloured down to this eye to my father incush<br>which I have had taken--Charpens to long-beaous confess<br>of hi | The was the<br>deduction to have none. That is that I had have hold rather than I cut to<br>read that.<br><br>"I belp up me conceals appear of poor opportunity. She become larger man, we intrusibly<br>were one should produced himself, "leaving the facts."<br><br>"I have didn't you were stood behind |
| 2 | updated_run | The Dure, and lyon<br>was no word dropped upon; The person had very brink to a past.<br><br>It was a friend it has my moulder appeared to know in cause, and the langers<br>definitely with home of its visitor. I had professed notice,<br>I should get open upon the turn. As the dent of him, has some t | The we shall<br>call the floor which came out, and that's carriage. And so you use it<br>with a rasp, between the bourdering garden. Lestrade?"<br><br>"He has too veaying my throat of his knee in a table, however, I have<br>looked intending to you to the else aloud some of my life. It is in the |
| 3 | updated_run_2 | The had gever matter of Minst mrown his pict-repare-to dad dright Caich<br>cary<br>very few thiths of up to what our trifial."<br><br>Before is it to have seen by<br>none it, but had work."<br><br>"And to he give, this horant dacket of the way." I conclise ot an window,<br>the lad.<br><br>"How!" | The Hals. That is now protess, there was not any<br>words. When I am seen then we shall feet me further. "It was pure him by from Deal his tapian women<br>mousing off all guess miss become."<br><br>"I must will both malast. We know the rooms was recy?"<br><br>"Well, as So, what dishoised in the looks |
| 4 | lr_0001 | The thing<br>he was Daken Street. I dure made a case from him., a black and a back<br>Room on the Lair, the hill to gray--I saw no carefully our lady appeans<br>make? We will, he will pite with the letter. Let kist Last then, heaven's<br>prisent, and the ampering gore-bay widel in the light upon | The buldy. It is the<br>unfortunate his infidence. It was more never<br>after Berdy, it are, the two pluttering than who paused all to<br>not in sequent wife when I get him to deep in my trouble. He was spoke<br>fits it. Certainly; but was the man with in suiting satisficate months.<br><br>"There |
| 5 | lr_0002 | The Stable. She<br>could go out in Cardias. I've glanced to see West's eyes as meet, you'll<br>not pursuit them by seeing you a satisfactive demente of<br>none and wanted cause in a hurard, and he should person along the<br>'emanly. These as every oldime still chamber had needed to me find my | The again<br>cizarys of the one were began to a morning after a street evident<br>as pointed him on.<br>This gun of bright among it as to a biolent outing lantern of<br>the step-parain, which over for us for Ga, and the omized of the<br>deed by good-noted in his face which is interviewing rou |
| 6 | lr_0003 | The young assuther<br>at the son's murderer. Thank go informated my task's bottom," said<br>the huge, geting the other victim, now, but does ever! I<br>could expect. Of course, with the morning, only opening her<br>all stair of the strange man whose rivent ended by the house of light. The coolle | The may<br>tel not kitchen. It is courted, but from Holmes," said he, criminals, and<br>a place and aslarental, clear that he turned and secured, which<br>seemed out of the natural very line.<br><br>"And I have have to have had our dear?"<br><br>Why drew him go hards at London, or his mind which wal |

## Discussion

The training curves (Figure above) and final loss table reveal a consistent pattern: across all hyperparameter settings, the NAS cell attains a lower cross-entropy than its LSTM counterpart. For the baseline configuration, NAS converges to ~1.10 versus LSTM's ~1.16, indicating a smoother loss landscape and more efficient gradient propagation. In our tuned experiments, the updated_run variants demonstrate nearly identical final losses (~1.188) for both LSTM and NAS, suggesting that certain hyperparameter choices (e.g. increased hidden size, dropout) can close the gap between architectures. However, under the more aggressive settings of updated_run_2, NAS again outperforms LSTM (1.50 vs. 1.55), consistent with its learned gating structure that better handles longer sequences (seq_length=100) and higher dropout rates (0.6).

In the final three models for each model type respectively, NAS also obtains a smaller final training loss value, at lr=0.001 (1.23 vs. 1.26), lr=0.002 (1.16 vs. 1.17), and lr=0.003 (1.18 vs. 1.22). However, obtaining a lower training loss doesn't necessarily guarantee that the generated sample text will be readable.

The most striking observation is that simply swapping from an LSTM cell to NAS under identical settings greatly reduces the frequency of nonsensical "words" and encourages more valid English patterns - even at modest capacity (2 layers, 128 units, sequence length 50). Increasing the hidden-state size to 256 and modestly extending context (sequence length 64) yields real character names and partial sentence structure, but it is only when we stack three layers, stretch the context window to 100 characters, and apply moderate dropout (keep probability 0.6) that both LTSM and NAS produce sentence fragments with proper punctuation. However, the LSTM variant for model 3, though has relatively accuracte sentence structure and syntax, has a bunch of nonsensical made-up words. The NAS variant of this model was actually able to, under these parameter conditions, strings together somewhat logical clauses ("That is now protest, there was not any words. When I am seen then we shall feet me further."), and frames short dialogues, whereas the LSTM counterpart still stumbles over invented tokens and fractured syntax. Learning-rate sweeps around 0.001–0.002 further refine the balance between stability and novelty: too low a rate produces overly safe, repetitive text, while too high a rate reintroduces spelling errors. Taken together, the best-performing sample - "`updated_run_2_nas`" - demonstrates that deeper stacking, longer context, moderate dropout, and the NAS cell's automated gating synergy are key to generating the most Holmesian-style prose with both creativity and readability. Additionally, it proves that achieving the lowest final training loss isn't everything - as `updated_run_2_nas` boasted the second highest final training loss value.

## Conclusion

Through careful hyperparameter tuning and by comparing LSTM and NAS cells, we showed that deeper networks with moderate dropout and longer context windows produce the most coherent Sherlock-style text. The best model we obtained, a 3-layer, 256-unit NAS rnn with a sequence length of 100 and keep-probabilities of 0.6, even had one of the higher final training loss values, but still created the most syntax-correct, somewhat context-logical mimicry of Sherlock Holmes text. Though, we were unable to completely and accurately produce perfect Holmesian sample sentences, there is definitely room for improvement. Initially though to not cause problems, working with the raw, unprocessed text data, without removing titles and the various newline characters caused a lot of them to appear in our sample text generations. Future work should clean the text. Experimenting with even more hyperparamter tuning, paying attention to sequence length, inner and outer keep probability, rnn_size, and num_layers might prove to improve text generation.

## References

References

1. Karpathy, A. (2015). The Unreasonable Effectiveness of Recurrent Neural Networks. The unreasonable effectiveness of recurrent neural networks. http://karpathy.github.io/2015/05/21/rnn-effectiveness/

2. Ozair, S. (2016). char-rnn-tensorflow. GitHub. https://github.com/sherjilozair/char-rnn-tensorflow

3. The Complete Sherlock Holmes. The complete Sherlock Holmes. (n.d.). https://sherlock-holm.es/stories/plain-text/cnus.tx