

# **Sherlock Holmes Character Level RNN Model**

## Table of Contents

Sherlock Holmes Character Level RNN Model .....	1
Problem 2.....	2
1. Overview.....	2
2. Data Loading & Processing .....	4
3. Develop the Character Generation Model .....	8
Model 1 LSTM RMSprop Baseline Model with 100 sequence_length .....	8
Model 2 LSTM RMSprop with 2 LSTM layers 100 sequence_length .....	12
Model 3 LSTM Adam Baseline Model 100 sequence_length.....	14
Model 4 LSTM Adam 2 layers, increase learning rate 100 sequence_length .....	16
Model 5 GRU RMSprop Base Model 100 sequence_length .....	18
Model 6 GRU RMSprop 2 layers with dropout 100 sequence_length .....	20
Model 7 GRU Adam 100 sequence_length.....	22
Model 8 LSTM RMSprop Change learning rate, add dropout 200 sequence_length .....	24
Model 11 GRU RMSprop Base Sequence length 200 .....	28
Model 13 copy of Model 9 but with 10 epochs .....	30
Model 14 LSTM Adam Sequence length 150 .....	32
4. Use the Models to Make Predictions and Recommending the Best Model .....	34
Model 13 (2/3) 1 label predicted correctly .....	38
Model 14 (1/3) 1 label predicted correctly .....	39
5. Summary .....	40

## **Problem 2**

### **1. Overview**

#### **The Problem**

The primary objective of this assignment is to develop and identify the optimal neural network model for predicting and generating the subsequent character following a given sequence of words or a single word. In approaching this problem, I have deeply considered the nuances of natural language generation and the challenges inherent in modeling such a complex system with computational tools.

In essence, this project simulates aspects of how Recurrent Neural Networks (RNNs) might generate English text which is a process that can, in some ways, be likened to the manner in which toddlers experiment with and create new words. However, I recognise that, unlike the rudimentary approximations of language generation by RNNs, adult humans excel in constructing coherent words, phrases, and sentences. Humans also naturally perceive and convey tone, emotion, and style that remain challenging for current software implementations.

This problem is framed as a multi-class, single-label classification task, where the model is constrained to predicting one character at a time from a set of 35 possible characters. In designing this model, I deliberated on the implications of generating text character-by-character versus word-level predictions, noting that the sequential nature of the task requires careful handling of long-term dependencies and error propagation. Furthermore, the challenge of aligning computational efficiency with the intricacies of natural language underscores the ongoing research in computational linguistics and artificial intelligence.

Overall, my approach involved a thoughtful exploration of various RNN architectures, such as LSTMs and GRUs, and a rigorous process of hyperparameter tuning and cross-validation to ensure that the model could effectively capture the underlying patterns of the English language. This assignment not only deepened my understanding of neural network design but also highlighted the broader limitations and potentials of machine-generated language.

#### **The Objective**

In this project, I aimed to explore the capacity of Recurrent Neural Networks (RNNs) to internalise and replicate the inherent patterns found in natural language. By examining how alphabets and characters are structured to form words, phrases, and ultimately sentences, the model is designed to predict the probability distribution of the next character in a sequence. An interesting aspect of this approach is the use of a temperature parameter, which allows the model to toggle between more deterministic and probabilistic outputs. This means that even with the same input, the model might generate different results depending on the temperature setting, a feature that adds a layer of complexity and realism to the text generation process.

The primary objective of this assignment is to build a character-level language model that can generate semi-coherent or somewhat coherent English sentences from scratch. Throughout this process, I engaged in a thoughtful exploration of the challenges and opportunities presented by this task. In my analysis, the optimal model should meet several key criteria:

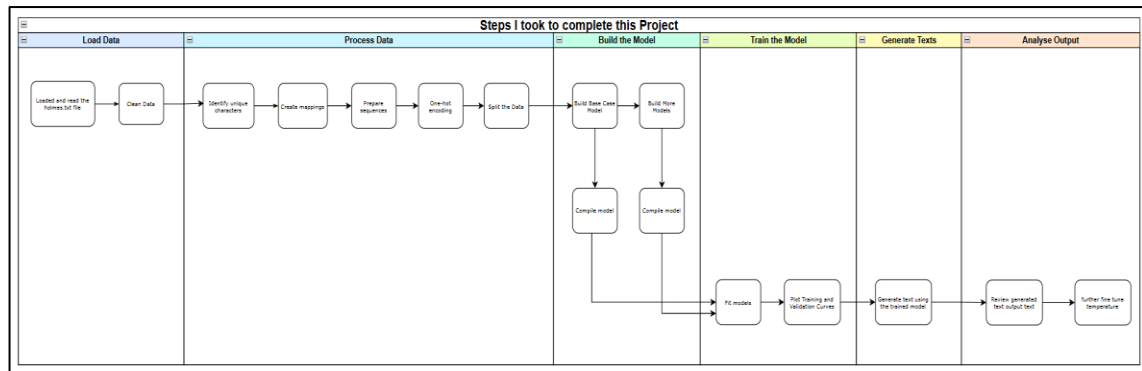
- **High Final Epoch Validation Accuracy:** This indicates that the model has learned the underlying patterns of the language well and can generalise to unseen data.
- **Quality of Generated Text at Different Temperature Settings:** When visually assessing the outputs, particularly at temperatures of 0.5 and 1.0, the model should produce a high percentage of recognisable English words, contributing to the overall coherence of the sentence.
- **Appropriate Punctuation Placement:** The model should correctly insert punctuation to enhance readability and mimic natural writing.
- **Realistic Word Distribution and Frequency:** The generated text should reflect the natural distribution of words in English, avoiding excessive repetition and ensuring variety.
- **Label Generation for New Inputs:** The model should be capable of generating a valid character label that could potentially be used to construct a new English word when presented with fresh input data.

To meet these objectives, I systematically experimented with different RNN architectures, such as LSTMs and GRUs, and engaged in extensive hyperparameter tuning. This iterative process deepened my understanding of how various model configurations affect performance, particularly in capturing long-term dependencies in text data.

Additionally, I incorporated both qualitative and quantitative evaluation methods. Qualitative assessments involved manually reviewing the generated sentences for grammatical coherence and natural language flow, while quantitative metrics, such as word frequency analysis and punctuation accuracy, provided objective measures of performance.

Overall, this project not only challenged me to apply theoretical concepts in neural network design and natural language processing but also provided valuable insights into the practical complexities of generating human-like text.

### The Approach



In tackling this problem, I adopted a systematic and experimental approach, deeply rooted in both theoretical insights and empirical testing. My investigation primarily centered on comparing different neural network architectures, specifically LSTMs and GRUs, given their proven efficacy in handling sequential data. I began by reflecting on the unique advantages and trade-offs offered by each architecture, such as the LSTM's capacity to capture long-term dependencies versus the GRU's computational efficiency.

A critical aspect of my methodology involved experimenting with various sequence lengths (100, 200, and 300). I hypothesised that the sequence length would significantly influence the model's ability to capture context and maintain coherence in generated text. To validate this, I constructed base models for each architecture variant (e.g., LSTM with RMSprop, LSTM with Adam) and observed their performance across the different sequence lengths. This allowed me to explore the interplay between the sequence length and the learning dynamics of the model.

Following this initial exploration, I engaged in comprehensive hyperparameter tuning. This process was iterative and informed by both quantitative metrics such as final epoch training and validation accuracy and qualitative assessments of the generated text. I systematically adjusted learning rates, batch sizes, and the number of hidden units, among other parameters, to optimise performance.

After refining the models, I shortlisted several candidates that showed promise based on high final epoch training accuracy and a visual evaluation of the generated words and sentences. This dual-evaluation strategy ensured that the models not only performed well on numerical metrics but also produced coherent and semantically meaningful text.

Finally, I conducted an in-depth error analysis to understand the strengths and weaknesses of each model variant. By analysing cases where the model excelled or struggled to predict the next character accurately, I gained valuable insights into how different architectural choices and hyperparameter settings affected the overall performance.

Overall, this approach allowed me to iteratively refine the models while deepening my understanding of the nuanced challenges inherent in character-level text generation.

## 2. Data Loading & Processing

## Data Loading

```
# read in the text file, transforming everything to lower case
text = open('holmes.txt').read().lower()
print('The original text has ' + str(len(text)) + ' characters.\n')
```

The original text has 562439 characters.

The initial phase of my project involved loading the dataset from the file holmes.txt. I began by reading the file's contents and immediately standardised the text by converting all uppercase letters to lowercase using the .lower() function. This preprocessing step was essential to reduce the variability in the dataset, ensuring that the model's focus remained on learning meaningful patterns rather than being distracted by case differences.

## Data Processing and X/y split

After loading the data, I dedicated considerable effort to the data cleaning process because I knew that bad data in, bad model out. My first task was to remove unwanted formatting characters, such as newline (\n) and carriage return (\r) characters, which often disrupt the continuity of text data. Recognising that not all characters contribute equally to the language model's objectives, I implemented a filtering function designed to extract only the allowable characters specifically, lowercase alphabets and a defined set of punctuation marks. This function iterates over each character, using an if statement to check whether it belongs to the allowed set. Characters that meet the criteria are appended to a new string, char\_list, thereby ensuring that the dataset doesn't contain irrelevant symbols.

The characters remaining after the cleaning are thereafter sorted and displayed in the chars list.

```
def clean_text(text):
    punctuation = ['!', ',', '.', ':', ';', '?', '-', '"', "' "]
    letters='abcdefghijklmnopqrstuvwxyz'
    char_list = ""
    # Enter your code here:
    for i in text:
        if i in punctuation or i in letters:
            char_list += i
    return char_list
```

```
# count the number of unique characters in the text
chars = sorted(list(set(text)))
print(chars)
# print some of the text, as well as statistics
print ("This document has " + str(len(text)) + " total number of characters.")
print ("This document has " + str(len(chars)) + " unique characters.")

[' ', '!', '"', ',', '-', '.', ':', ';', '?', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']
This document has 544340 total number of characters.
This document has 35 unique characters.
```

```
# this dictionary is a function mapping each unique character to a unique integer
chars_to_indices = dict((c, i) for i, c in enumerate(chars)) # map each unique character to unique integer

# this dictionary is a function mapping each unique integer back to a unique character
indices_to_chars = dict((i, c) for i, c in enumerate(chars)) # map each unique integer back to unique character
```

Once the cleaning process was complete, the unique characters from the dataset were sorted and stored in a list named `chars`. This list served as the foundation for creating a mapping dictionary, where each character is paired with a unique integer index. This mapping is crucial for converting textual data into a numerical format that can be readily processed by neural networks.

```
step = 3

def generate_text_io(text, maxlen):
    inputs = [] # store inputs
    labels = [] # stores label

    # Enter your code here:
    for i in range(0, len(text) - maxlen, step):
        inputs.append(text[i: i + maxlen])
        labels.append(text[i + maxlen])
    print('Number of sequences:', len(inputs))

    # List of unique characters in the corpus
    chars = sorted(list(set(text)))
    print('Unique characters:', len(chars))
    # Dictionary mapping unique characters to their index in `chars`
    char_indices = dict((char, chars.index(char)) for char in chars)

    return inputs, labels, char_indices, chars
```

With the dataset cleaned and the mapping established, I then proceeded to generate input sequences (X) and corresponding labels (y) using a custom function, `generate_text`. Initially, I initialised two lists to hold these inputs and labels. I employed a sliding window approach over the dataset, where the window size (specified by the parameter `maxlen`) determined the length of each input sequence. For every sequence extracted, the character immediately following the sequence was designated as its label. Additionally, I introduced a `step` parameter to skip three characters after each sequence, thereby diversifying the training data by preventing consecutive overlaps. This approach not only increased the volume of training examples but also promoted variety in the inputs.



```
import numpy as np

# create a function 'encode_io_pairs' to perform one-hot encoding of inputs and labels
def encode_io_pairs(text, maxlen, inputs): # window_size determines # of characters in each input

    # Enter your code here:
    print('Vectorization...')
    x = np.zeros((len(inputs), maxlen, len(chars)), dtype=np.bool)
    y = np.zeros((len(inputs), len(chars)), dtype=np.bool)
    for i, sentence in enumerate(inputs):
        for t, char in enumerate(sentence):
            x[i, t, char_indices[char]] = 1
        y[i, char_indices[labels[i]]] = 1
    return x, y
```

Subsequently, I applied one-hot encoding to both the inputs and labels. Given the relatively small number of unique characters, one-hot encoding was an appropriate and efficient method for representing the data. I initialised a 3D numpy array filled with zeros using `np.zeros()`, where the dimensions correspond to the number of sequences, the window size, and the total number of unique characters. For each character in the sequence, the corresponding position in the array was set to 1 based on its index from the mapping dictionary. This encoding process was similarly applied to the labels. The resulting arrays, where False represents a 0 and True represents a 1, provide a clear and structured representation of the text data, ready for model training.

```
#window_size = 100 # Models 1-7
window_size = 200 # Models 8-14
#window_size = 300
inputs, labels, char_indices, chars = generate_text_io(text, window_size)
print(inputs[0] + '\n' + labels[0])
X, y = encode_io_pairs(text, window_size, inputs)
print(X[0])
print(y[0])

Number of sequences: 181380
Unique characters: 35
the adventures of sherlock holmes by sir arthur conan doyle  i. a scandal in bohemia ii. the red-headed league iii. a case of identity iv. the boscombe
e valley mystery  v. the five orange pips vi.

Vectorization...
[[False False False ... False False False]
 [False False False ... False False False]
 [False False False ... False False False]
 ...
 [False False False ... False False False]
 [False False False ... False False False]
 [False False False ... False False False]
 [ True False False False False False False False False False
 False False False False False False False False False False
 False False False False False False False False False False]
```

The code below shows how I used different sequence lengths and shows the inputs and labels after it is returned from `generate_text` and after it is one-hot encoded using the `encode_ohe_pairs` function. In this case, False represents a '0' and True represents a '1'. One hot encoding is used in this case since the number of unique possible character labels are small thus this representation is suitable to be used in model training.

```
def sample(preds, temperature=1.0):
    preds = np.asarray(preds).astype('float64')
    preds = np.log(preds) / temperature
    exp_preds = np.exp(preds)
    preds = exp_preds / np.sum(exp_preds)
    probas = np.random.multinomial(1, preds, 1)
    return np.argmax(probas)
```

In addition to preparing the data for training, I implemented a sampling function critical for the model's inference phase. When the model predicts the next character, it outputs a probability distribution over the possible characters. Instead of simply choosing the character with the highest probability, I employed a sampling mechanism that incorporates a temperature parameter. This parameter modulates the randomness of the selection process: a higher temperature (with the default set at 1.0) allows for more exploration and variability, whereas a lower temperature makes the outcome more deterministic. This nuanced approach to sampling is vital, as it ensures that the model can generate text that is both coherent and varied, capturing a more natural and human-like distribution of language.

### 3. Develop the Character Generation Model

The number of output nodes in this case will be determined by the length of the chars list, which is in this case, 35. The activation function used in all models is categorical crossentropy since the problem is a multi- class, single label type of classification. The optimisers used will be defined later in the respective models and accuracy is used as a metric for model evaluation and this also allows for both the training accuracy and loss curves to be plotted later on. But generally, I used the same code structure and the content for building, fitting, and training the models except for instances where I adjusted the batch size during training as part of hyperparameter tuning. And so, only for those cases, I would further explain myself and attach the modified code.

I also decided that since I would be training multiple models and using all of them for prediction and further evaluation would be time consuming (as elaborated in the next step of this Problem), I would shortlist best 5 models for prediction with the 2 requirements:

- Have a high final epoch validation accuracy
- Analyse the coherence of the words generated at last epoch for
  - Temperature = 0.5 and
  - Temperature = 1.0

Model 1 LSTM RMSprop Baseline Model with 100 sequence\_length



```

# Build the Model
model = keras.models.Sequential()
model.add(layers.LSTM(128, input_shape=(sequence_length, len(chars))))
model.add(layers.Dense(len(chars), activation='softmax'))

optimizer = optimizers.RMSprop(learning_rate=0.01)
model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['acc'])
model.summary()

```

Code to build Model 1's architecture.

To start off, for my very first model, I used a Sequence length of 100 with a RMSprop optimiser, and a single layer of LSTM with 128 nodes. The initial learning rate is set at 1e-2. The input shape is set to the product of the sequence\_length(which is 100 for this model) and the length of the chars list which is found to be 35.

```

# Lists to store accuracy and loss for training and validation
train_acc_list = []
train_loss_list = []
val_acc_list = []
val_loss_list = []

# Training the model

for epoch in range(1, 6): # 5 epochs
    print('epoch', epoch)

    # Fit the model for 1 epoch on the available training data
    history = model.fit(X, y, batch_size=128, epochs=1, validation_split=0.2)

    # Store training and validation metrics
    train_acc_list.append(history.history['acc'])
    train_loss_list.append(history.history['loss'])
    val_acc_list.append(history.history['val_acc'])
    val_loss_list.append(history.history['val_loss'])

    # Select a text seed at random
    start_index = random.randint(0, len(text) - sequence_length - 1)
    generated_text = text[start_index: start_index + sequence_length]
    print('--- Generating with seed: "' + generated_text + '"')

```

Code for fitting and training the model.

And this shows the fitting of the model, as well as the generation of the 400 characters repeatedly with the use of the for loop based on the different temperatures. I also included some additional variables and loops which help to collect and store both the training and validation accuracy and loss scores for plotting the model performance curves later on.

The model is fitted and ran for 5 epochs. In this case X which represents the inputs,

y which represents the labels, a batch size of 128 and epochs is set to 1 since the for loop already helps us simulate training for 5 epochs in this case. And I split the data, to have 20% of the data used for validation.

From the dataset, a random start index is generated. Thereafter, the generated text will contain the 400 characters generated based on the starting index.

And now finally the train\_acc\_list, train\_loss\_list, val\_acc\_list, and val\_loss\_list are all initialised to collect the training and validation accuracy and loss values at each iteration of the loop

```
for temperature in [0.2, 0.5, 1.0, 1.2]:
    print('----- temperature:', temperature)
    sys.stdout.write(generated_text)

    # Generate 400 characters
    for i in range(400):
        sampled = np.zeros((1, sequence_length, len(chars)))
        for t, char in enumerate(generated_text):
            sampled[0, t, char_indices[char]] = 1.

        preds = model.predict(sampled, verbose=0)[0]
        next_index = sample(preds, temperature)
        next_char = chars[next_index]

        generated_text += next_char
        generated_text = generated_text[1:]

        sys.stdout.write(next_char)
        sys.stdout.flush()
    print()
```

Code for text generation and temperature testing

This for loop generates text using a machine learning model while testing different randomness levels (temperature). It loops through four temperature values and prints the generated text for each one. For each temperature, it generates 400 characters by looking at the current text, turning it into a format the model understands, and predicting the next character. So, the temperature will affect how random the predictions are where the lower values make the text more predictable, while higher values make it more creative. And the new character is added to the text, and the process keeps going until all 400 characters are generated.

## Plotting Training and validation curves code

```
# Flatten accuracy and loss lists
train_acc_plot = [j for i in train_acc_list for j in i]
train_loss_plot = [j for i in train_loss_list for j in i]
val_acc_plot = [j for i in val_acc_list for j in i]
val_loss_plot = [j for i in val_loss_list for j in i]
```

Code to store the training and validation values for plotting the model performance curves.

Next to be able to plot the model performance curves, I created flattened the nested lists of training and validation accuracy and loss values into simple, one-dimensional lists. The original lists (train\_acc\_list, train\_loss\_list, val\_acc\_list, and val\_loss\_list) contain sublists, where each sublist holds values for different training epochs or batches. Then after that I extracted all the individual values from these sublists and stores them in new lists (train\_acc\_plot, train\_loss\_plot, val\_acc\_plot, and val\_loss\_plot). And then now I'm able to plot the performance curves since the values are now in a single continuous sequence instead of being grouped into separate sublists.

```
# Plot training and validation accuracy
epochs = range(1, len(train_acc_plot) + 1)

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(epochs, train_acc_plot, 'bo-', label='Training Accuracy')
plt.plot(epochs, val_acc_plot, 'ro-', label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Plot training and validation loss
plt.subplot(1, 2, 2)
plt.plot(epochs, train_loss_plot, 'bo-', label='Training Loss')
plt.plot(epochs, val_loss_plot, 'ro-', label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```

Code to plot training and validation curves

I used len(graph\_acc), which is the length of the cleansed training accuracy list which I used for the plotting instead of the variable 'epochs' which is set to 1 and so it will not work iterating through the list with the help of the range function in this case scenario.

```
----- temperature: 0.5
ently and the story with the state and the simple and a concentice and the street of the comman of the man in it which he arrev
er and dress the man are which we hear left her made the socing in the first of the daid which are, read a small at the stare i
t some brien brien to the last a great him and for a confiction of the point. it was comment to the sair man in the singul was
a matter him you are of the land it had man are the man was all the last at the state, his position was at the all of t
----- temperature: 1.0
e of the land it had man are the man was all the last at the state, his position was at the all of the word to be this client b
rrais so doving out back reknieve.'! his nat the dead-tair ma. asknen, smell of his oblight it out tograid dreat in the things.
yes, the tack and ever it in the carrking briencenty down he man! he considerain, as it adreast?prevent in.its all he askun, fo
rwholewethe preserve, for s hanged all really into then'togine methreek, i propoced one at to sent me at the lamy, ifo
```

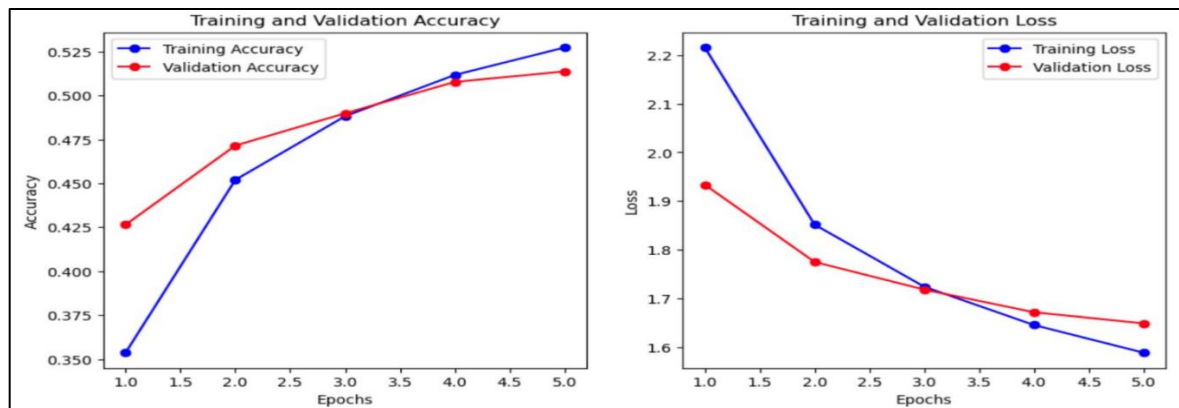
Generated text by Model 1



I used an [online spell checker tool](#), on the words generated by Model 1, I found that:

- At temperature = 0.5, approximately 75% of the words are English
- At temperature = 1.0, approximately 68% of the words are English

I think that the results of this model were moderate since it was able to generate some coherent English sentences or sequences of words but with some spelling and grammar errors. And since this was my first model, which is my baseline model, I shortlisted this as one of the models to evaluate.



Model 1 Training and Validation Curves

Here, the validation loss and accuracy initially outperform training which is a sign the model generalised well early on before converging by epoch 3. Post-convergence, we can see that the training accuracy edges ahead, which could mean overfitting and that the model has started memorising training nuances. Similarly, the training loss drops faster, while validation loss plateaus slightly higher. And I was considering if this means that model might benefit if trained longer.

```
epoch 5  
1134/1134 — 404s 356ms/step - acc: 0.5268 - loss: 1.5849 - val_acc: 0.5137 - val_loss: 1.6487
```

Final epoch results

As shown, the model achieved a validation accuracy of 51.37 % and a loss score of 1.6487 after 5 epochs of training.

## Model 2 LSTM RMSprop with 2 LSTM layers 100 sequence\_length

For this second model, I decided to start simple with the fine tuning and simply add a layer of 128 nodes to the LSTM Model. I kept the rest of the hyperparameters the same, including the initial batch size of 128.

```
# Build the Model 2
model2 = keras.models.Sequential()
model2.add(layers.LSTM(128, return_sequences=True, input_shape=(sequence_length, len(chars))))
model2.add(layers.LSTM(128))
model2.add(layers.Dense(len(chars), activation='softmax'))

optimizer = optimizers.RMSprop(learning_rate=0.01)
model2.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['acc'])
model2.summary()
```

Code to build Model 2's architecture.

My reasoning for Model 2's architecture was, I expected an improved model result since I've increased the complexity of the model's architecture in Model 2.

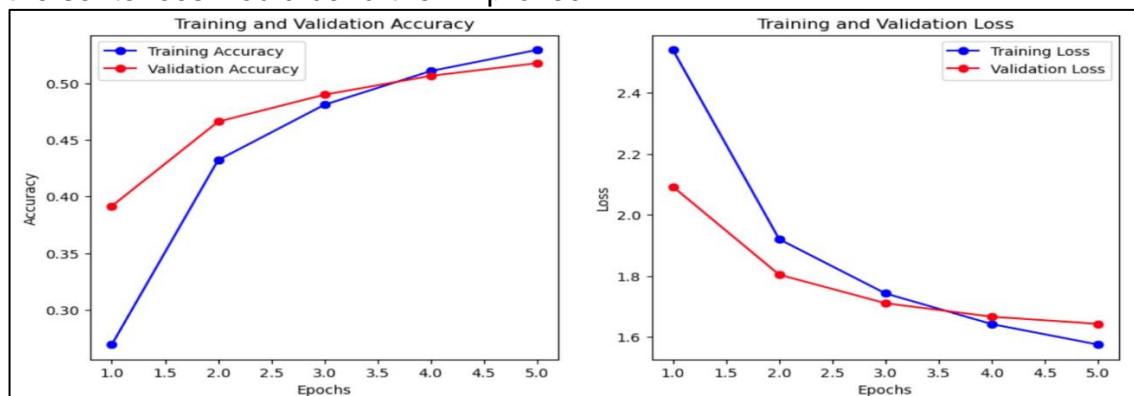
```
----- temperature: 0.5
the starriage and his compossible to be a commonal and things and the stard of the constion of the other, and the door, but wi
th the coat which i ammpised to my ewith the call for the man straction which lead both for a commise of the correr of the stan
d in the singular and thing of the lady come which may goss if so some for the right have will passels the dood. i have been st
ood to the windows on the pocket, i was the singular to the last on the some and this on the man was in the darkered, a
----- temperature: 1.0
on the pocket, i was the singular to the last on the some and this on the man was in the darkered, abmult brow my waiting for i
n them of the lever tricingbetween pied from anything 'no parnty fast nearly, inmerturesword of itpution to a nealfarwery hand.
sunder really swing by the hands forwhat a miss your'mupbed story, one of my lodge hissertal nothing how shorp condered, and i
white?'we was very emqurgy to take woment byingrispoffected:minute is bosion dirpapriives to be reache--jest rixe, sai
```

Generated text by Model 2

Using the same [online spell checker tool](#), I checked the words generated by Model 2:

- At temperature = 0.5, approximately 88% of the words are english
- At temperature = 1.0, approximately 83% of the words are english


Thus, in terms of generating the semi-coherent English sentence, model 2 performed much better as compared to Model 1 but the coherence and the flow of the sentences would be further improved.



Model 2 Training and Validation Curves

Model 2 achieved a slightly improved final epoch validation accuracy of 51.8%, which is almost 4% higher as compared to Model 1. It also has a slightly lower loss score of 1.64. And to be expected, it takes more than twice as long to train as compared to model 1 because of the added complexity of the model, with an average training time of 2000s per epoch. However, the similarity drawn from the 2 models is that the accuracy starts increasing at a decreasing rate, suggesting that maximum accuracy of the model may be reached soon and since the curve isn't

stagnant, we could potentially increase the number of epochs.

epoch 5 1134/1134  1952s 2s/step - acc: 0.5309 - loss: 1.5655 - val_acc: 0.5175 - val_loss: 1.6423
--

Final epoch results

Since increasing the number of layers resulted in an insignificant improvement final epoch validation accuracy and had minimal improvement on the coherence of the words generated, I decided to remove the additional layer and test out other hyperparameters instead.

### **Model 3 LSTM Adam Baseline Model 100 sequence\_length**

For this model, the general architecture is exactly the same as the Base Model in Model 1, with the exception of the optimiser. I switched from an RMSprop to an Adam optimiser to assess if this would have a much more significant improvement on the final epoch accuracy, as well as the percentage of English words, which



determines the coherence of the English sentence.

```
# Build the Model
model3 = keras.models.Sequential()
model3.add(layers.LSTM(128, input_shape=(sequence_length, len(chars))))
model3.add(layers.Dense(len(chars), activation='softmax'))

optimizer = optimizers.Adam(learning_rate=0.01)
model3.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['acc'])
model3.summary()
```

Code to build Model 3's architecture.

I kept the batch size the same as Model 1 at 128.

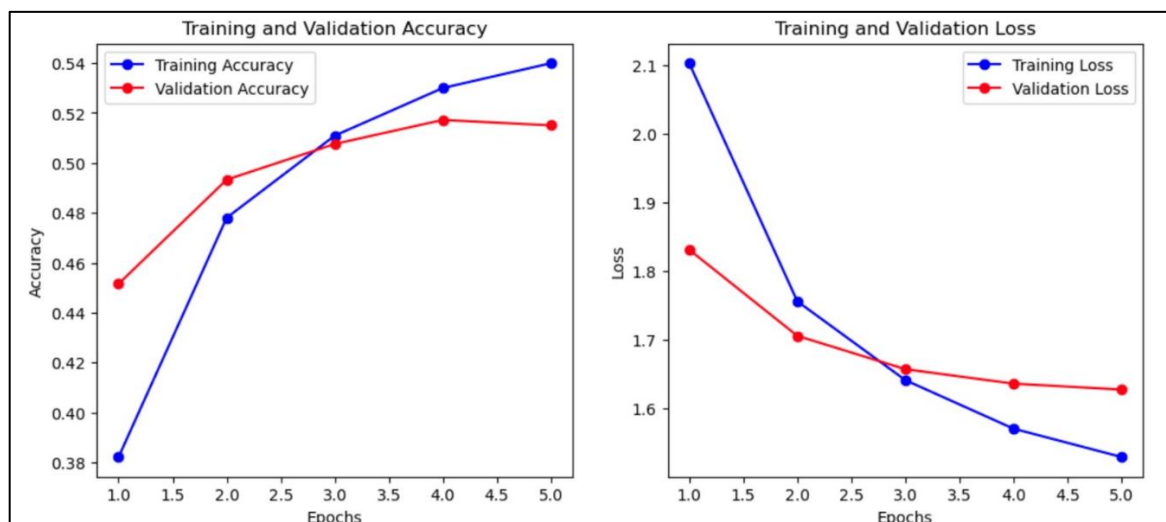
```
----- temperature: 0.5
h a little to be a street. it is a man to the come of the start of the man to the cold have a come on the closed had been or a
back to the startle to the cold you to be the morning to her little truage of the face, with my room. it is a cleart which was
obvious, brike the contrared at the watsto talk and that i could you blue the confinely are all the man or course to the land o
f the paster of the man to his closs was destair to his face of a probled to part of man to the occanly good--a consto
----- temperature: 1.0
e man to his closs was destair to his face of a probled to part of man to the occanly good--a constoupe.german.'when what i sun
trainh, whisellachlitt me mother, so if it would call red to the slight, from for then sy.'the peculiaged.it'st arxamationed.wh
eno's hadsent his lensung hard cullark. and is incrong park. thet we was an.he betal have refasher opening of me a quick of fro
nter.holmes ofknow oecoot, i won't littles took got wearhed a purficolate ofdidy a very featon of his clossige for it t
```

Generated text by Model 3

Using the same [online spell checker tool](#), I checked the words generated by Model 3

- At temperature = 0.5, approximately 89% of the words are english
- At temperature = 1.0, approximately 73% of the words are english

Compared to the RMSprop Base Model, this model had more english words generated at both temperatures as compared to Model 1. This implies that change in the optimiser definitely has a large impact on the coherence of the language generated.



Model 3 Training and Validation Curves

In both the training and validation loss and accuracy charts, it shows an increasing

difference in the training and validation curves, which suggests overfitting. While training accuracy steadily increases to over 0.54, validation accuracy plateaus around 0.52 after an initial rise. This could indicate the model is increasingly memorising training data rather than generalising to unseen data. Similarly, the training loss decreases consistently, while the validation loss, although initially declining, flattens and shows a slight upturn towards the later epochs. All of these suggests the model is beginning to overfit.

epoch 5	1134/1134	544s	479ms/step	- acc: 0.5428	- loss: 1.5199	- val_acc: 0.5150	- val_loss: 1.6271
---------	-----------	------	------------	---------------	----------------	-------------------	--------------------

Final epoch results

Despite the model showing a significant improvement in the percentage of actual English words, the validation accuracy still doesn't show a significant improvement and only in the training accuracy.

#### **Model 4 LSTM Adam 2 layers, increase learning rate 100 sequence\_length**

Thereafter, I tried to fine tune the Adam model by experimenting with a slightly different architecture. For this model, I used 2 layers of 64 LSTM nodes, because from model 2, I realised that adding more nodes would have a negative effect on both the accuracy as well as the coherence of the text generated. Thus, I reduced the number of nodes to 64, hoping that this would not severely affect the accuracy of the model. I also increased the learning rate to 5e-2 from 1e-2, with the batch size the same.

```
# Build the Model
model4 = keras.models.Sequential()
model4.add(layers.LSTM(64, return_sequences=True, input_shape=(sequence_length, len(chars))))
model4.add(layers.LSTM(64))
model4.add(layers.Dense(len(chars), activation='softmax'))

optimizer = optimizers.Adam(learning_rate=0.05)
model4.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['acc'])
model4.summary()
```

Code to build Model 4's architecture.

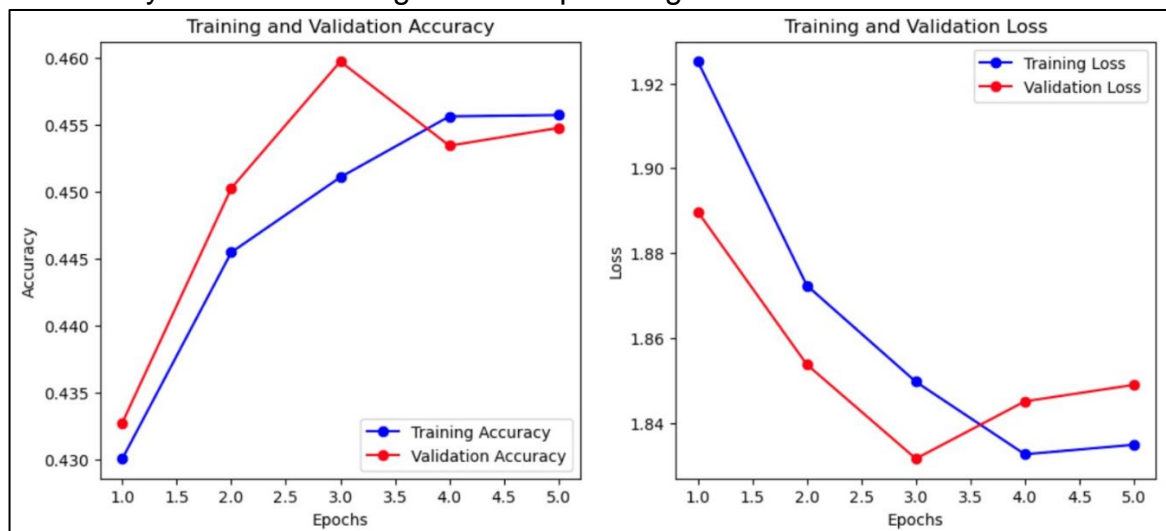
```
----- temperature: 0.5
e that i was i had that i had the had the had the had that i had the side that i had the hard a startion a pright his should to
a that i way a should you that i have a cal was that i was the said holmes had to little strarklather a thindress that i was th
e was it had to the to on the hassistrarthat i had to the seemion do a cal that i wass to a sand a chaid that he had in the lad
starthe had to the seadinter the was i pird who hadsome a shall a canning a marrear come was i was the got a more a s
----- temperature: 1.0
e seadinter the was i pird who hadsome a shall a canning a marrear come was i was the got a more a steanst mire her what conon,
spracter to the everyinates to no mywhich everess has time it his weigniongeusics.we haviile the maid to the coursh my that bui
anter, but i colve the rever bok, that he that we seare me, that i the hant ofbegk asthat his indibletsing fso beatinge rayespa
ss candrielachenththe apoor amd the wectnain had midifubly.holmes, if wasine right.ot variard out shallspancha hotse
```

Generated text by Model 4

Using the same [online spell checker tool](#), I checked the words generated by Model 4

- At temperature = 0.5, approximately 87% of the words are english
- At temperature = 1.0, approximately 88% of the words are english

The text generated started to have more repeated words in this instance as the model tries to emphasise a particular point, which is often due to limitations in its vocabulary or understanding of certain phrasings.



Model 4 Training and Validation Curves

The model's curves have strong fluctuations and drop in the accuracy and loss and the training accuracy starts to drop at 3 epochs. The loss score isn't great and isn't bad either but the sudden change in the curves is very unusual and interesting. And I think the sudden changes and the drop in performance could be a result from the increase in the learning rate to five times greater than all the previous models.

```
epoch 5  
1134/1134 — 397s 350ms/step - acc: 0.5188 - loss: 1.6129 - val_acc: 0.4977 - val_loss: 1.7088
```

### Final epoch results

There wasn't any improvement in the accuracy and the loss and instead showed a poorer performance. Thus, I'll be moving onto another model architecture.

### Model 5 GRU RMSprop Base Model 100 sequence\_length

Since adding additional layers and changing the learning rate or optimiser did not perform as expected and showed a poorer performance, I decided to then go on to explore using Gated Recurrent Units (GRUs) instead. For the base model I have a layer of 128 units and an RMSprop optimiser with a learning rate of magnitude  $1e-2$ , as I was trying to compare if there was progress made comparing the LSTM and GRU base models. I kept the Sequence length at 100.

```
# Build the Model  
model5 = keras.models.Sequential()  
model5.add(layers.GRU(128, input_shape=(sequence_length, len(chars))))  
model5.add(layers.Dense(len(chars), activation='softmax'))  
  
optimizer = optimizers.RMSprop(learning_rate=0.01)  
model5.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['acc'])  
model5.summary()
```

Code to build Model 5's architecture.

```

----- temperature: 0.5
and the looking and light to the advection and in the allooking in the lady and a very allow and and of a little to an and lig
ht and presing, and it was a man, and i cause and the door and in his ereselp and a priors in the reach was a door every to th
e eright there are us in the bellare in miles of the lady at his allsow, it would like a distince in in the reamorsignate one o
f the allarked it down the barrow and a begn that i was and in the astion of the hands ararried the man the allors anse
----- temperature: 1.0
own the barrow and a begn that i was and in the astion of the hands ararried the man the allors ansell,thand, i reeally i.he wa
s dark?you--a mightok, as. ently, and lught tenk a goo retare hormoming 'to knew a parmainially alvice butappess in the letter
s grise withoutgent.i must do down estarlara, iadnce, in there not me.nocked, i seotther, and i totoot drassed at a fiight, se
very some neight whoke sow, rycip i known tody us was you looking as which could kind--so. there walstok leave in the a

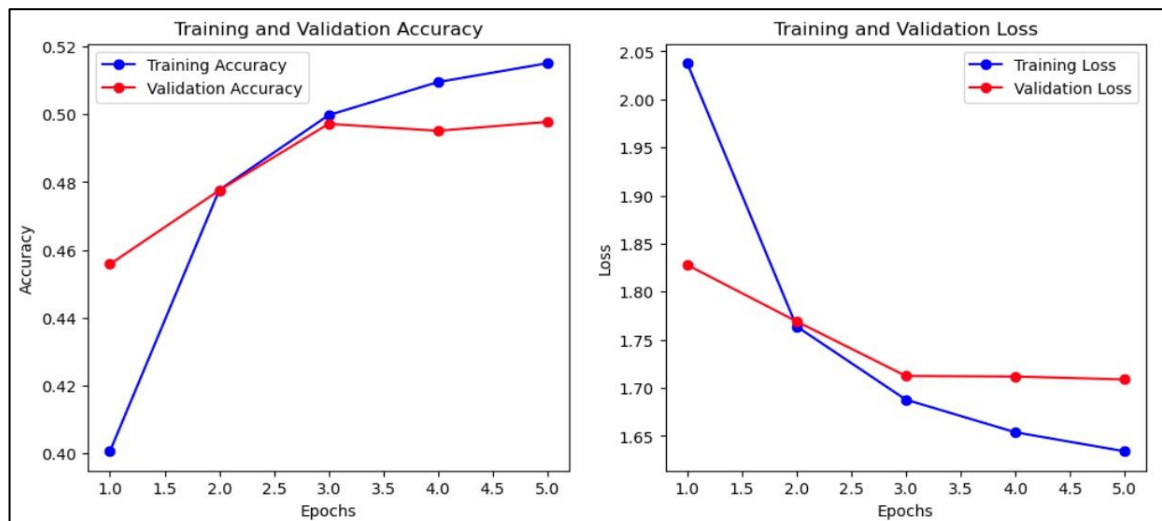
```

### Generated text by Model 5

Using the same [online spell checker tool](#), I checked the words generated by Model 5

- At temperature = 0.5, approximately 87% of the words are English
- At temperature = 1.0, approximately 79% of the words are English

Even though there was a moderate percentage of English words generated at temperature 1.0, the sentences did not really make sense, and it wasn't as coherent as the sentences generated at temperature 0.5.



### Model 5 Training and Validation Curves

This model achieved a validation accuracy of 49.77%, which is poorer as compared to LSTM Base Models. It also achieved a loss score of 1.7088 after being ran for 5 epochs. I think there's also some potential mild overfitting. Training accuracy and loss steadily climbs to approximately 0.52, while validation accuracy and loss, though initially tracking closely, plateaus around 0.50, suggesting a divergence in performance on unseen data. But the validation loss can remain higher than training loss.

```

epoch 5
1134/1134 — 397s 350ms/step - acc: 0.5188 - loss: 1.6129 - val_acc: 0.4977 - val_loss: 1.7088

```

### Final epoch results

This model performs worse than the LSTM base model in Model 1 by 2 to 3% in the accuracy which isn't that bad.



## Model 6 GRU RMSprop 2 layers with dropout 100 sequence\_length

Here I added an additional layer with 64 nodes and added dropout and recurrent dropout to help to prevent the model from becoming stagnant and capping at its maximum training accuracy too fast.

```
# Build the Model
model6 = keras.models.Sequential()
model6.add(layers.GRU(128,return_sequences=True, input_shape=(sequence_length, len(chars))))
model6.add(layers.GRU(64, dropout = 0.1, recurrent_dropout = 0.1))
model6.add(layers.Dense(len(chars), activation='softmax'))

optimizer = optimizers.RMSprop(learning_rate=0.01)
model6.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['acc'])
model6.summary()
```

Code to build Model 6's architecture.

```
----- temperature: 0.5
we was not which was a little was a little was a man which was a little stating which was not which was for streak to dead, but
do not station was was to the give the door which was a long with the stored after the stade a morny it was a latture were some
with to see when you have an and what wes see about which was of was and door which you thing with the when i am a prise with a
long which was a long the lane it words the give and moverading to be all you could do you was mores which will were
----- temperature: 1.0
long the lane it words the give and moverading to be all you could do you was mores which will weren,, whaince i havestandss t
he for intin to,wes swencaid's. that einling of them your was,for of it days and no rewatt. step mardow, whtlestaning of the lo
m witchay, belangationedinteringt's doby, i rathy to was, tho all light which, prose ofand,wes see no own mo, jafy stattedrainc
ut a turne, for redingtude ow by whithin, or, i would, ' so-did your appened with fy sprige infoched. go two the tise v
```

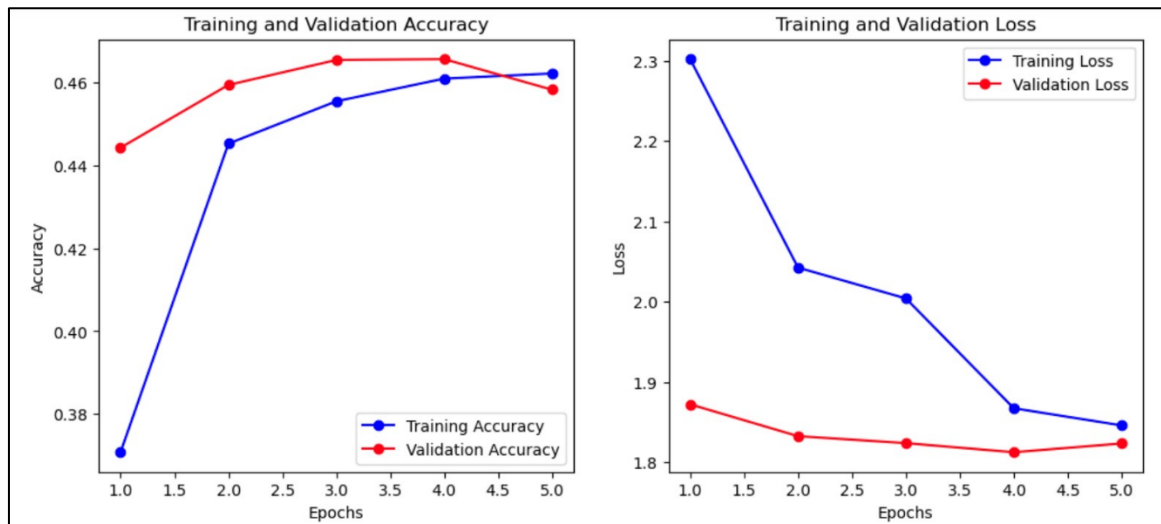
Generated text by Model 6

Using the same [online spell checker tool](#), I checked the words generated by Model 6

- At temperature = 0.5, approximately 77% of the words are English
- At temperature = 1.0, approximately 64% of the words are English

The text generated from both the temperature at 0.5 and 1.0 shows that Model 6 is struggling to generate coherent sentences or understand the overall meaning as the percentage of of actual English words is extremely low.





Model 6 Training and Validation Curves

Training accuracy steadily improves to around 0.46, while validation accuracy, though initially higher, plateaus around 0.47 and exhibits a slight decline towards the end, indicating a divergence in performance. Similarly, the training loss sharply decreases, while validation loss plateaus and fluctuates, remaining above 1.8. I was thinking that this could be because of the added layer and units.

epoch 5  
1134/1134 — 751s 662ms/step - acc: 0.4640 - loss: 1.8370 - val\_acc: 0.4582 - val\_loss: 1.8233

Final epoch results

The accuracy of Model 6 is just decreased significantly as compared to than Model 5, which shows that the tuned model didn't help the model generalise better.

## Model 7 GRU Adam 100 sequence\_length

Since the above parameters did not help to improve model accuracy and instead lowered the performance, I tried a different optimiser to check if this would have the same effect on the GRU model as it had on the LSTM architectures. For this I kept the rest of the hyperparameters the same, including the batch size of 128, the Sequence length of 100 and ran it for 5 epochs, with the optimiser I used is Adam instead of RMSprop.

```
# Build the Model
model7 = keras.models.Sequential()
model7.add(layers.GRU(128, input_shape=(sequence_length, len(chars))))
model7.add(layers.Dense(len(chars), activation='softmax'))

optimizer = optimizers.Adam(learning_rate=0.01)
model7.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['acc'])
model7.summary()
```

Code to build Model 7's architecture.

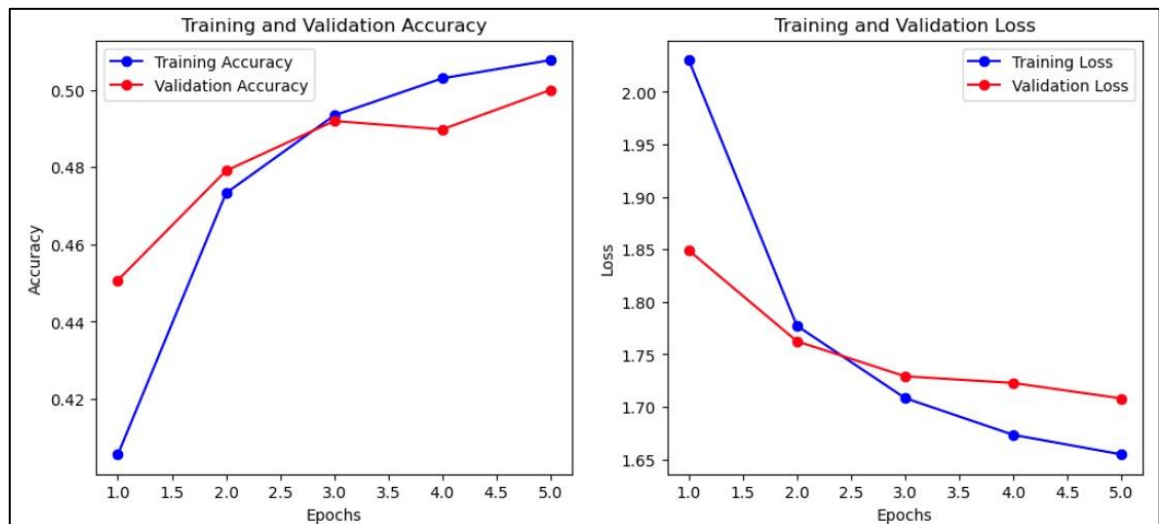
```
----- temperature: 0.5
or with the stands and to the pares and the stands a cound the stands and the stands and to the counting of the standed to make
to been the stands the trunk that i cleared. i have could get of the stapped to perion and to hand to a toppair be all-might th
e stope.what is a packed, which i have man at the down the vine.and is and to the sold to is at it has and in the door and with
a county make and and to simply man that he say the in the man was the stapped to down at the dear. i had been a trie
----- temperature: 1.0
and to simply man that he say the in the man was the stapped to down at the dear. i had been a trie cunture toon even ansterpy
sm--noom, and man the therlo-die's rood.ahm was knew htwere.vileay leadniandselve, howextle to thealtine, singe that iman, outc
hastle towathim towat and handrably to at dool-stike clear.i famiat, he lond, what you call down ireathes,fifit, thut tompati
on. i canst, wookin might hold gatherle. 'who poilt out, smeass inteentarriedchulleggandby-pace.thip of the andlace mar
```

Generated text by Model 7

Using the same [online spell checker tool](#), I checked the words generated by Model 7

- At temperature = 0.5, approximately 85% of the words are English
- At temperature = 1.0, approximately 53% of the words are English

Both temperature at 0.5 and 1.0 still shows that Model 7 is struggling to generate coherent sentences as the percentage of actual English words is extremely low.



Model 7 Training and Validation Curves

However, the final epoch validation accuracy was very similar to the RMSprop Base Model 5 as I expected.

```
epoch 5  
1134/1134 — 522s 460ms/step - acc: 0.5110 - loss: 1.6416 - val_acc: 0.5000 - val_loss: 1.7080
```

Final epoch results

After several hyperparameters already, I decided to try a different sequence length of 200 where I had to rerun the functions `generate_text` and `encode_ohe_pairs`.

## Model 8 LSTM RMSprop Change learning rate, add dropout 200 sequence\_length

For this model, I chose to keep the number of layers at 1 and add a lower magnitude of dropout to see if this would have a positive effect on the final epoch training accuracy as well as the quality of the words generated. I also increased the magnitude of the learning rate to 5e-2.

```
# Build the Model
model8 = keras.models.Sequential()
model8.add(layers.LSTM(128, dropout = 0.005, recurrent_dropout = 0.005, input_shape=(sequence_length, len(chars))))
model8.add(layers.Dense(len(chars), activation='softmax'))

optimizer = optimizers.RMSprop(learning_rate=0.05)
model8.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['acc'])
model8.summary()
```

Code to build Model 8's architecture.

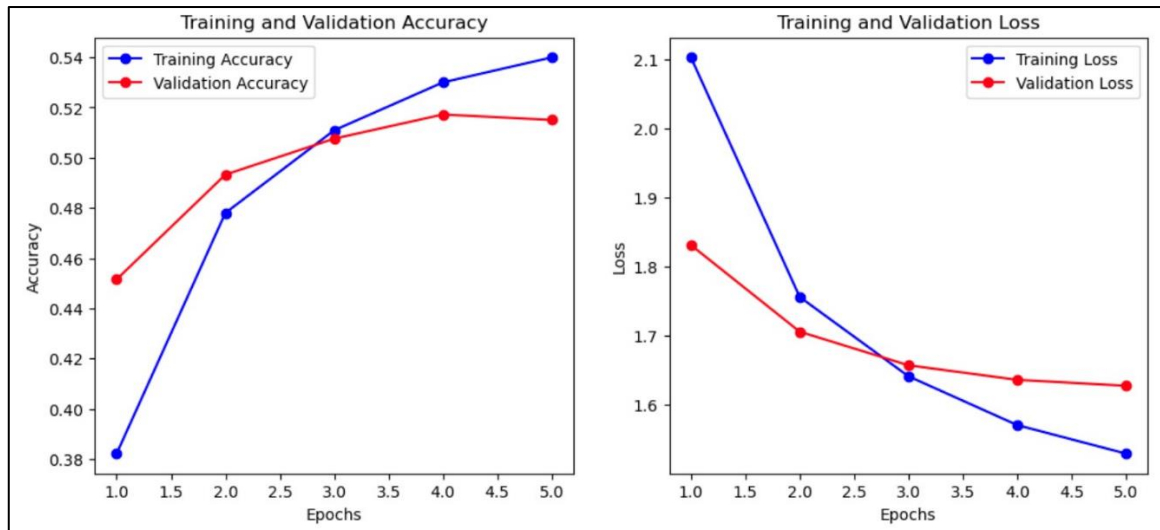
```
----- temperature: 0.5
d that i were all the track to the man to the compraced to the word the compance to the face to the track to the room the commo
n that the song the sondered to the colem. i have a sondered that the man a crips ratter and thefatter wish a guent if i have a
covered to the the get the more a with a best me it to a son setty. they were was a crimines to the sumperge me to the sort to
the corner with the man this is this were went and the igned that the son to wat and had have the red the seltost. and that wer
e man loth in the seced to in the raying the between that i have a in a distend to the man
----- temperature: 1.0
e man this is this were went and the igned that the son to wat and had have the red the seltost. and that were man loth in the
seced to in the raying the between that i have a in a distend to the man well, rilinguntile, holmes roomhorast. they roof was,
i was of the isonting to that the had mind, i prack for that dombest be nami tome town secreculing componord itthe whice for t
he colud with anything.' oh, sid see your.throwing.theresis her usfi's klok-- heddream this, door me gard.dome in these man tom
ystere to mind the demed. with the ahahn compvanshe.hind sarwthy satelo,much, i worn of alas
```

Generated text by Model 8

Using the same [online spell checker tool](#), I checked the words generated by Model 8

- At temperature = 0.5, approximately 75% of the words are English
- At temperature = 1.0, approximately 67% of the words are English

Both temperature at 0.5 and 1.0 still shows that Model 7 is struggling to generate coherent sentences as the percentage of actual English words is extremely low but there is a slight improvement in the coherence.



Model 8 Training and Validation Curves

The training accuracy of the validation accuracy rose very sharply from 1 to 2 epochs and subsequently dropped gradually to about 51% before tapering off and plateauing later on.

```
epoch 5
1134/1134 — 1510s 1s/step - acc: 0.5097 - loss: 1.6558 - val_acc: 0.5072 - val_loss: 1.6892
```

Final epoch results

## Model 9 LSTM RMSprop change learning rate 200 sequence\_length

For this model I experimented with increasing the number of nodes in the LSTM layer from 128 to 512 to see if this will increase the accuracy. I also changed the learning rate to a smaller magnitude of 1e-3, down from the previous 1e-2 in the LSTM base model.

```
# Build the Model
model9 = keras.models.Sequential()
model9.add(layers.LSTM(256, input_shape=(sequence_length, len(chars))))
model9.add(layers.Dense(len(chars), activation='softmax'))

optimizer = optimizers.Adam(learning_rate=0.01)
model9.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['acc'])
model9.summary()
```

Code to build Model 9's architecture.

```
----- temperature: 0.5
se that i had been the strong and a parting that the man who had been a little between the strange of the companion with the st
range of the strange of the strong the corner and a little the strange the companion. it is in a leagure of the lady lady that
the door in into the all parting the dear your papers from the lance of a sight of the came of the man and read and contradical
arged and a little some thing was a very faced the came of the possed to do in a large the day's of the blue complete be an ide
ned in the chears in the face, who had been of the room that i shall prove that the simple a
----- temperature: 1.0
some thing was a very faced the came of the possed to do in a large the day's of the blue complete be an idened in the chears i
n the face, who had been of the room that i shall prove that the simple a from the kell, with my tintery in believe opened in a
gightic in sixt's case very subcuster, but i allow very amoranw so crugglide, revart paper is made unwilligure you and have in
the rownd were beconversisent of the small sabout of the remember and man was door blotch abrabgic of goose repole track with a
case from minucate on the agopaiaing as where mean he something outor thebroadedout of th
```

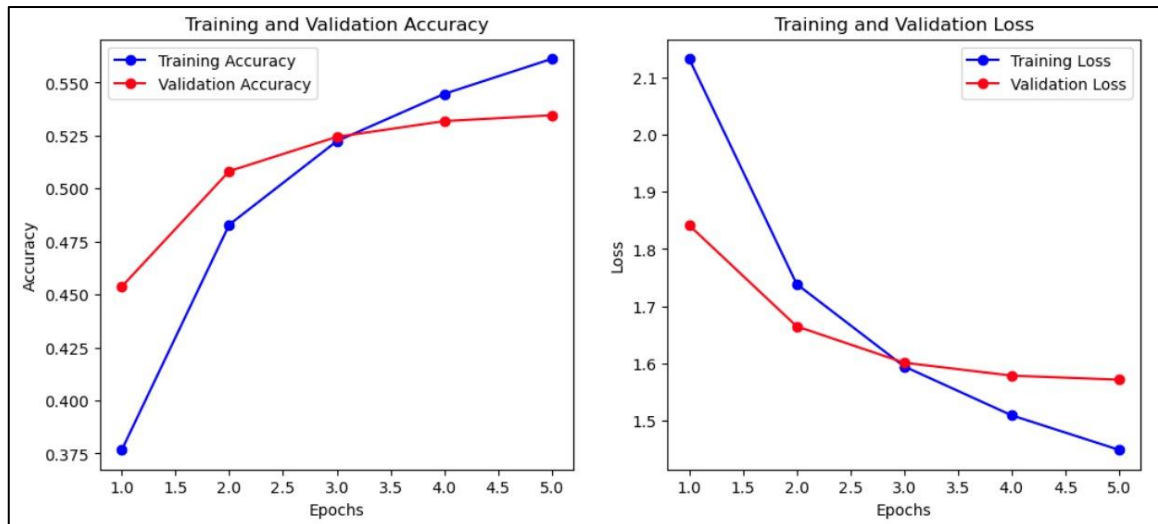
Generated text by Model 9

Using the same [online spell checker tool](#), I checked the words generated by Model 9

- At temperature = 0.5, approximately 85% of the words are English
- At temperature = 1.0, approximately 87% of the words are English

The temperature at 0.5 and 1.0 generated more coherent text and some of the text makes sense which shows an improvement from the previous model.





### Model 8 Training and Validation Curves

The model achieved quite a high final epoch validation accuracy of 53.76% although it still does not show a significant sign of plateauing. I think that this is a significant improvement from the base model.

```
epoch 5
1134/1134 — 6290s 6s/step - acc: 0.5625 - loss: 1.4379 - val_acc: 0.5346 - val_loss: 1.5718
```

Final epoch results

## Model 11 GRU RMSprop Base Sequence length 200

I then moved on to train a GRU model. In this case, I increased the number of nodes to 512 and decreased the learning rate further to 1e-4, as doing so had seen positive effects on overall model performance as shown in the previous iteration of experimentation with the various hyperparameters. I used an RMSprop optimiser for this model with a batch size of 128.

```
# Build the Model
model11 = keras.models.Sequential()
model11.add(layers.GRU(512, input_shape=(sequence_length, len(chars))))
model11.add(layers.Dense(len(chars), activation='softmax'))

optimizer = optimizers.RMSprop(learning_rate=0.001)
model11.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['acc'])
model11.summary()
```

Code to build Model 11's architecture.

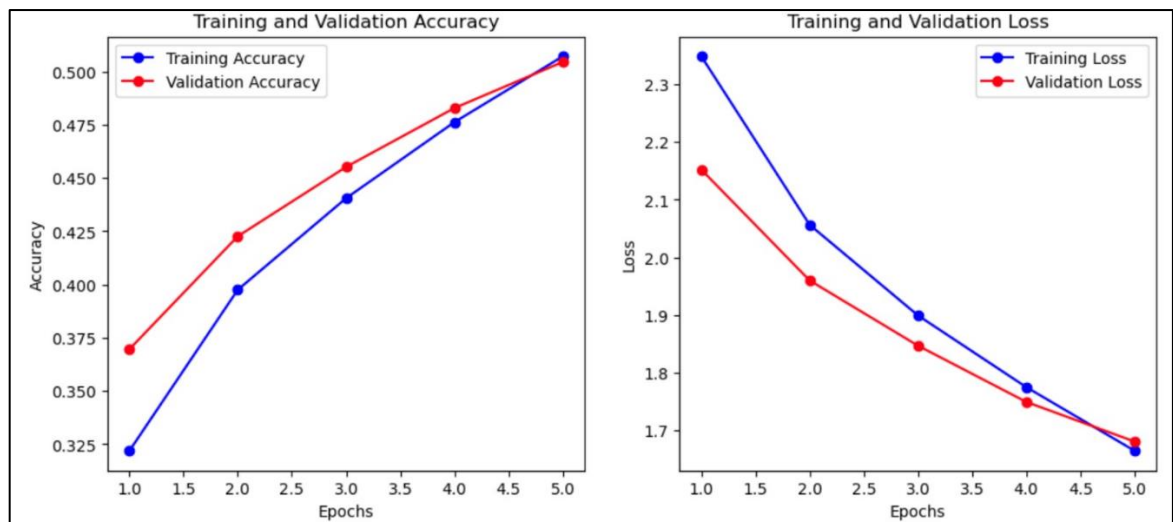
```
----- temperature: 0.5
on the sing of the sing that the states of the read and stard the single and the firet of the corner of the office when i have
been the single of the face to the corner of the self could not the corner with a single of a glanced the ordinctical this with
a sirntle in the rish of the one of the said the from the forcheward do be with the room at the fort, and we to the to the dear
nich i have to the word of the was off colf in the clay dight of nowning but the sedgethe was his came the read of a lotters of
sime to one for the ling stands that poshis comesting up the remored of such a colfing and
----- temperature: 1.0
word of the was off colf in the clay dight of nowning but the sedgethe was his came the read of a lotters of sime to one for t
he ling stands that poshis comesting up the remored of such a colfing and omenting my ewhich salitinely not of years lewnle as
a plectshimbecome, whichto out wousted it wit, firely aparks. they to be come of the op downing thequeptile the just with a she
gh facthis lendonad to the lidst up what befind st. grive, an wethere ilany wanging upon the knotis watablere ot tained. there
a closswhat holver seefchat theseat and half atulalid s'tyhe wiman the mirature. thel as ver
```

Generated text by Model 11

Using the same [online spell checker tool](#), I checked the words generated by Model 11

- At temperature = 0.5, approximately 88% of the words are English
- At temperature = 1.0, approximately 84% of the words are English

At both the temperature at 0.5 and 1.0, even though a lot of the text was recognisable, the sentences didn't really have any meaning and it sounded like a collection of words instead of a somewhat understandable sentence.



### Model 11 Training and Validation Curves

The model achieved a moderate validation accuracy of 50%, which is a pretty average score.

```
epoch 5
1134/1134 — 3745s 3s/step - acc: 0.5050 - loss: 1.6689 - val_acc: 0.5045 - val_loss: 1.6811
```

Final epoch results

## Model 13 copy of Model 9 but with 10 epochs

I wanted to experiment if I could further improve the Model 9 because of its good accuracy score and thus, I used the same hyperparameters apart from increasing the number of epochs to 10.

```
# Build the Model
model13 = keras.models.Sequential()
model13.add(layers.LSTM(256, input_shape=(sequence_length, len(chars))))
model13.add(layers.Dense(len(chars), activation='softmax'))

optimizer = optimizers.Adam(learning_rate=0.01)
model13.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['acc'])
model13.summary()
```

Code to build Model 13's architecture.

```
----- temperature: 0.5 a colonel was struck the stone the stone which he had been some the stone of the stone with a sheet and stoner to me to the stone of the states at the stone and i will be
a consider the stone of the colonel to me in the other shouldersatality which he can footed the carriage a same and the friend which i sat the colonel with a little famelittle to street his chair, and
when he had beginst of a light of the bed to stood a start and the single dark little man stood a monograph, and the name, said holmes. i have already must be a satisfyea at a convicial a dark
and had heard a shabe and so it i

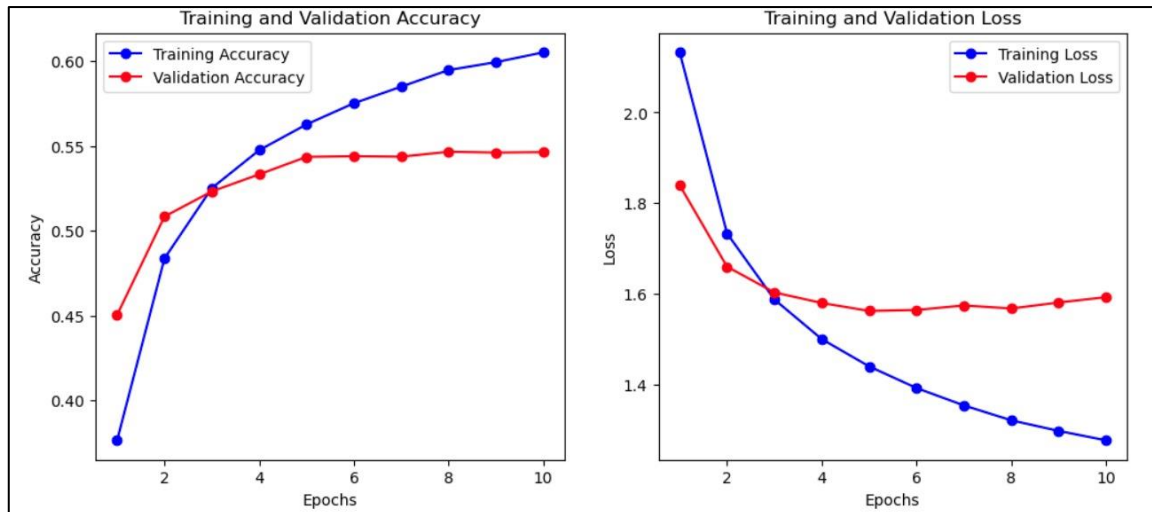
----- temperature: 1.0 ght of the bed to stood a start and the single dark little man stood a monograph, and the name, said holmes. i have already must be a satisfyea at a convicial a dark and
had heard a shabe and so it is evidently for who may neil and placed wand the invisitar of leashed my is squy the'lapt to deexed at six more for his own boordow, as he was bricked a deave a
nothing matter to mes the as six ywould a colonel is a to carry. jodd i beg the provesus very letters at the s. walked uccursed manne heavers were joc mannes made a curren, and when a dr.
anoth.then drevionled as ratherward to agannites of
```

Generated text by Model 13

Using the same [online spell checker tool](#), I checked the words generated by Model 13

- At temperature = 0.5, approximately 95% of the words are English
- At temperature = 1.0, approximately 90% of the words are English

The temperature at 0.5 and 1.0 generated more coherent text and some of the text makes sense at temperature 1.0 which shows an improvement from the previous model. And the appropriate full stops for shorter terms like 'dr.' for doctor was used, which was really fascinating to me as other models didn't show such improvements.



Model 13 Training and Validation Curves

Firstly, a significant gap is seen in the curves between the training and validation curves which signify overfitting but on the positive side, the model has reached an all time high for the validation accuracy and the validation loss with the best scores yet. And I think this could support the hypothesis that the percentage of English words is in a relationship with the performance of the model accuracy and loss.

```
epoch 10
1134/1134 — 2635s 2s/step - acc: 0.6111 - loss: 1.2529 - val_acc: 0.5464 - val_loss: 1.5928
```

#### Final epoch results

I was really happy with the accuracy for this model since they achieved 54% accuracy, which is the highest for the models trained so far. And it also had a highest percentage of English words so far as well. I think im quite happy with my decision of increase the number of epochs to 10 but I think I could have stopped the model early to tone down the overfitting using early stopping.

## Model 14 LSTM Adam Sequence length 150

I used 150 sequences as I wanted to identify if this could help balance it. Thereafter I decided to train an Adam model instead to see if it would do slightly better than the RMSprop base model. Thus, I decided to first increase the learning rate slightly to a magnitude of 5e-3. But this resulted in my computer crashing once again so I decided to increase the batch size from 256 to 512.

```
# Build the Model
model14 = keras.models.Sequential()
model14.add(layers.LSTM(512, input_shape=(sequence_length, len(chars))))
model14.add(layers.Dense(len(chars), activation='softmax'))

optimizer = optimizers.Adam(learning_rate=0.01)
model14.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['acc'])
model14.summary()

# Training the model
for epoch in range(1, 6): # 5 epochs
    print('epoch', epoch)

    # Fit the model for 1 epoch on the available training data
    history = model14.fit(X, y, batch_size=512, epochs=1, validation_split=0.2)

    # Store training and validation metrics
    train_acc_list.append(history.history['acc'])
    train_loss_list.append(history.history['loss'])
    val_acc_list.append(history.history['val_acc'])
    val_loss_list.append(history.history['val_loss'])

    # Select a text seed at random
    start_index = random.randint(0, len(text) - sequence_length - 1)
    generated_text = text[start_index: start_index + sequence_length]
    print('--- Generating with seed: "' + generated_text + '"')
```

Code to build Model 14's architecture.



```

----- temperature: 0.5
in the street the door of the stand of the man who is a said holmes and the street that i have a langer and a carried the coun
try to the ling of the dack to a preaced and dunded the side the patter of the poon at a carting a ready has no door in a windo
w, and the paped at lain at the long it for my propession. with a carrow stark and that the stone of the day in a very gave an
a little letter down the lang and street of the paled his feet in the chair for a hand of the stremide is a clue down a strange
that he seeding have soleting upon the la
----- temperature: 1.0
n the lang and street of the paled his feet in the chair for a hand of the stremide is a clue down a strange that he seeding ha
ve soleting upon the latter with when you leave bomn the beel.my, and will not half anyhunger, toovery paper upon a defore, str
enk good has paper of the windowway?'a thank their letter, mr.what youthough made? wellot glad shood to his know. i lite, and oc
he the fator was commind ony amvalpmusting orcaming ot late, father in somef pear.than, i thoughteryou. he thote of their putt
urat. i'll in that the more handly, to you

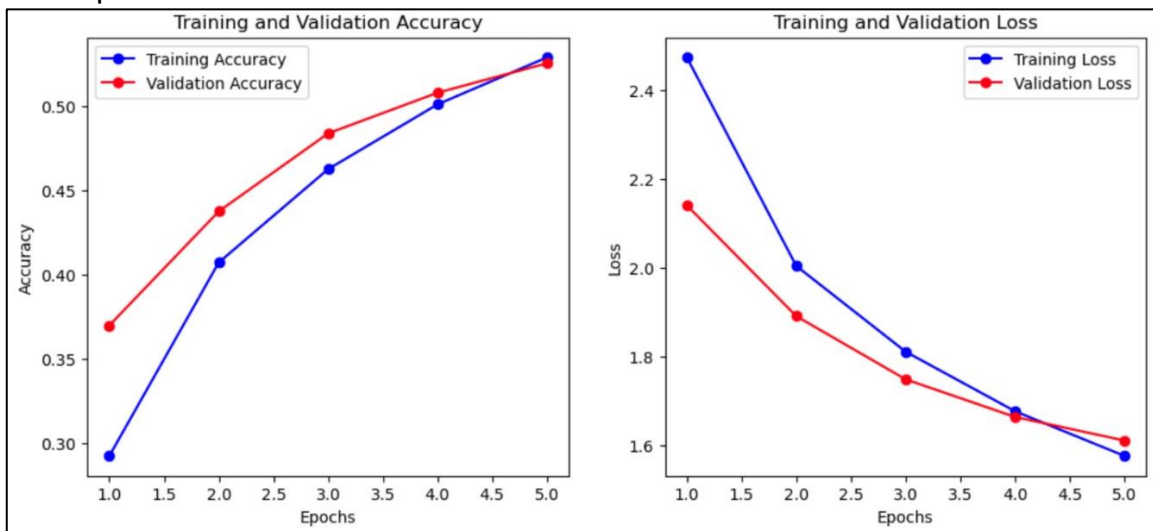
```

### Generated text by Model 14

Using the same [online spell checker tool](#), I checked the words generated by Model 14

- At temperature = 0.5, approximately 88% of the words are English
- At temperature = 1.0, approximately 80% of the words are English

The temperature at 0.5 and 1.0 generated more text that were correctly spelled but most of the sentences don't really make sense which could be a sign to train over more epochs.



### Model 14 Training and Validation Curves

There is also no sign of the accuracy plateauing in this model as it seems to be rising at a steady rate of about 4% per epoch. The model achieves a final epoch validation accuracy of 52.56%, and it also has very little sign of overfitting which is great. In the future, where there's more time. This model can be further pushed to more epochs till there's a more significant split between the training and validation curves.

```

epoch 5
284/284 ————— 4583s 16s/step - acc: 0.5264 - loss: 1.5827 - val_acc: 0.5256 - val_loss: 1.6105

```

### Final epoch results

#### Key insights:

- A sequence length of 200 is the optimal since models achieve the highest final epoch validation accuracy scores and good coherence in text.

- Recurrent dropout and dropout has a huge impact on the model's performance.
- Using the same hyperparameters, LSTM models tend to have better performance compared with GRU models.
- I think that there is a linear relationship between the final epoch accuracy and the percentage of english words in the generated characters for some models.

#### 4. Use the Models to Make Predictions and Recommending the Best Model

All the models

Model	Type	Final Epoch Training Accuracy (%)	Final Epoch Validation Accuracy (%)	Final Epoch Training Loss (%)	Final Epoch Validation Loss (%)
1	LSTM RMSProp Baseline Model for LSTM	52.7	51.4	1.58	1.65
2	LSTM RMSProp with 2 LSTM layers	53.1	51.8	1.57	1.64
3	LSTM Adam 1 LSTM layer	54.3	51.5	1.52	1.63
4	LSTM Adam Increase learning rate 2 LSTM layers	45.5	45.5	1.83	1.84
5	GRU RMSProp Baseline Model for GRU	51.9	49.8	1.61	1.71
6	GRU RMSProp 2 GRU layers	46.4	45.8	1.84	1.82
7	GRU Adam 1 layer	51.1	50.0	1.64	1.71
8	LSTM RMSProp decrease learning rate and dropout	51.0	50.7	1.66	1.69
9	LSTM Adam Increase Model Capacity to	56.3	53.5	1.44	1.57

	256 units				
11	GRU RMSProp Base Sequence length 200	50.5	50.5	1.67	1.68
13	Model 13 copy of Model 9 but with 10 epochs	61.1	54.6	1.25	1.59
14	LSTM Adam Sequence length 150	52.6	52.5	1.58	1.61

Since the performance of the model was quite varied even in similar architectures, I only chose the top 2 models based on the coherence and the final epoch validation accuracy. Below are the top 2 models that I shortlisted.

Model	Type	Final Epoch Training Accuracy (%)	Final Epoch Validation Accuracy (%)	% of english words at temperature = 0.5	% of english words at temperatur e = 1.0
13	Model 13 copy of Model 9 but with 10 epochs	61.1	54.6	95	89
14	LSTM Adam Sequence length 150	52.6	52.5	90	70

Since these 2 models meet the first two conditions of my evaluation matrix, I then proceeded on to further evaluate the models to determine the best model based on the 3 other criteria.

- Appropriate punctuations placements
- Be able to generate a label that can possibly be used construct and English word given a new input

### **Requirement 1: Must have appropriate punctuations placements**

For this criterion, we mainly have to consider the text generated at temperature = 0.5 since we want coherence of the sentences generated. In this situation the model is expected to often predict a definite label for a given input seed and thus the texts at a lower temperature are of a higher importance.

### **Model 13**

There are definitely more english words generated at temperature = 0.5 for this model as compared to Models 1 and 10. At the start at temperature 0.5, the phrase “the stone” kept coming up, which shows repetition in the generated text.

For temperature = 1.0, lesser repetitions of words were used and thus why I chose to use temperature 1.0 when testing out generating text by user input in the next segment. There are places where fullstops are used before names and pronouns which is appropriate, and the full stop is used after “dr.” to show that it is a short form of the word doctor. And that was really interesting to me. But there are some places where fullstops were overly used as well.

```
----- temperature: 0.5 a colonel was struck the stone the stone which he had been some the stone of the stone with a sheet and stoner to me to the stone of the states at the stone and i will be
a consider the stone of the colonel to me in the other shouldersatily which he can footed the carriage a same and the friend which i sat the colonel with a little famelittle to street his chair, and
when he had beginst of a light of the bed to stood a start and the single dark little man stood a monograph, and the name, said holmes. i have already must be a satisfyea at a conviecial a dark
and had heard a shabe and so it i

----- temperature: 1.0 ght of the bed to stood a start and the single dark little man stood a monograph, and the name, said holmes. i have already must be a satisfyea at a conviecial a dark and
had heard a shabe and so it is evidently for who may neil and placed wand the invisitar of leashed my is squy the'lapt to deexed at six more for his own boordow, as he was bricked a deave a
nothing matter to mes the as six ywould a colonel is a to carry. jodd i beg the proveseue very letters at the s. walked occursed manne heavers were joc mannes made a curren, and when a dr.
anoth.then drevionled as ratherward to agannites of
```

## Model 14

There are some inappropriate uses of punctuation in this instance at temperature = 0.5. For example, the use of a comma in the beginning of the third line after. There is also an open quote suggesting that someone is speaking but no closing quotation marks. However, there is still a high percentage of english words that make up the generated text in this instance.

As for temperature = 1.0, there are some inappropriately placed commas and full stops and fewer english like words.

```
----- temperature: 0.5
in the street the door of the stand of the man who is a said holmes and the street that i have a langer and a carried the coun
try to the ling of the dack to a preaced and dunded the side the patter of the poon at a carting a ready has no door in a windo
w, and the paped at lain at the long it for my propession. with a carrow stark and that the stone of the day in a very gave an
a little letter down the lang and street of the paled his feet in the chair for a hand of the stremide is a clue down a strange
that he seeding have soleting upon the la

----- temperature: 1.0
n the lang and street of the paled his feet in the chair for a hand of the stremide is a clue down a strange that he seeding ha
ve soleting upon the latter with when you leave bomn the beel.my, and will not half anyhunger, toovery paper upon a defore, str
enk good has paper of the windowway?'a thank their letter, mr.what youthough made? wellot glad shood to his know. i lite, and oc
he the fartor was commind ony amvalpmusting orcaming ot late, father in somef pear.than, i thoughteryou. he thote of their putt
urat. i'll in that the more handly, to you
```

Based on my observations, I would rank Model 13 to be closest to a semi coherent english text followed by Model 14.

## Requirement 2: Be able to generate a label that can possibly be used construct and English word given a new input

Firstly, we need to load the model to evaluate it using the text inputs.

Thereafter the user is prompted to enter some text which will be fed into the model later as the data and predict a label.

```
from tensorflow.keras import models
import numpy as np
model = models.load_model('char_rnn_model_14.keras')
```

In this case I decided to use 3 test texts to evaluate each model's performance

based on this requirement.

- the game is afoo (t) ## the iconic phrase in the sherlock holmes series
- he nodde (d)
- deep learning is fu (n)

Thereafter I defined two functions as shown below to encode the text\_input.

```
def encode_test_text(text_input, maxlen):
    print('Vectorization...')
    x = np.zeros((len(text_input), maxlen, len(chars)), dtype=np.bool)
    for i, sentence in enumerate(text_input):
        for t, char in enumerate(sentence):
            x[i, t, char_indices[char]] = 1
    return x

def predicta(preds):
    preds = np.asarray(preds).astype('float64')
    preds = np.log(preds)
    exp_preds = np.exp(preds)
    preds = exp_preds / np.sum(exp_preds)
    probas = np.random.multinomial(1, preds, 1)
    print(probas)
    return np.argmax(probas)
```

In this case we do not need to one-hot encode any labels since the label is generated using the predicta() method, which is rather similar to the sample() method in the data sampling step.

After the label is one-hot encoded using encode\_test\_text(), the encoded data is

```
x = encode_test_text(text_input, window_size)
preds = model.predict(x, verbose=0)[0]
next_index = predicta(preds)
print(next_index)
next_char = chars[next_index]
print(f"The predicted next character for {text_input} is {next_char}")
```

sent into the .predict() function which returns a probability distribution function on the possible labels. The result of this is then fed into the predicta() function and the next character is retrieved by indexing the element from the chars list defined earlier during data processing.

Since the predicta() function may output different labels every time it is run, I chose to feed the data inputs twice to the model.

the game is  $a_{foo}(t)$

In this case the character generated, 'l' could be the next character for a 2-word phrase "a fool" which is commonly used in book genre's like Sherlock Holmes. So that could be a possible character generation of the model trying to generate a word. And I think that it is relevant in this scenario because of the context of the dataset.

he node(d)

But in this case, the correct character 'd' wasn't predicted correctly and instead predicted a similar character 'r'. And both are common characters in the end of many words. While not a standard English word in this context, looks like a word and that suggests the model is at least grasping some aspects of English morphology.

deep learning is  $f_u(n)$

In this case, the correct character 'n' wasn't correctly predicted and instead a character 'g' was generated. The word formed after the character generated was completely unrelated to the prompt and shows no understanding of either English



words or the Sherlock Holmes context. Hence, why this is not relevant.

```
Vectorization...  
[[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]]  
15  
The predicted next character for ['deep learning is fu'] is g
```

### Model 14 (1/3) 1 label predicted correctly

the game is afoo(t)

Enter a string of characters: the game is afoo

In this case the character generated, 'd' could be the next character in words like food. But the character wasn't correctly predicted here.

```
Vectorization...  
[[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0]]  
21  
The predicted next character for ['the game is afoo'] is m
```

he nodde(d)

Enter a string of characters: he nodde

But in this case, the correct character 'd' wasn't predicted correctly and instead predicted a similar character 't'. And both are common characters in the end of many words.

```
Vectorization...  
[[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0]]  
28  
The predicted next character for ['he nodde'] is t
```

deep learning is fu(n)

Enter a string of characters: deep learning is fu

In this case, the correct character 'n' was correctly predicted.

```
Vectorization...  
[[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0]]  
22  
The predicted next character for ['deep learning is fu'] is n
```

Reasons for the results

Based on the results, Model 13 is the best model for this problem. Even though it had there was some overfitting with a gap between the training accuracy and the validation accuracy, it still outperformed all of the models in the 2 evaluation tests, probably due to the fact that it has high percentages of english words and hence that will guarantee that it has the best coherence amongst all of the models.

I think that model 14 actually could have been improved by training it over more epochs as the model hasn't started to plateau yet and there could be a possibility that it hasn't reached its cap accuracy and loss yet. So if time permits, I'll probably go experiment with increasing the number of epochs for Model 14 after the assignment submission. If interested, you could look up my GitHub as I'll probably share more about the model's performance after increasing the epochs.

## 5. Summary

Upon completing the model evaluation, I took the opportunity to reflect on the results and consider how both the models and my overall approach could be enhanced in future work. This reflective process revealed several avenues for improvement that I believe are worth pursuing.

Firstly, regarding model training, further hyperparameter exploration appears promising. In addition to experimenting with different batch sizes, I plan to investigate the impact of incorporating L1 and L2 regularisation techniques. These methods could potentially improve model generalisation and performance. I also think that while I applied dropout and recurrent dropout concurrently, testing these strategies independently might yield clearer insights. It is possible that one type of dropout may significantly improve performance, while the other could detrimentally affect it.

Moreover, I was thinking of experimenting with architectures featuring an even higher number of nodes per layer (example, 1024 or 2048) to determine whether the added capacity translates into improved learning of the language structure. Adjusting the "step" parameter during data preprocessing could also be influential, and exploring different configurations here might further enhance accuracy.

Secondly, time constraints and no GPU availability meant that I could not evaluate every possible model variant across all evaluation criteria. Although I selected models based on high training accuracy and positive visual assessments, it is possible that some models even surpassing the performance of Model 16 were overlooked. Future work, ideally with more robust computational resources, should aim for a comprehensive evaluation of all candidate models to ensure no promising approaches are missed. Furthermore, when time permits, I'd like to explore ensemble methods like stacking or voting to potentially combine the strengths of different models and improve overall performance.

Finally, a key observation from my assignment was that LSTM models consistently outperformed GRU models when using identical hyperparameters. This suggests that the additional complexity of LSTMs, such as the inclusion of an output gate for

better memory control, may confer a significant advantage in capturing the nuances of character-level language generation. So, I was thinking of further researching on optimising LSTM architectures, as they hold greater potential for high-performance text generation.

This reflective summary not only highlights the strengths and limitations of my current approach but also lays a clear roadmap for future experiments aimed at refining and advancing character-level language modeling.