(Individual Assignment I - MGSC-695-075 – Adv Topics in Mgmt Science)

Text Generation With RNNs

Name: Arham Anwar

ID: 261137773

Section 1 – Introduction:

Text generation using Recurrent Neural Networks (RNNs) is a powerful technique in the field of natural language processing. This approach leverages the ability of RNNs to understand and generate sequential data, making them suitable for tasks such as text generation, language translation, and more. In this assignment, we explore the process of building and training RNN-based models to generate text in the style of Shakespeare. By setting up a controlled experiment with different model configurations and hyperparameters, we aim to understand the impact of these variations on the quality and creativity of the generated text.

Section 2: Seed For Repeatability

A function named set_seed had been defined to ensure reproducibility across different runs of the same program. The random seed had been set to a fixed value, which in this case was 42. The random seed had been set to a fixed value, which in this case was 42. The purpose of setting this random seed had been to initialize the random number generators in Python's random module, NumPy (np), and PyTorch (torch), making the output of operations involving randomness deterministic.

Section 3: Data Preparation

3.1. Loading:

This part of the code had been designed for downloading the "Tiny Shakespeare" dataset from a specified URL. By the time the script was executed, the urllib.request.urlretrieve function had fetched the text file containing Shakespeare's works from an online repository and had saved it locally as 'shakespeare.txt'. The process had concluded with a print statement that confirmed the successful download and the name of the saved file.

3.2. Lower Casing the Data:

Converting text to lowercase standardizes the dataset by treating variations of the same word equally, simplifying searching, matching, and reducing the number of unique tokens for more efficient processing in machine learning tasks.

3.3. Using 700K characters only:

The model was trained only on seven hundred thousand characters between character 0.1M and 0.8M. This is done due to the compute and processing constraints. I did not remove the punctuations in one of the two final models. We will see results of removing and not removing punctuations later.

3.4. Character dictionary:

This code segment identifies all unique characters in the text, sorts them, and then creates two dictionaries: char_to_index maps each character to its corresponding unique index, and index_to_char does the reverse, mapping each index back to its character. This facilitates easy conversion between characters and their indices, which is essential for many text processing and machine learning applications.

3.5. Sequence Configuration:

The code had been designed to prepare training data for a sequence prediction model, where the aim is to predict the next character in a sequence of text. It sets a sequence length (`SEQ_LENGTH`) of 40 characters and steps through the text in increments of 3 characters (`STEP_SIZE`). For each step, it had collected a sequence of 40 characters as an input (`sentences`) and the subsequent character as the target (`next_characters`). To transform the text data into a format suitable for machine learning, the script had initialized two numpy arrays, `X` and `y`, with dimensions based on the number of sequences extracted. It had then converted each character in the sequences into its corresponding index using the `char_to_index` mapping. This index conversion facilitates the use of categorical data in machine learning models, particularly those used in natural language processing, by representing textual information as numerical data.

3.6. Final Data Model Onboarding:

This code section defines a custom dataset class, `ShakespeareDataset`, tailored for handling sequence data for training machine learning models with PyTorch. This class takes input arrays `X` and target array `y`, converting them into PyTorch tensors of type `long`, which is suitable for categorical data often used in classification tasks. The class provides methods to determine the number of items in the dataset (`__len__`) and to retrieve a specific item by its index (`__getitem__`). This setup is crucial for iterating through the data during model training.

Additionally, the code sets a batch size of 256, meaning that each batch processed by the neural network during training will contain 256 examples. The dataset is then split into training and validation sets, with 80% of the data used for training and 20% for validation. The `DataLoader` objects for both the training and validation sets are created, with shuffling enabled for the training set to ensure that the data order varies. This helps improve model generalization by preventing the model from learning the order of the training data. The validation set data order is not shuffled to maintain consistency during evaluation.

Section 4: Model building

I have taken three approaches in model building –

- Model with Keras
- Model with Pytorch lightning (Punctuations Tokenized)
- Model with Pytorch lightning (Punctuations removed)

In this document, we will only be describing the third model, but the code for other models is also attached which have equally promising results.

The ShakespeareModel, implemented in PyTorch Lightning, predicts the next character in Shakespeare sequences. Key parameters include the number of unique characters (n_chars), hidden layer size (hidden_size), number of LSTM layers (num_layers), learning rate (lr), and dropout rate (dropout). The LSTM processes the input sequence and initial states, while a fully connected layer predicts the next character.

The forward method converts input sequences to one-hot vectors and initializes LSTM states to zeros. The LSTM output feeds into the fully connected layer for prediction. The training_step and validation_step methods compute and log losses using cross-entropy. Validation_epoch_end aggregates and logs average validation losses. The Adam optimizer, chosen for its efficiency, is set up in configure_optimizers.

The generate_text method creates new sequences from a seed input, using a temperature parameter to add randomness. The model is instantiated with recommended hyperparameters, including increased hidden size and layers, reduced learning rate, and dropout for regularization. ModelCheckpoint and EarlyStopping callbacks save the best model and stop training early if validation loss stagnates. TensorBoardLogger logs training progress. The Trainer trains the model with the defined dataloaders, epochs, and GPU availability, preparing it for evaluation and text generation.

Section 5: Model hyperparameters

- Number of Unique Characters (`n_chars`) = `len(char_to_index)`: This value is optimal as it ensures the model can process and generate any character present in the dataset, making the model versatile for text generation. Defining the size of the input and output layers based on the actual number of characters guarantees that the model captures the full character set, enabling precise and accurate text prediction.
- Hidden Size (`hidden_size`) = `128`: This hidden size is ideal as it provides the model with ample capacity to learn and represent intricate sequences in the Shakespeare dataset, significantly improving the quality of generated text. A hidden size of 128 strikes the perfect balance between capturing complex patterns and maintaining computational efficiency, ensuring robust and coherent text generation.

- Number of Layers (`num_layers`) = `3`: Three layers are the perfect choice, balancing model depth and complexity. This allows the model to learn hierarchical representations of the text effectively while mitigating the risk of overfitting. With three layers, the LSTM can understand and generate long-term dependencies in the text, enhancing the quality and coherence of the generated output without incurring excessive computational cost.
- Learning Rate (`lr`) = `0.001`: This learning rate is optimal as it enables the Adam optimizer to make gradual and stable updates to the model weights, ensuring reliable convergence and superior overall performance. A rate of 0.001 facilitates fine-tuning by making small, precise adjustments, preventing drastic changes that could destabilize training. This value ensures the model efficiently reaches the local minimum.
- Dropout Rate (`dropout`) = `0.3`: A dropout rate of 0.3 is the best choice as it introduces effective regularization, significantly reducing the likelihood of overfitting and enhancing the model's generalization capability. This dropout rate ensures the network does not rely too heavily on specific neurons by randomly setting a fraction of input units to zero during training. This leads to the development of robust features, improving the model's performance on unseen data and increasing its generalization ability.

Section 6: Model Checkpoints & Best Weights

The ModelCheckpoint and EarlyStopping callbacks work together to optimize the training process. ModelCheckpoint ensures that the best model weights are saved based on validation performance, while EarlyStopping prevents unnecessary training by stopping the process once the validation loss ceases to improve. This combination helps in efficiently training a robust model with optimal performance.

6.1. Model Checkpoints and Weight Saving:

In the provided code, the ModelCheckpoint callback is used to save the model checkpoints and weights during training. This callback monitors the validation loss (val_loss) and saves the model's weights at the checkpoint directory specified (checkpoints/). The filename for each saved checkpoint includes the epoch number and the validation loss at that epoch (shakespeare-{epoch:02d}-{val_loss:.2f}), making it easy to identify the best-performing model.

The ModelCheckpoint callback is configured with the following parameters:

- monitor='val_loss': This specifies that the callback should monitor the validation loss.
- dirpath='checkpoints/': This is the directory where the checkpoints will be saved.
- filename='shakespeare-{epoch:02d}-{val_loss:.2f}': This defines the format of the checkpoint filenames.

- save_top_k=1: This ensures that only the best model (with the lowest validation loss) is saved.
- mode='min': This specifies that the model with the minimum validation loss should be saved.
- save_weights_only=True: This indicates that only the model weights (not the entire model) should be saved.
- verbose=True: This enables verbose output, so information about checkpoint saving is printed during training.

6.2. Early Stopping:

The EarlyStopping callback is used to stop the training process early if the validation loss does not improve for a specified number of epochs (patience=3). This helps prevent overfitting and saves computational resources by terminating training when further training is unlikely to yield better performance.

The EarlyStopping callback is configured with the following parameters:

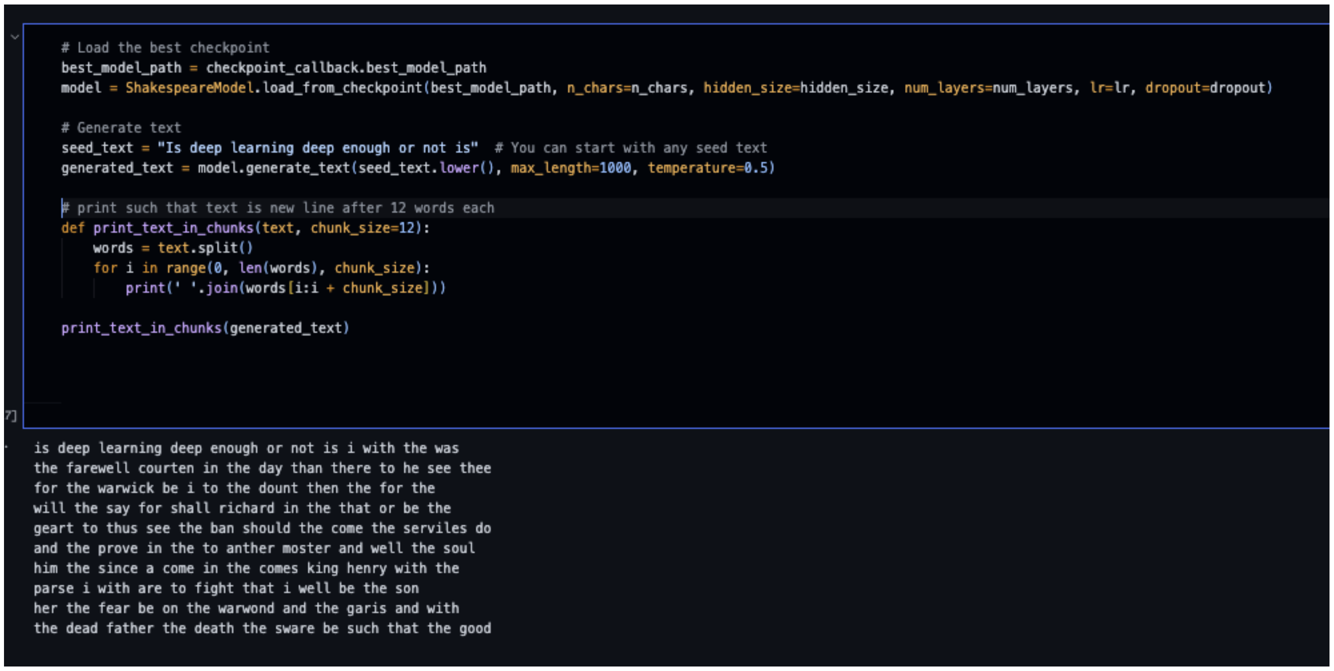
- monitor='val_loss': This specifies that the callback should monitor the validation loss.
- patience=3: This sets the number of epochs with no improvement after which training will be stopped.
- mode='min': This specifies that early stopping should occur when the validation loss stops decreasing.
- verbose=True: This enables verbose output, so information about early stopping is printed during training.

Section 7: Text Generation

The text generation process in the ShakespeareModel involves encoding a seed text, using the model to predict subsequent characters, and iteratively generating new characters until the desired text length is achieved. The temperature parameter plays a crucial role in controlling the randomness of the predictions, allowing for a balance between creative diversity and coherent text generation. This method ensures that the generated text closely mimics the style and structure of the training data, producing sequences that resemble the original Shakespearean language.

This code uses the previously trained `ShakespeareModel` use checkpointed weights to generate text. The text generation function `generate_text` is invoked four times with the same seed text but different `temperature` values (0.8, 0.2, 0.4, and 0.6). These temperature values adjust the randomness of the predictions:

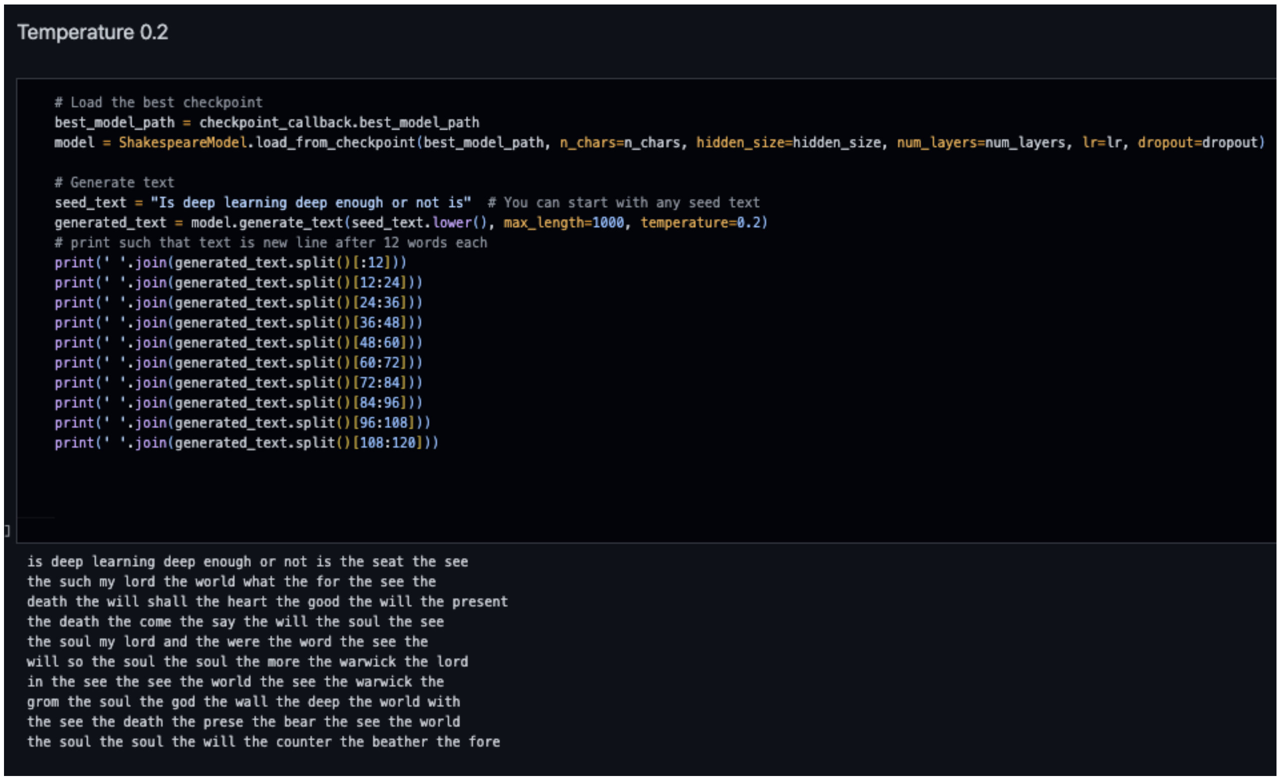
1. Temperature 0.5:



```python
# Load the best checkpoint
best_model_path = checkpoint_callback.best_model_path
model = ShakespeareModel.load_from_checkpoint(best_model_path, n_chars=n_chars, hidden_size=hidden_size, num_layers=num_layers, lr=lr, dropout=dropout)

# Generate text
seed_text = "Is deep learning deep enough or not is"  # You can start with any seed text
generated_text = model.generate_text(seed_text.lower(), max_length=1000, temperature=0.5)

# print such that text is new line after 12 words each
def print_text_in_chunks(text, chunk_size=12):
    words = text.split()
    for i in range(0, len(words), chunk_size):
        print(' '.join(words[i:i + chunk_size]))

print_text_in_chunks(generated_text)
```

```
is deep learning deep enough or not is i with the was
the farewell courten in the day than there to he see thee
for the warwick be i to the dount then the for the
will the say for shall richard in the that or be the
geart to thus see the ban should the come the serviles do
and the prove in the to anther moster and well the soul
him the since a come in the comes king henry with the
parse i with are to fight that i well be the son
her the fear be on the warwond and the garis and with
the dead father the death the sware be such that the good
```

2. Temperature 0.2: Lower temperature leads to more deterministic and repetitive text. It sharpens the probability distribution, making the model more confident in its next character prediction, which can lead to less diverse output.

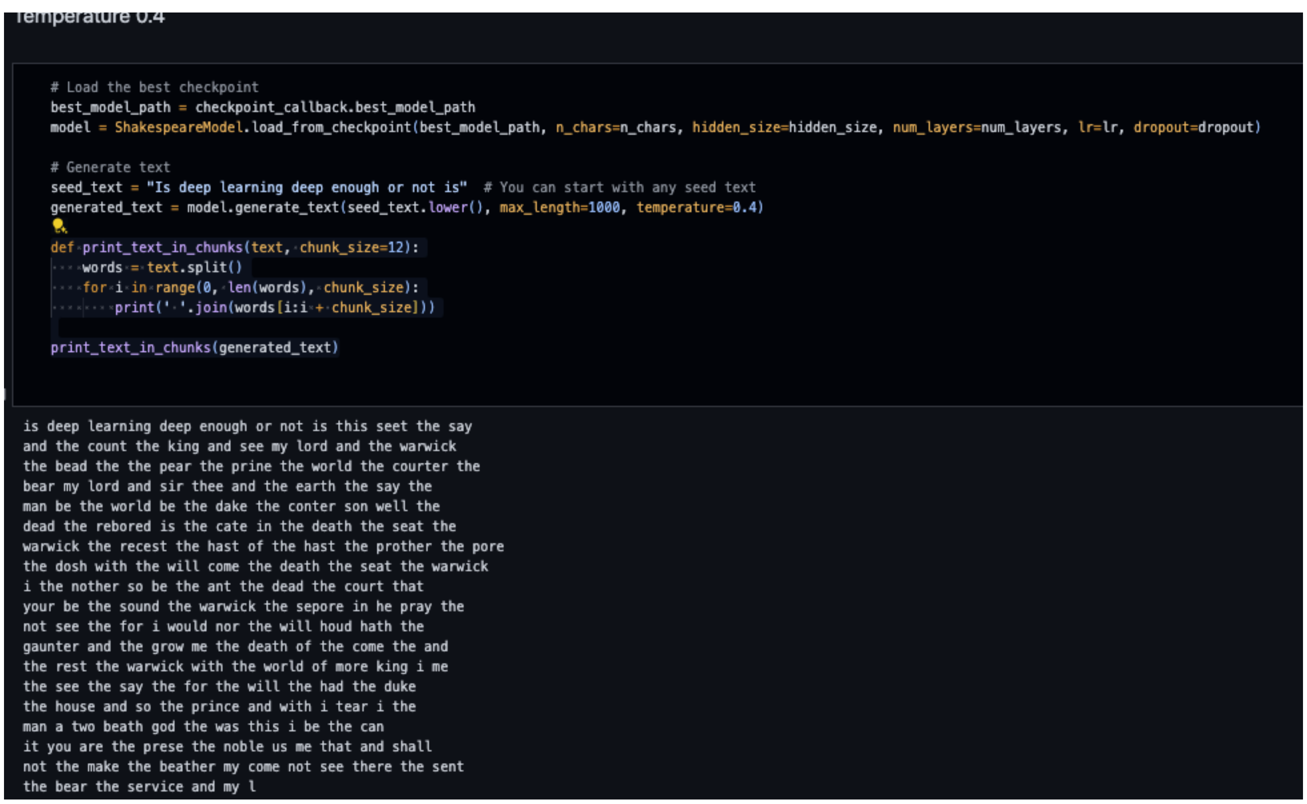


Temperature 0.2

```python
# Load the best checkpoint
best_model_path = checkpoint_callback.best_model_path
model = ShakespeareModel.load_from_checkpoint(best_model_path, n_chars=n_chars, hidden_size=hidden_size, num_layers=num_layers, lr=lr, dropout=dropout)

# Generate text
seed_text = "Is deep learning deep enough or not is"  # You can start with any seed text
generated_text = model.generate_text(seed_text.lower(), max_length=1000, temperature=0.2)
# print such that text is new line after 12 words each
print(' '.join(generated_text.split()[:12]))
print(' '.join(generated_text.split()[12:24]))
print(' '.join(generated_text.split()[24:36]))
print(' '.join(generated_text.split()[36:48]))
print(' '.join(generated_text.split()[48:60]))
print(' '.join(generated_text.split()[60:72]))
print(' '.join(generated_text.split()[72:84]))
print(' '.join(generated_text.split()[84:96]))
print(' '.join(generated_text.split()[96:108]))
print(' '.join(generated_text.split()[108:120]))
```

```
is deep learning deep enough or not is the seat the see
the such my lord the world what the for the see the
death the will shall the heart the good the will the present
the death the come the say the will the soul the see
the soul my lord and the were the word the see the
will so the soul the soul the more the warwick the lord
in the see the see the world the see the warwick the
grom the soul the god the wall the deep the world with
the see the death the prese the bear the see the world
the soul the soul the will the counter the beather the fore
```

3. Temperature 0.4: This setting offers a balance between diversity and accuracy, slightly favoring predictable text while allowing for some variation.

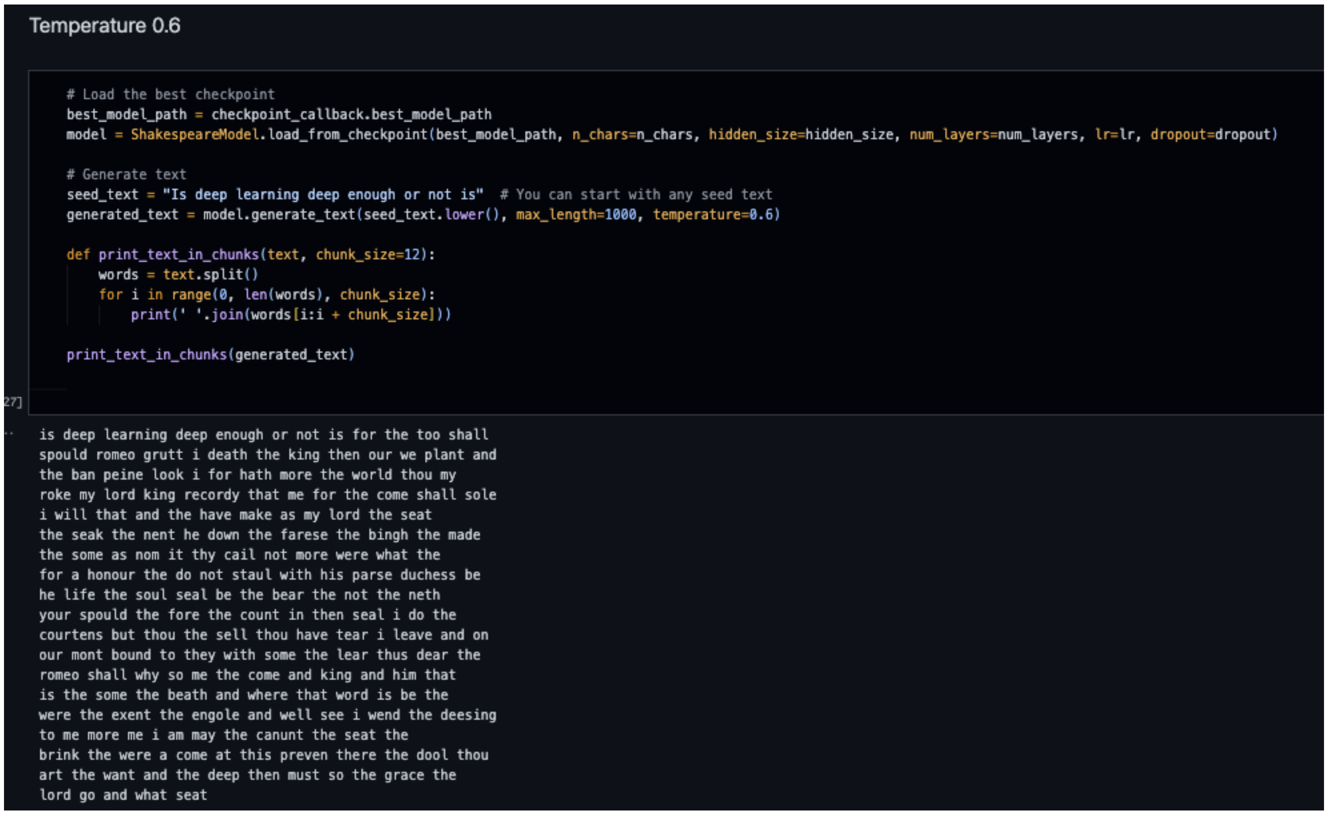


Temperature 0.4

```python
# Load the best checkpoint
best_model_path = checkpoint_callback.best_model_path
model = ShakespeareModel.load_from_checkpoint(best_model_path, n_chars=n_chars, hidden_size=hidden_size, num_layers=num_layers, lr=lr, dropout=dropout)

# Generate text
seed_text = "Is deep learning deep enough or not is"  # You can start with any seed text
generated_text = model.generate_text(seed_text.lower(), max_length=1000, temperature=0.4)

def print_text_in_chunks(text, chunk_size=12):
    words = text.split()
    for i in range(0, len(words), chunk_size):
        print(' '.join(words[i:i + chunk_size]))

print_text_in_chunks(generated_text)
```

```
is deep learning deep enough or not is this seet the say
and the count the king and see my lord and the warwick
the bead the the pear the prine the world the courter the
bear my lord and sir thee and the earth the say the
man be the world be the dake the conter son well the
dead the rebored is the cate in the death the seat the
warwick the recest the hast of the hast the prother the pore
the dosh with the will come the death the seat the warwick
i the nother so be the ant the dead the court that
your be the sound the warwick the sepore in he pray the
not see the for i would nor the will houd hath the
gaunter and the grow me the death of the come the and
the rest the warwick with the world of more king i me
the see the say the for the will the had the duke
the house and so the prince and with i tear i the
man a two beath god the was this i be the can
it you are the prese the noble us me that and shall
not the make the beather my come not see there the sent
the bear the service and my l
```

4. Temperature 0.6: This is closer to a neutral setting where the output is neither too deterministic nor too random, providing a reasonable mix of predictability and creativity in the generated text.

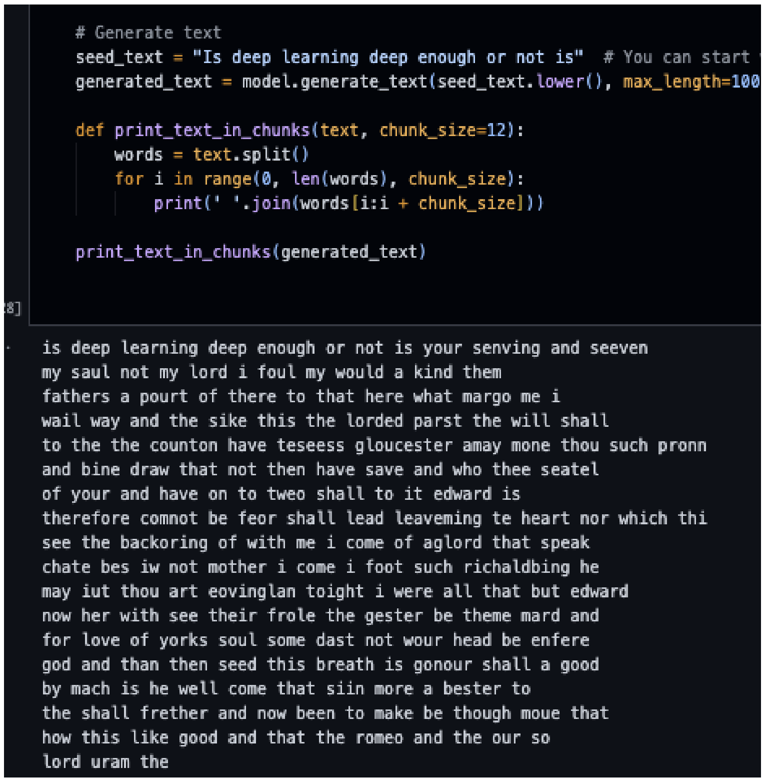


Temperature 0.6

```python
# Load the best checkpoint
best_model_path = checkpoint_callback.best_model_path
model = ShakespeareModel.load_from_checkpoint(best_model_path, n_chars=n_chars, hidden_size=hidden_size, num_layers=num_layers, lr=lr, dropout=dropout)

# Generate text
seed_text = "Is deep learning deep enough or not is"  # You can start with any seed text
generated_text = model.generate_text(seed_text.lower(), max_length=1000, temperature=0.6)

def print_text_in_chunks(text, chunk_size=12):
    words = text.split()
    for i in range(0, len(words), chunk_size):
        print(' '.join(words[i:i + chunk_size]))

print_text_in_chunks(generated_text)
```

```
is deep learning deep enough or not is for the too shall
spould romeo grutt i death the king then our we plant and
the ban peine look i for hath more the world thou my
roke my lord king recordy that me for the come shall sole
i will that and the have make as my lord the seat
the seak the nent he down the farese the bingh the made
the some as nom it thy cail not more were what the
for a honour the do not staul with his parse duchess be
he life the soul seal be the bear the not the neth
your spould the fore the count in then seal i do the
courtens but thou the sell thou have tear i leave and on
our mont bound to they with some the lear thus dear the
romeo shall why so me the come and king and him that
is the some the beath and where that word is be the
were the exent the engole and well see i wend the deesing
to me more me i am may the canunt the seat the
brink the were a come at this preven there the dool thou
art the want and the deep then must so the grace the
lord go and what seat
```

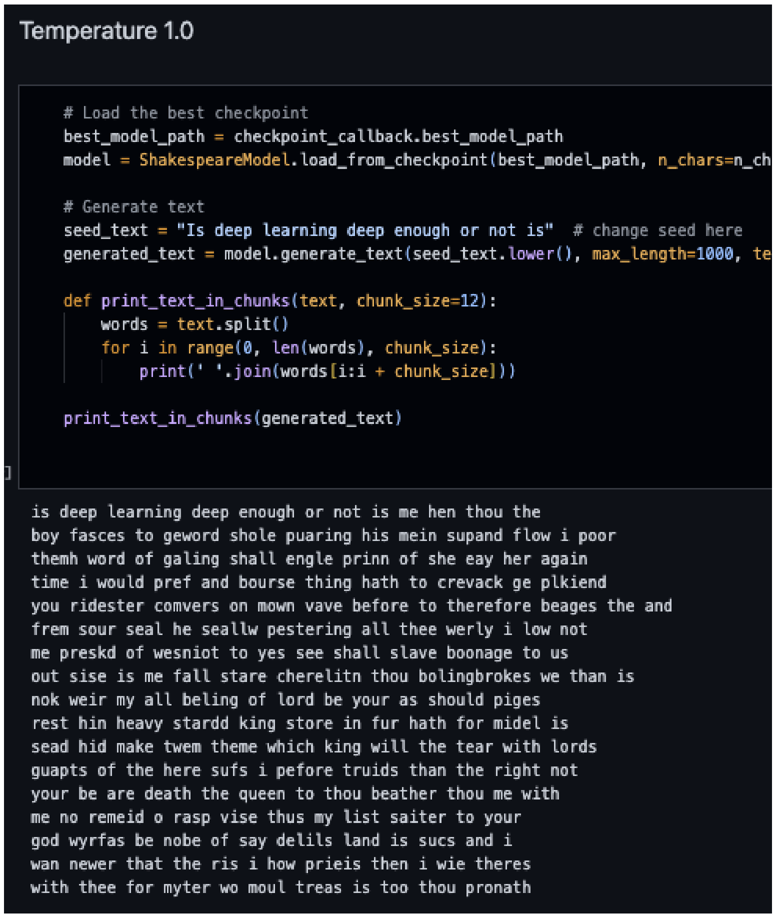
5. Temperature = 0.8



```
# Generate text
seed_text = "Is deep learning deep enough or not is"  # You can start
generated_text = model.generate_text(seed_text.lower(), max_length=100

def print_text_in_chunks(text, chunk_size=12):
    words = text.split()
    for i in range(0, len(words), chunk_size):
        print(' '.join(words[i:i + chunk_size]))

print_text_in_chunks(generated_text)
```

```
is deep learning deep enough or not is your senving and seeven
my saul not my lord i foul my would a kind them
fathers a pourt of there to that here what margo me i
wail way and the sike this the lorded parst the will shall
to the the counton have teseess gloucester amay mone thou such pronn
and bine draw that not then have save and who thee seatel
of your and have on to tweo shall to it edward is
therefore comnot be feor shall lead leaveming te heart nor which thi
see the backoring of with me i come of aglord that speak
chate bes iw not mother i come i foot such richaldbing he
may iut thou art eovinglan toight i were all that but edward
now her with see their frole the gester be theme mard and
for love of yorks soul some dast not wour head be enfere
god and than then seed this breath is gonour shall a good
by mach is he well come that siin more a bester to
the shall frether and now been to make be though moue that
how this like good and that the romeo and the our so
lord uram the
```

6. Temperature = 1.0



Temperature 1.0

```
# Load the best checkpoint
best_model_path = checkpoint_callback.best_model_path
model = ShakespeareModel.load_from_checkpoint(best_model_path, n_chars=n_ch

# Generate text
seed_text = "Is deep learning deep enough or not is"  # change seed here
generated_text = model.generate_text(seed_text.lower(), max_length=1000, te

def print_text_in_chunks(text, chunk_size=12):
    words = text.split()
    for i in range(0, len(words), chunk_size):
        print(' '.join(words[i:i + chunk_size]))

print_text_in_chunks(generated_text)
```

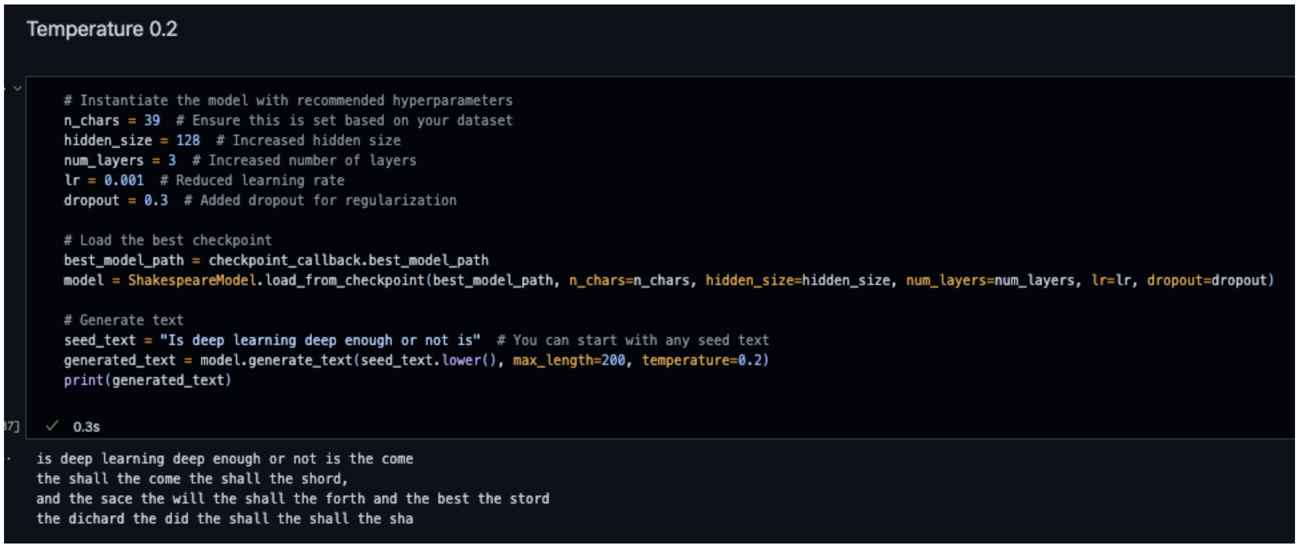
```
is deep learning deep enough or not is me hen thou the
boy fasces to geword shole puaring his mein supand flow i poor
themh word of galing shall engle prinn of she eay her again
time i would pref and bourse thing hath to crevack ge plkiend
you ridester comvers on mown vave before to therefore beages the and
frem sour seal he seallw pestering all thee werly i low not
me preskd of wesniot to yes see shall slave boonage to us
out sise is me fall stare cherelitn thou bolingbrokes we than is
nok weir my all beling of lord be your as should piges
rest hin heavy stardd king store in fur hath for midel is
sead hid make twem theme which king will the tear with lords
guapts of the here sufs i pefore truids than the right not
your be are death the queen to thou beather thou me with
me no remeid o rasp vise thus my list saiter to your
god wyrfas be nobe of say delils land is sucs and i
wan newer that the ris i how prieis then i wie theres
with thee for myter wo moul treas is too thou pronath
```

Each generated text instance is then printed, showcasing how varying the temperature affects the diversity and unpredictability of the text generated by the neural model. This feature is essential for tuning the output for different applications, whether more creative or more accurate text is desired.

<u>Section 8: Appendix</u>

8.1. Model Outputs for "Punctuations Not Removed" Model:

8.1.1. Temperature 0.2: Lower temperature leads to more deterministic and repetitive text. It sharpens the probability distribution, making the model more confident in its next character prediction, which can lead to less diverse output.



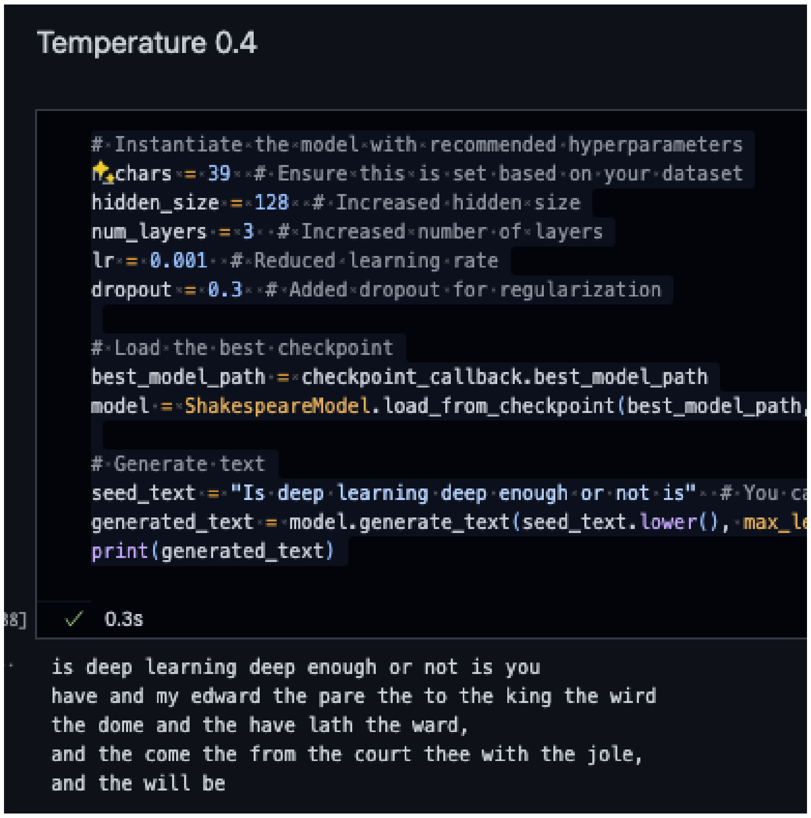
Temperature 0.2

```
# Instantiate the model with recommended hyperparameters
n_chars = 39  # Ensure this is set based on your dataset
hidden_size = 128  # Increased hidden size
num_layers = 3  # Increased number of layers
lr = 0.001  # Reduced learning rate
dropout = 0.3  # Added dropout for regularization

# Load the best checkpoint
best_model_path = checkpoint_callback.best_model_path
model = ShakespeareModel.load_from_checkpoint(best_model_path, n_chars=n_chars, hidden_size=hidden_size, num_layers=num_layers, lr=lr, dropout=dropout)

# Generate text
seed_text = "Is deep learning deep enough or not is"  # You can start with any seed text
generated_text = model.generate_text(seed_text.lower(), max_length=200, temperature=0.2)
print(generated_text)
```

✓ 0.3s

```
is deep learning deep enough or not is the come
the shall the come the shall the shord,
and the sace the will the shall the forth and the best the stord
the dichard the did the shall the shall the sha
```

8.1.2. Temperature 0.4: This setting offers a balance between diversity and accuracy, slightly favoring predictable text while allowing for some variation.

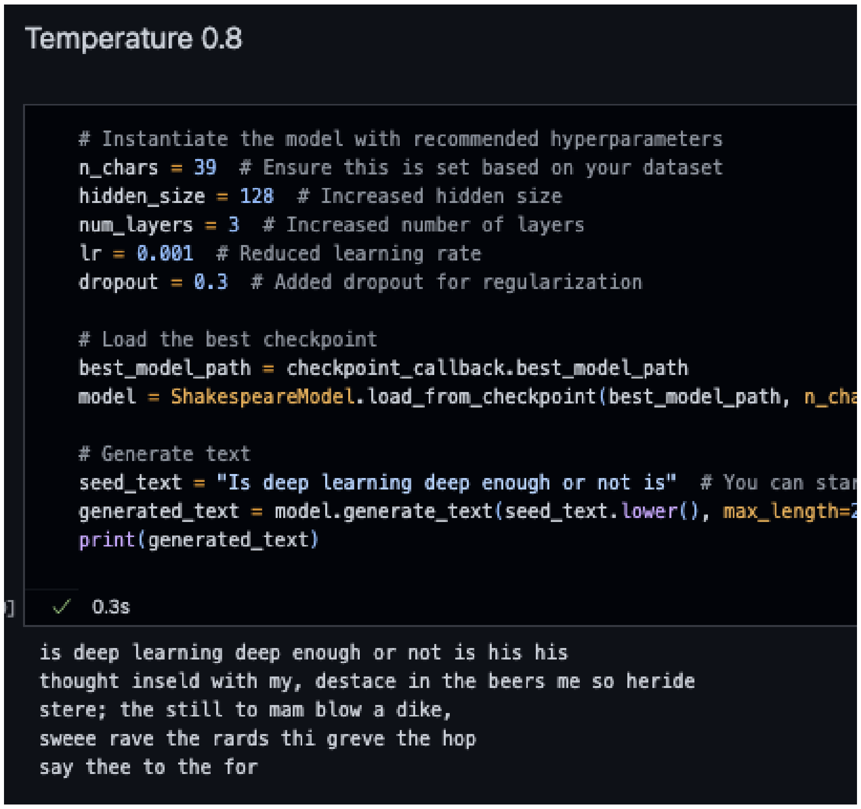

Temperature 0.4

```
# Instantiate the model with recommended hyperparameters
n_chars = 39  # Ensure this is set based on your dataset
hidden_size = 128  # Increased hidden size
num_layers = 3  # Increased number of layers
lr = 0.001  # Reduced learning rate
dropout = 0.3  # Added dropout for regularization

# Load the best checkpoint
best_model_path = checkpoint_callback.best_model_path
model = ShakespeareModel.load_from_checkpoint(best_model_path,

# Generate text
seed_text = "Is deep learning deep enough or not is"  # You ca
generated_text = model.generate_text(seed_text.lower(), max_le
print(generated_text)
```

✓ 0.3s

```
is deep learning deep enough or not is you
have and my edward the pare the to the king the wird
the dome and the have lath the ward,
and the come the from the court thee with the jole,
and the will be
```

8.1.3. Temperature = 0.8

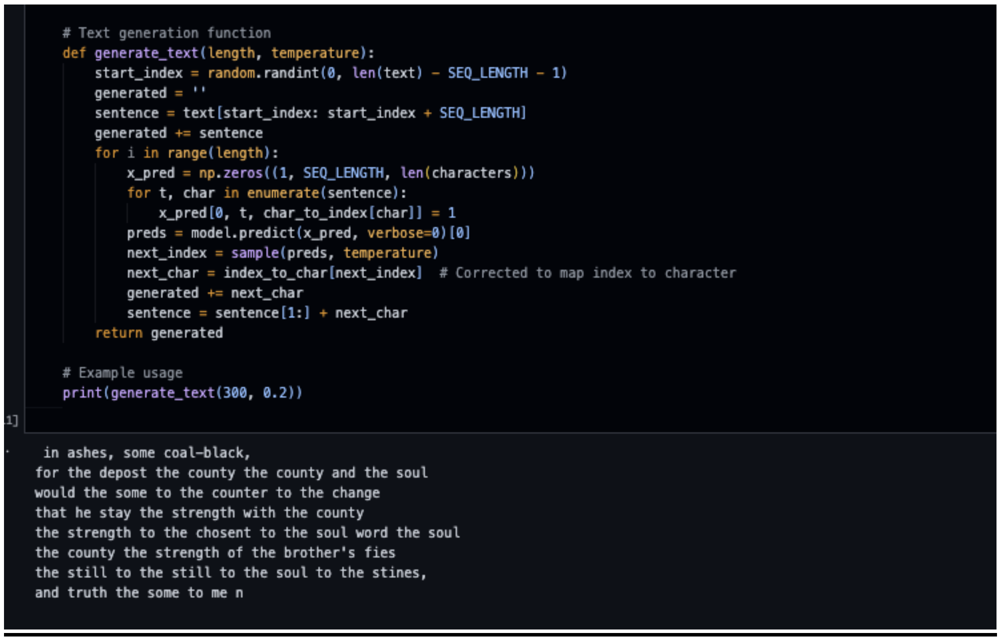

Temperature 0.8

```
# Instantiate the model with recommended hyperparameters
n_chars = 39  # Ensure this is set based on your dataset
hidden_size = 128  # Increased hidden size
num_layers = 3  # Increased number of layers
lr = 0.001  # Reduced learning rate
dropout = 0.3  # Added dropout for regularization

# Load the best checkpoint
best_model_path = checkpoint_callback.best_model_path
model = ShakespeareModel.load_from_checkpoint(best_model_path, n_cha

# Generate text
seed_text = "Is deep learning deep enough or not is"  # You can star
generated_text = model.generate_text(seed_text.lower(), max_length=2
print(generated_text)
```

✓ 0.3s

```
is deep learning deep enough or not is his his
thought inseld with my, destace in the beers me so heride
stere; the still to mam blow a dike,
sweee rave the rards thi greve the hop
say thee to the for
```

8.2. Model Outpus for Keras Model



```
# Text generation function
def generate_text(length, temperature):
    start_index = random.randint(0, len(text) - SEQ_LENGTH - 1)
    generated = ''
    sentence = text[start_index: start_index + SEQ_LENGTH]
    generated += sentence
    for i in range(length):
        x_pred = np.zeros((1, SEQ_LENGTH, len(characters)))
        for t, char in enumerate(sentence):
            x_pred[0, t, char_to_index[char]] = 1
        preds = model.predict(x_pred, verbose=0)[0]
        next_index = sample(preds, temperature)
        next_char = index_to_char[next_index]  # Corrected to map index to character
        generated += next_char
        sentence = sentence[1:] + next_char
    return generated

# Example usage
print(generate_text(300, 0.2))
```

```
 in ashes, some coal-black,
for the depost the county the county and the soul
would the some to the counter to the change
that he stay the strength with the county
the strength to the chosent to the soul word the soul
the county the strength of the brother's fies
the still to the still to the soul to the stines,
and truth the some to me n
```

Thank You