BUAN 6382: Deep Learning

Project 2 Report

Group #2:

Vishala Vemuri (vxv230006)
Megha Nair (mxc220113)
Swapnil Kaner (ssk230013)

The goal of this project is to develop a text generation model using Recurrent Neural Networks (RNNs). You will learn how to preprocess data, build and train an RNN, evaluate the model's performance, and generate new text. The project will be divided into several steps to ensure a structured approach.

1 Basic Setup

1.1 Importing required libraries for text preprocessing, tokenization, model definition, metric calculation, etc

```
[ ]  from google.colab import drive
     from pathlib import Path
     import string
     import numpy as np
     import torch
     from torch.utils.data import Dataset, DataLoader
     import torch.nn as nn
     import torch.optim as optim
     import math
     from collections import Counter
     import matplotlib.pyplot as plt
     from tokenizers import Tokenizer
     from tokenizers.models import BPE
     from tokenizers.trainers import BpeTrainer
     from tokenizers.pre_tokenizers import Whitespace
```

1.1 Mounting Google Drive to store and load models.

```
[ ]  if 'google.colab' in str(get_ipython()):
         drive.mount('/content/drive')
         base_folder = Path('/content/drive/MyDrive/Colab Notebooks/BUAN 6382')
         models_folder = base_folder / "models"  # Subfolder for saving models
         models_folder.mkdir(parents=True, exist_ok=True)  # Create folder if it doesn't exist
```

1.1 Text cleaning and preprocessing function

1.1.1 Converting all text to lowercase
1.1.2 Removing URLs and Metadata
1.1.3 Removing brackets, underscores and carat markers.
1.1.4 Removing punctuations and numbers.
1.1.5 Removing any extra spaces.

```
[ ]  def enhanced_clean_text(text):
         # Step 1: Find the actual start of the content (e.g., "Chapter 1")
         start_marker = "BOOK ONE: 1805"
         start = text.lower().find(start_marker)
         if start != -1:
             text = text[start:]

         # Step 2: Remove URLs and metadata
         text = ' '.join([word for word in text.split() if not word.startswith('http')])

         # Step 3: Remove special formatting markers and transcriber notes
         text = text.replace('_', '')  # Remove underscores
         text = text.replace('^', '')  # Remove carat markers
         text = text.replace('{', '').replace('}', '')  # Remove curly brackets

         # Step 4: Remove punctuation, numbers, and convert to lowercase
         text = text.translate(str.maketrans("", "", string.punctuation))  # Remove punctuation
         text = ''.join([char for char in text if char.isalpha() or char.isspace()])  # Remove numbers
         text = text.lower()  # Convert to lowercase

         # Step 5: Remove extra spaces
         text = ' '.join(text.split())  # Remove redundant spaces

         return text
```

1.1 Loading the Text: “War and Peace”. Cleaning the text.

```
# Load raw text file
with open("War and Peace.txt", "r", encoding="utf-8") as file:
    raw_text = file.read()

# Apply the cleaning function
cleaned_text = enhanced_clean_text(raw_text)

# Output the first 500 characters of the cleaned text
print(cleaned_text[:500])
```

```
war and peace by leo tolstoytolstoi contents book one chapter i chapter ii chapter iii chapter iv ch
```

2 Tokenization and Dataset Creation

2.1 Character level tokenization

2.1.1 Creating Character to integer mapping and Integer to character mapping.

2.1.2 Transforming all the characters in our text using integer mapping.

2.1.3 Creating input-output pairs from the data. Input is a sequence of characters and output is target character to be predicted by the model.

```
[ ]  # Tokenize the text at character level
     chars = sorted(set(cleaned_text))  # Unique characters
     char_to_index = {char: idx for idx, char in enumerate(chars)}  # Character to integer mapping
     index_to_char = {idx: char for idx, char in enumerate(chars)}  # Integer to character mapping

     # Convert text into integers
     text_as_int = np.array([char_to_index[char] for char in cleaned_text])

     # Define sequence length and prepare input-output pairs
     sequence_length = 200
     sequences = []
     targets = []

     for i in range(len(text_as_int) - sequence_length):
         sequences.append(text_as_int[i:i+sequence_length])
         targets.append(text_as_int[i+sequence_length])

     sequences = np.array(sequences)
     targets = np.array(targets)
```

2.2 Custom dataset class to store the input sequence and output target in the same object.

```
[ ]  # Custom dataset class
     class TextDataset(Dataset):
         def __init__(self, sequences, targets):
             self.sequences = torch.tensor(sequences, dtype=torch.long)
             self.targets = torch.tensor(targets, dtype=torch.long)

         def __len__(self):
             return len(self.sequences)

         def __getitem__(self, idx):
             return self.sequences[idx], self.targets[idx]
```

2.3 Splitting the data into Train and Validation datasets.

```
     # Split data into training and validation sets
     train_size = int(0.9 * len(sequences))
     val_size = len(sequences) - train_size

     train_dataset = TextDataset(sequences[:train_size], targets[:train_size])
     val_dataset = TextDataset(sequences[train_size:], targets[train_size:])
```

3 Train_model function. This function will be used to train, monitor and visualize the training process.

```
[ ] def train_model(model, train_loader, val_loader, criterion, optimizer, num_epochs, device):
        """
        Train the given model and visualize training and validation loss over epochs.

        Args:
            model: The PyTorch model to train.
            train_loader: DataLoader for training data.
            val_loader: DataLoader for validation data.
            criterion: Loss function.
            optimizer: Optimizer for training.
            num_epochs: Number of training epochs.
            device: Device to train on (e.g., "cpu" or "cuda").

        Returns:
            model: Trained model.
            train_losses: List of training losses for each epoch.
            val_losses: List of validation losses for each epoch.
        """
        # Move model to the device
        model.to(device)

        # Calculate the number of trainable parameters
        total_params = sum(p.numel() for p in model.parameters() if p.requires_grad)
        print(f"Total number of trainable parameters in the model: {total_params}")

        # Lists to store losses
        train_losses = []
        val_losses = []

        # Training loop
        for epoch in range(num_epochs):
            model.train()
            train_loss = 0
            for inputs, targets in train_loader:
                inputs, targets = inputs.to(device), targets.to(device)

                # Forward pass
                outputs, _ = model(inputs)
                loss = criterion(outputs, targets)
```

```
[ ]             # Backward pass
                optimizer.zero_grad()
                loss.backward()
                optimizer.step()

                train_loss += loss.item()

            # Validation loop
            model.eval()
            val_loss = 0
            with torch.no_grad():
                for inputs, targets in val_loader:
                    inputs, targets = inputs.to(device), targets.to(device)
                    outputs, _ = model(inputs)
                    loss = criterion(outputs, targets)
                    val_loss += loss.item()
```

```
            # Calculate average losses for the epoch
            train_loss /= len(train_loader)
            val_loss /= len(val_loader)

            # Store losses
            train_losses.append(train_loss)
            val_losses.append(val_loss)

            print(f"Epoch {epoch+1}/{num_epochs}, Train Loss: {train_loss:.4f}, Val Loss: {val_loss:.4f}")

        # Save the final trained model
        model_path = "lstm_model.pth"
        torch.save(model.state_dict(), model_path)
        print(f"Model saved to {model_path}")

        # Visualization
        plt.figure(figsize=(10, 6))
        plt.plot(range(1, num_epochs + 1), train_losses, label='Train Loss')
        plt.plot(range(1, num_epochs + 1), val_losses, label='Validation Loss')
        plt.xlabel('Epochs')
        plt.ylabel('Loss')
        plt.ylim(0)  # Start y-axis from 0
        plt.title('Training and Validation Loss over Epochs')
        plt.legend()
        plt.grid(True)
        plt.show()

        return model, train_losses, val_losses
```

4 Generate_text function. This function will help us to generate text from the model, one token at a time.

```
def generate_text(model, seed_text, length):
    model.eval()  # Set the model to evaluation mode
    generated_text = seed_text
    hidden = None

    for _ in range(length):
        # Convert seed text to integers
        input_seq = torch.tensor([char_to_index[char] for char in seed_text[-sequence_length:]], dtype=torch.long).

        # Get predictions
        with torch.no_grad():
            output, hidden = model(input_seq, hidden)  # Forward pass
            output = output.squeeze(0)  # Remove batch dimension
            next_char_index = torch.argmax(output).item()  # Select the most probable character deterministically

        # Append the predicted character
        next_char = index_to_char[next_char_index]
        generated_text += next_char
        seed_text += next_char

    return generated_text
```

5 Load_model function. Used to load saved models from google drive.

```
[ ]  # Load the saved model and generate text
     def load_model(model_class, model_path, vocab_size, embed_size, hidden_size, num_layers, dropout_rate, device):
         """
         Load the saved model state dictionary into a model instance.

         Args:
             model_class: The class definition of the model.
             model_path: Path to the saved model file.
             vocab_size: Size of the vocabulary.
             embed_size: Size of the embedding layer.
             hidden_size: Size of the hidden state.
             num_layers: Number of LSTM layers.
             dropout_rate: Dropout rate.
             device: Device to load the model on ("cpu" or "cuda").

         Returns:
             model: Loaded model ready for inference.
         """
         # Instantiate the model
         model = model_class(vocab_size, embed_size, hidden_size, num_layers, dropout_rate)
         model.load_state_dict(torch.load(model_path, map_location=device))  # Load state dictionary
         model.to(device)  # Move to device
         model.eval()  # Set to evaluation mode
         print(f"Model loaded from {model_path}")
         return model
```

6 Calculate_perplexity function. Calculates perplexity metric for the model based on a particular dataset. Lower perplexity for a model is better.

```
[ ]  def calculate_perplexity(model, data_loader):
         model.eval()  # Set the model to evaluation mode
         total_loss = 0
         total_count = 0
         criterion = nn.CrossEntropyLoss()  # Loss function used for training

         with torch.no_grad():
             for inputs, targets in data_loader:
                 inputs, targets = inputs.to(device), targets.to(device)
                 outputs, _ = model(inputs)  # Get model predictions
                 loss = criterion(outputs, targets)  # Compute loss
                 total_loss += loss.item() * inputs.size(0)  # Accumulate loss
                 total_count += inputs.size(0)  # Count tokens

         # Calculate perplexity
         average_loss = total_loss / total_count
         perplexity = math.exp(average_loss)
         return perplexity
```

7 Calculate_entropy function. Calculates entropy for the generated text. More entropy score suggests a diverse text from the model.

```
[ ] def calculate_entropy(generated_text):
        # Count the frequency of each character
        char_counts = Counter(generated_text)
        total_chars = sum(char_counts.values())

        # Compute entropy
        entropy = -sum((count / total_chars) * math.log2(count / total_chars) for count in char_counts.values())
        return entropy
```

```
[ ] device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    print(device)
```

8 Model Training: Section 1

8.1 In this section we will test different model architectures and sizes for a sequence length of 200 (Sequences are created in section 2.1).

8.2 A sequence length of 200 signifies that each next character will be predicted using the preceding 200 characters in the text.

8.3 We are starting with a 200 sequence length so as to capture long term dependencies in the text. This may help the model generate more coherent and context rich text.

```
[ ] # Define the LSTM model
    class LSTMModel(nn.Module):
        def __init__(self, vocab_size, embed_size, hidden_size, num_layers=1, dropout=0):
            super(LSTMModel, self).__init__()
            self.embedding = nn.Embedding(vocab_size, embed_size)
            self.lstm = nn.LSTM(
                embed_size,
                hidden_size,
                num_layers=num_layers,
                batch_first=True,
                dropout=dropout
            )
            self.dropout = nn.Dropout(dropout)  # Dropout after LSTM output
            self.fc = nn.Linear(hidden_size, vocab_size)

        def forward(self, x, hidden=None):
            x = self.embedding(x)  # Convert input to embeddings
            output, hidden = self.lstm(x, hidden)  # Pass through LSTM layers
            output = self.dropout(output)  # Apply dropout to LSTM outputs
            output = self.fc(output[:, -1, :])  # Use the last output for prediction
            return output, hidden
```

8.4 Layers of the model:

8.4.1 **Embedding Layer**: Converts token indices to dense vectors, capturing semantic relationships between tokens. It reduces the dimensionality of input features and provides a better representation of words for downstream layers to process, making it essential for handling text data.

8.4.2 **LSTM Layer**: Processes sequential data, capturing long-term dependencies and patterns. LSTMs are well-suited for handling sequences, such as text, because they capture long-term dependencies and can provide coherent text generation.
8.4.3 **Dropout Layer**: Prevents overfitting by randomly zeroing out a fraction of inputs during training. Dropout is applied after the LSTM layer to ensure that the model does not overfit to the training data, especially when working with large, complex architectures or datasets.
8.4.4 **Linear Layer**: Maps the LSTM output to the target vocabulary size for predictions. After processing the sequence through the LSTM, the last hidden state is passed to the linear layer to make predictions, such as the next word in a sequence or the class label in a classification task.

8.5 For the BASIC MODEL0, we will start with a simple model with a single LSTM layer with hidden size of 128. Batch size is taken as 1024 and embedding for each character is set to 128.

```
[ ]  train_loader = DataLoader(train_dataset, batch_size=1024, shuffle=True)
     val_loader = DataLoader(val_dataset, batch_size=1024)
```

Basic Model0:

```
Single layer LSTM.
Training Sequence Length 200
Hidden size 128
Character level Embedding size 128
Batch Size 1024
```

8.6 A generation task is basically a multiclass classification task with classes equal to the length of the vocabulary. In a character level generation model, we should have approximately 30 output classes.

8.6.1 Therefore, CrossEntropyLoss is suitable for our multi-class classification problem. It combines LogSoftmax and NLLLoss, making it ideal for calculating the difference between the predicted probabilities and the true next tokens.

```
# Define model parameters
vocab_size = len(chars)
embed_size = 128
hidden_size = 128

model = LSTMModel(vocab_size, embed_size, hidden_size)

# Define loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

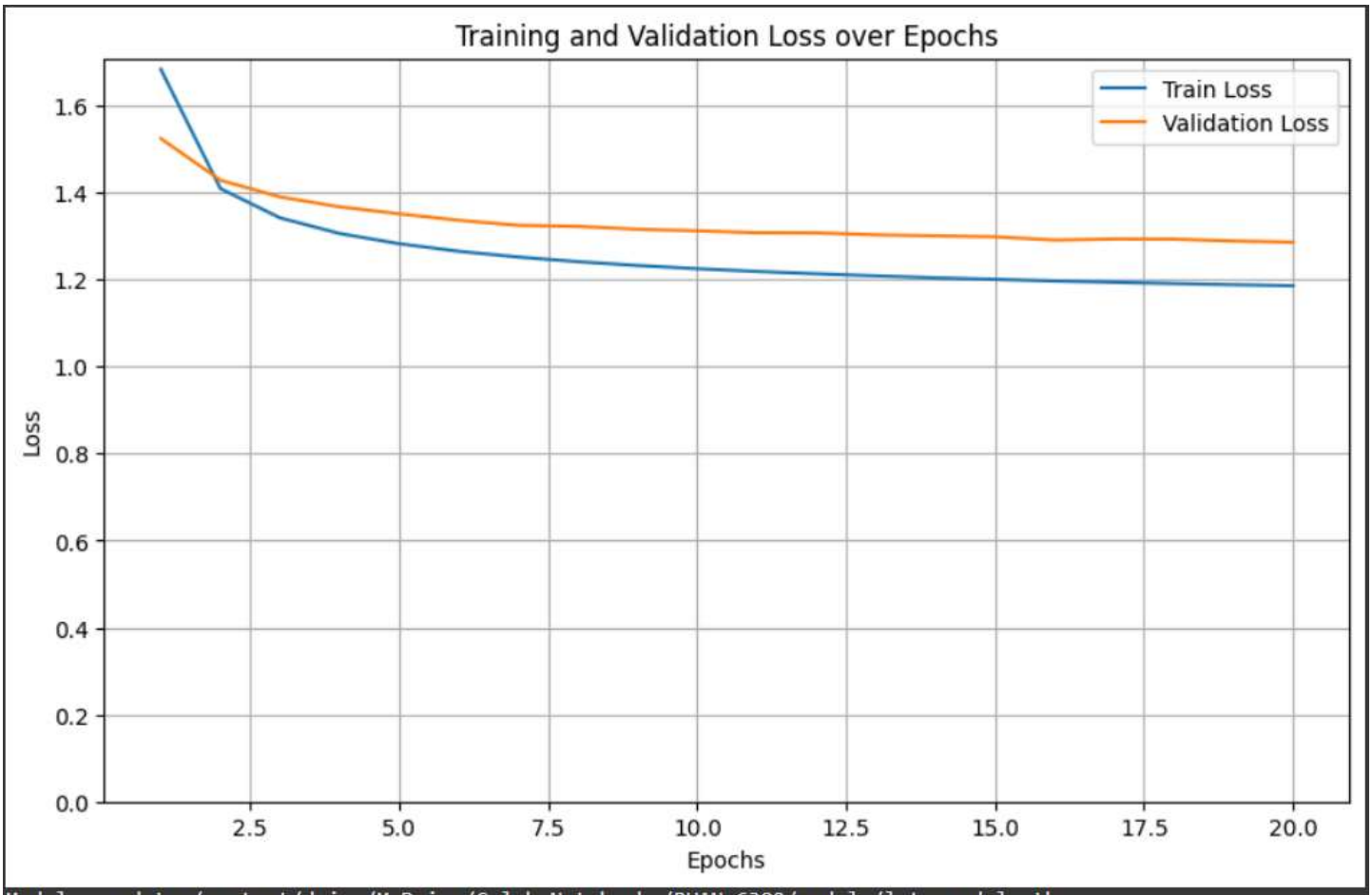
8.7 We will be using Adam optimizer for our LSTM based generation model. Adam optimizer adapts the learning rate for each parameter, providing efficient and stable training, especially for our LSTM model dealing with sequential text data, by combining the advantages of both SGD and RMSProp.

8.8 In this basic model we will have 20 epochs and observe the results. Based on these observations, we will choose an appropriate number of epochs for further models.

```
# Train the model
trained_model, train_losses, val_losses = train_model(
    model, train_loader, val_loader, criterion, optimizer, num_epochs=20, device=device
)

# Save the model to Google Drive
model_save_path = models_folder / "lstm_model0.pth"
torch.save(trained_model.state_dict(), model_save_path)
print(f"Model saved to {model_save_path}")
```

8.9 The model is saved at the end of training in order to reuse it later. The load_model function is used to load the model into memory.

```
Total number of trainable parameters in the model: 144175
Epoch 1/20, Train Loss: 1.6830, Val Loss: 1.5240
Epoch 2/20, Train Loss: 1.4088, Val Loss: 1.4280
Epoch 3/20, Train Loss: 1.3416, Val Loss: 1.3897
Epoch 4/20, Train Loss: 1.3056, Val Loss: 1.3668
Epoch 5/20, Train Loss: 1.2821, Val Loss: 1.3509
Epoch 6/20, Train Loss: 1.2647, Val Loss: 1.3360
Epoch 7/20, Train Loss: 1.2515, Val Loss: 1.3243
Epoch 8/20, Train Loss: 1.2409, Val Loss: 1.3220
Epoch 9/20, Train Loss: 1.2321, Val Loss: 1.3153
Epoch 10/20, Train Loss: 1.2247, Val Loss: 1.3117
Epoch 11/20, Train Loss: 1.2185, Val Loss: 1.3074
Epoch 12/20, Train Loss: 1.2130, Val Loss: 1.3070
Epoch 13/20, Train Loss: 1.2082, Val Loss: 1.3025
Epoch 14/20, Train Loss: 1.2038, Val Loss: 1.3002
Epoch 15/20, Train Loss: 1.2002, Val Loss: 1.2980
Epoch 16/20, Train Loss: 1.1966, Val Loss: 1.2907
Epoch 17/20, Train Loss: 1.1935, Val Loss: 1.2931
Epoch 18/20, Train Loss: 1.1906, Val Loss: 1.2927
Epoch 19/20, Train Loss: 1.1880, Val Loss: 1.2884
Epoch 20/20, Train Loss: 1.1856, Val Loss: 1.2856
Model saved to lstm_model.pth
```




```
Model saved to /content/drive/MyDrive/Colab Notebooks/BUAN 6382/models/lstm_model.pth
```

8.10 We see that training and validation loss are decreasing with each epoch. Further training this model may lower the loss further.

8.11 We also observe that the train loss is higher than the validation loss at the end of the first epoch. This must be due to the fact that in the start of the first epoch, the model is initialized with random weights and as the training progresses, with each batch, the weights are updated. This leads to poor performance on the initial training batches thus increasing the overall training loss. By the time the validation dataset is evaluated, the model is trained to a certain degree and hence we see a lower validation loss.

```
[ ]  # Define model parameters
     model_path = models_folder / "lstm_model0.pth"
     vocab_size = len(chars)
     embed_size = 128
     hidden_size = 128
     num_layers = 1
     dropout_rate = 0

     model0 = load_model(LSTMModel, model_path, vocab_size, embed_size, hidden_size, num_layers, dropout_rate, device)
```

```
Model loaded from /content/drive/MyDrive/Colab Notebooks/BUAN 6382/models/lstm_model0.pth
```

8.12 Generation of sample text from the BASIC MODEL0.

```
[ ]  seed_text = "The soldiers marched forward and at a".lower()  # Convert to lowercase
     generated_text = generate_text(model0, seed_text, length=200)
     print(generated_text)
```

```
the soldiers marched forward and at a standing and the soldiers and the soldiers and the soldiers and
```

```
the soldiers and the soldiers and the soldiers and the soldiers and the soldiers and the soldiers and the
```

8.13 We see that the model is repeating the same phrase ‘and the soldiers’ again and again. This model was a very small model with only 144,175 trainable parameters.

```
[ ]  # Evaluate perplexity on the validation set
     val_perplexity = calculate_perplexity(model0, val_loader)
     print(f"Validation Perplexity: {val_perplexity:.2f}")

     entropy = calculate_entropy(generated_text)
     print(f"Entropy of Generated Text: {entropy:.2f}")
```

```
Validation Perplexity: 3.61
Entropy of Generated Text: 3.59
```

8.14 Perplexity for the BASIC MODEL0 on validation dataset is 3.61 and entropy of the generated text is 3.59. These will be our baseline. We will try to improve upon them with modifications to the BASIC MODEL0.

Model1:

8.15 Next, we will increase the hidden size of the model from 128 to 512 while keeping all the rest of the model parameters, optimizer, learning rate and loss function the same as BASIC MODEL0. This new model will be Model1.

```
Single layer LSTM.
Training Sequence Length 200
Hidden size 512
Character level Embedding size 128
Batch Size 2048
```

8.16 To speed up training and better utilized our GPU, we will increase the batch size to 2048.

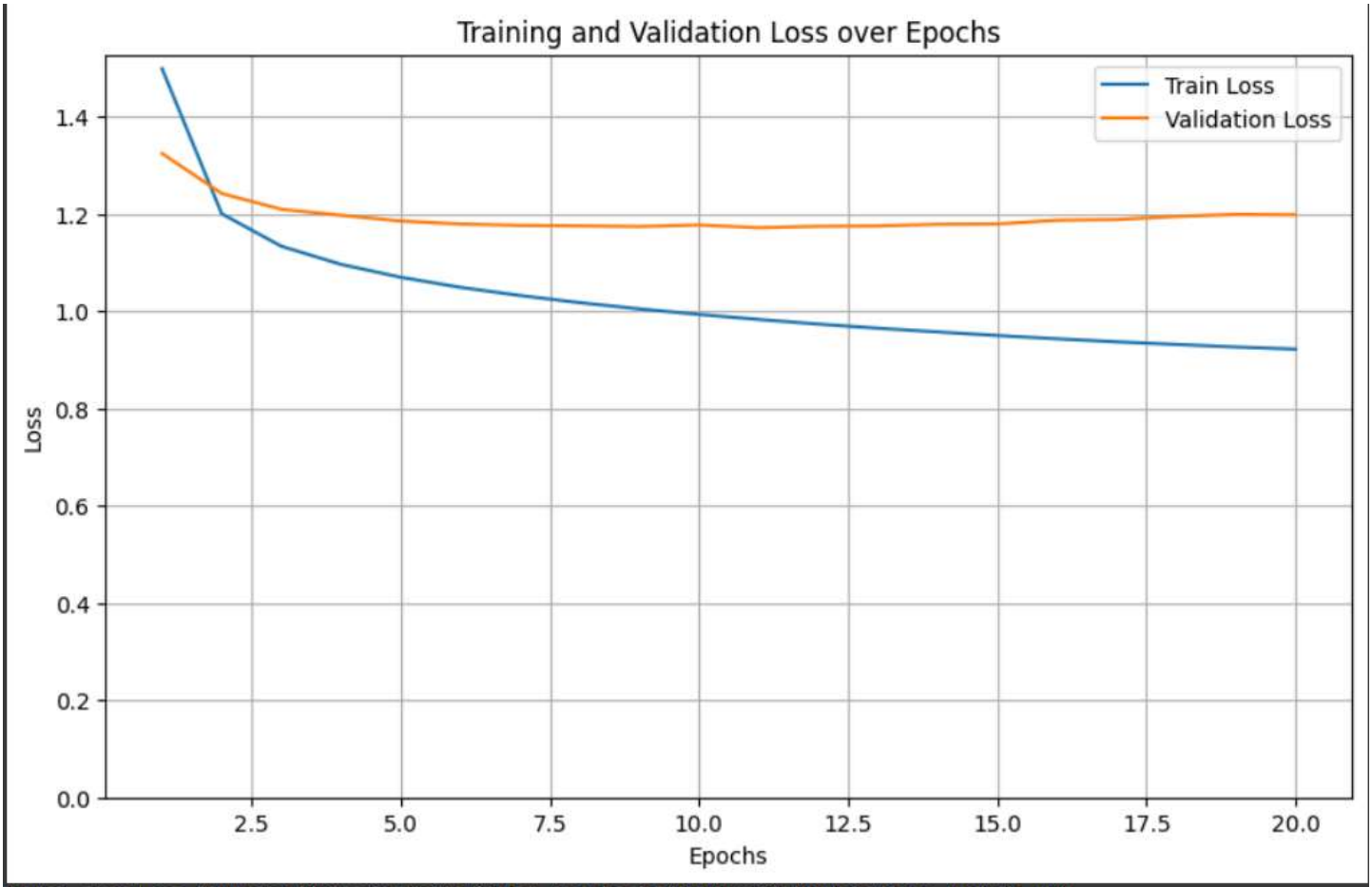
```
[ ]  train_loader = DataLoader(train_dataset, batch_size=2048, shuffle=True)
     val_loader = DataLoader(val_dataset, batch_size=2048)
```

```python
# Define model parameters
vocab_size = len(chars)
embed_size = 128
hidden_size = 512

model1 = LSTMModel(vocab_size, embed_size, hidden_size)

# Define loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer1 = optim.Adam(model1.parameters(), lr=0.001)
```

```python
# Train the model
trained_model1, train_losses, val_losses = train_model(
    model1, train_loader, val_loader, criterion, optimizer1, num_epochs=20, device=device
)

# Save the model to Google Drive
model_save_path = models_folder / "lstm_model1.pth"
torch.save(trained_model1.state_dict(), model_save_path)
print(f"Model saved to {model_save_path}")
```

8.17 Total trainable model parameters are increased to 1,344,943.

```
Total number of trainable parameters in the model: 1344943
Epoch 1/20, Train Loss: 1.4988, Val Loss: 1.3246
Epoch 2/20, Train Loss: 1.2005, Val Loss: 1.2423
Epoch 3/20, Train Loss: 1.1338, Val Loss: 1.2098
Epoch 4/20, Train Loss: 1.0965, Val Loss: 1.1975
Epoch 5/20, Train Loss: 1.0700, Val Loss: 1.1857
Epoch 6/20, Train Loss: 1.0493, Val Loss: 1.1795
Epoch 7/20, Train Loss: 1.0325, Val Loss: 1.1767
Epoch 8/20, Train Loss: 1.0178, Val Loss: 1.1758
Epoch 9/20, Train Loss: 1.0050, Val Loss: 1.1744
Epoch 10/20, Train Loss: 0.9935, Val Loss: 1.1776
Epoch 11/20, Train Loss: 0.9833, Val Loss: 1.1719
Epoch 12/20, Train Loss: 0.9737, Val Loss: 1.1748
Epoch 13/20, Train Loss: 0.9652, Val Loss: 1.1757
Epoch 14/20, Train Loss: 0.9576, Val Loss: 1.1789
Epoch 15/20, Train Loss: 0.9503, Val Loss: 1.1795
Epoch 16/20, Train Loss: 0.9434, Val Loss: 1.1873
Epoch 17/20, Train Loss: 0.9373, Val Loss: 1.1887
Epoch 18/20, Train Loss: 0.9318, Val Loss: 1.1951
Epoch 19/20, Train Loss: 0.9266, Val Loss: 1.1994
Epoch 20/20, Train Loss: 0.9222, Val Loss: 1.1985
Model saved to lstm_model.pth
```




```
Model saved to /content/drive/MyDrive/Colab Notebooks/BUAN 6382/models/lstm_model1.pth
```

8.18 We see that training loss is decreasing with each epoch, whereas the validation loss initially decreases and then starts to rise from the 11th epoch, suggesting overfitting.

8.19 We also observe that the train loss is higher than the validation loss at the end of the first epoch. This must be due to the fact that in the start of the first epoch, the model is initialized with random weights and as the training progresses, with each batch, the weights are updated. This leads to poor performance on the initial training batches thus increasing the overall training loss. By the time the validation dataset is evaluated, the model is trained to a certain degree and hence we see a lower validation loss.

```
[ ]  # Define model parameters
     model_path = models_folder / "lstm_model1.pth"
     vocab_size = len(chars)
     embed_size = 128
     hidden_size = 512
     num_layers = 1
     dropout_rate = 0

     model1 = load_model(LSTMModel, model_path, vocab_size, embed_size, hidden_size, num_layers, dropout_rate, device)
```

```
Model loaded from /content/drive/MyDrive/Colab Notebooks/BUAN 6382/models/lstm_model1.pth
```

8.20 The model is saved at the end of training in order to reuse it later. The load_model function is used to load the model into memory.

8.21 Generation of sample text from the MODEL1.

```
[ ]  seed_text = "The soldiers marched forward and at a".lower()  # Convert to lowercase
     generated_text = generate_text(model1, seed_text, length=200)
     print(generated_text)
```

```
the soldiers marched forward and at a gallop and the same step off the french army was a smile of
service the countess had been reading the strange look of sixteen and serious and in the same way to the counts house
```

8.22 We see that the model has performed much better than BASIC MODEL0. The model has learned to spell the words correctly, has proper spacing between words, uses a great variety of words in the generation, although does not make much sense.

```
[ ]  # Evaluate perplexity on the validation set
     val_perplexity = calculate_perplexity(model1, val_loader)
     print(f"Validation Perplexity: {val_perplexity:.2f}")

     entropy = calculate_entropy(generated_text)
     print(f"Entropy of Generated Text: {entropy:.2f}")
```

```
Validation Perplexity: 3.31
Entropy of Generated Text: 3.96
```

8.23 Enhaced performance over the BASIC MODEL0 is also evident from the perplexity and entropy score as well. The perplexity has reduced from 3.61 to 3.31 while the entropy rises from 3.59 to 3.96.

9 Model Training: Section 2

9.1 In this section we will try a reduced sequence length of 100 for all the architectures in the previous section.

9.2 The main idea here is to test our hypothesis that an LSTM model is able to capture long-term dependencies over 200 time steps.

9.3 We will be modifying the BASIC MODEL0 by training the model using a sequence of 100 characters only. This model will be MODEL0_1.

Model0_1:

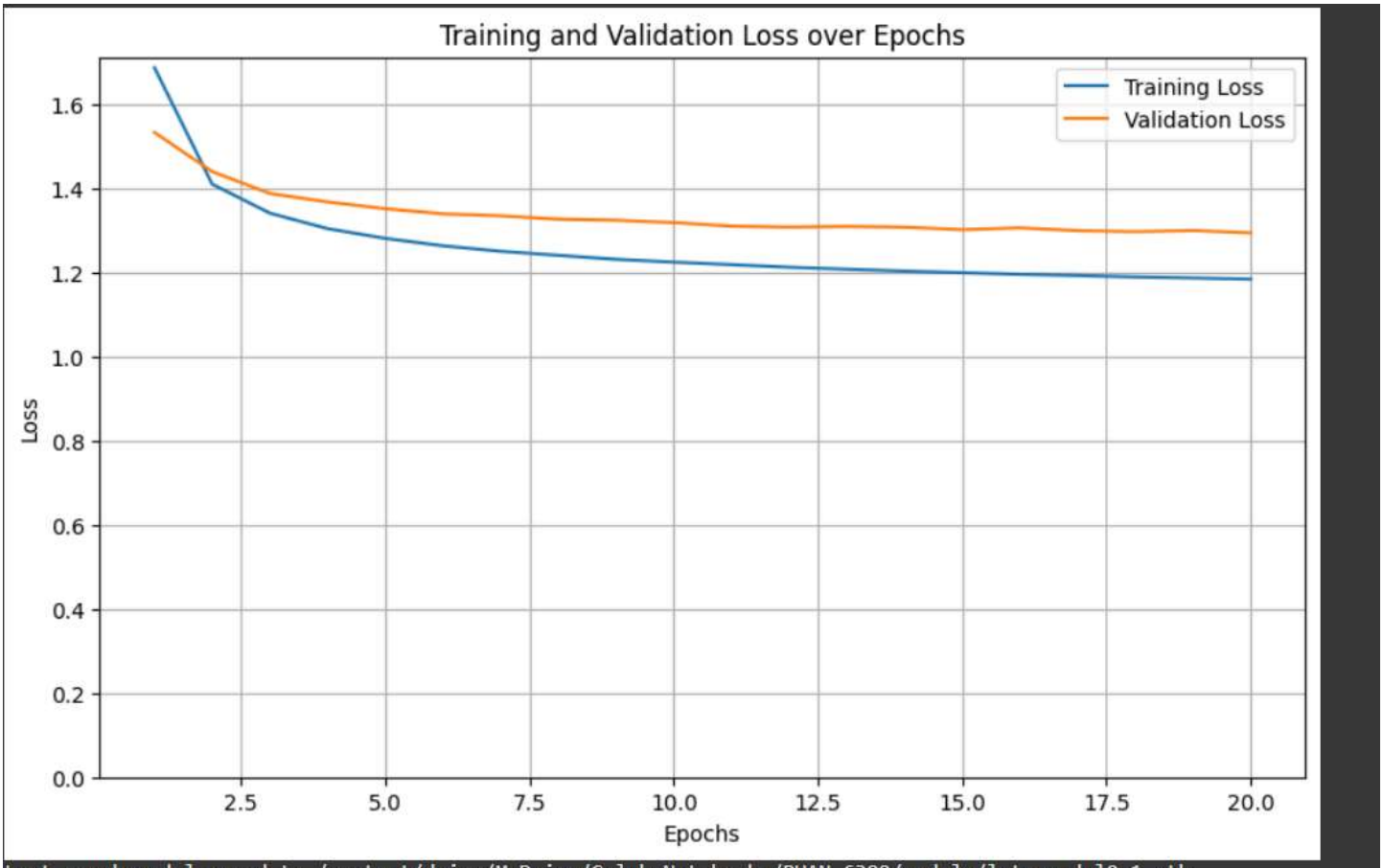
```
Single layer LSTM.
Training Sequence Length 100
Hidden size 128
Character level Embedding size 128
Batch Size 1024
```

9.4 Rest all the model parameters, optimizer, learning rate and loss function the same as BASIC MODEL0.

```python
# Define model parameters
vocab_size = len(chars)
embed_size = 128
hidden_size = 128

model0_1 = LSTMModel(vocab_size, embed_size, hidden_size)

# Define loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer0_1 = optim.Adam(model0_1.parameters(), lr=0.001)
```

```python
# Train the model
trained_model0_1, best_model_state_dict0_1, train_losses, val_losses = train_model(
    model0_1, train_loader, val_loader, criterion, optimizer0_1, num_epochs=20, device=device
)

# Save the last epoch model
last_epoch_model_path = models_folder / "lstm_model0_1.pth"
torch.save(trained_model0_1.state_dict(), last_epoch_model_path)
print(f"Last epoch model saved to {last_epoch_model_path}")

# Save the best model based on validation loss
best_model_path = models_folder / "lstm_model0_1_best.pth"
torch.save(best_model_state_dict0_1, best_model_path)
print(f"Best validation loss model saved to {best_model_path}")
```

```
Total number of trainable parameters in the model: 144175
Epoch 1/20, Train Loss: 1.6869, Val Loss: 1.5330
Epoch 2/20, Train Loss: 1.4100, Val Loss: 1.4399
Epoch 3/20, Train Loss: 1.3411, Val Loss: 1.3879
Epoch 4/20, Train Loss: 1.3043, Val Loss: 1.3678
Epoch 5/20, Train Loss: 1.2811, Val Loss: 1.3517
Epoch 6/20, Train Loss: 1.2635, Val Loss: 1.3395
Epoch 7/20, Train Loss: 1.2504, Val Loss: 1.3347
Epoch 8/20, Train Loss: 1.2408, Val Loss: 1.3268
Epoch 9/20, Train Loss: 1.2313, Val Loss: 1.3242
Epoch 10/20, Train Loss: 1.2246, Val Loss: 1.3187
Epoch 11/20, Train Loss: 1.2187, Val Loss: 1.3103
Epoch 12/20, Train Loss: 1.2125, Val Loss: 1.3079
Epoch 13/20, Train Loss: 1.2078, Val Loss: 1.3096
Epoch 14/20, Train Loss: 1.2033, Val Loss: 1.3078
Epoch 15/20, Train Loss: 1.1995, Val Loss: 1.3019
Epoch 16/20, Train Loss: 1.1957, Val Loss: 1.3061
Epoch 17/20, Train Loss: 1.1927, Val Loss: 1.2995
Epoch 18/20, Train Loss: 1.1894, Val Loss: 1.2971
Epoch 19/20, Train Loss: 1.1869, Val Loss: 1.2997
Epoch 20/20, Train Loss: 1.1842, Val Loss: 1.2943
```




```
Last epoch model saved to /content/drive/MyDrive/Colab Notebooks/BUAN 6382/models/lstm_model0_1.pth
Best validation loss model saved to /content/drive/MyDrive/Colab Notebooks/BUAN 6382/models/lstm_model0_1_best.pth
```

9.5 We see that training and validation loss are decreasing with each epoch. Further training this model may lower the loss further.

9.6 The model is saved at the end of training in order to reuse it later. The load_model function is used to load the model into memory.

```
[ ]  # Define model parameters
     model_path = models_folder / "lstm_model0_1.pth"
     vocab_size = len(chars)
     embed_size = 128
     hidden_size = 128
     num_layers = 1
     dropout_rate = 0

     model0_1 = load_model(LSTMModel, model_path, vocab_size, embed_size, hidden_size, num_layers, dropout_rate, device)
```

```
Model loaded from /content/drive/MyDrive/Colab Notebooks/BUAN 6382/models/lstm_model0_1.pth
```

9.7 Generation of sample text from the MODEL0_1.

```
[ ]  seed_text = "The soldiers marched forward and at a".lower()  # Convert to lowercase
     generated_text = generate_text(model0_1, seed_text, length=200)
     print(generated_text)
```

```
the soldiers marched forward and at a strange and the street the count and the count and the count and the count and
```

```
the count and the count and the count and the count and the count and the count and the count and the count and the count
```

9.8 We see that the model is repeating the same phrase ‘and the count’ again and again. This model was a very small model with only 144,175 trainable parameters.

```
[ ]  # Evaluate perplexity on the validation set
     val_perplexity = calculate_perplexity(model0_1, val_loader)
     print(f"Validation Perplexity: {val_perplexity:.2f}")

     entropy = calculate_entropy(generated_text)
     print(f"Entropy of Generated Text: {entropy:.2f}")
```

```
Validation Perplexity: 3.58
Entropy of Generated Text: 3.50
```

9.9 Compared to BASIC MODEL0, the perplexity is reduce by a small amount to 3.58 and entropy of the generated text is also reduced to 3.50 for the MODEL0_1.

9.10 Next we will try MODEL1_1, this model will have the same architecture as MODEL1 with sequence length of 100.

Model 1_1:

Single layer LSTM.

Training Sequence Length 100

Hidden size 512

Character level Embedding size 128

Batch Size 2048

9.11 Rest all the model parameters, optimizer, learning rate and loss function the same as MODEL1.

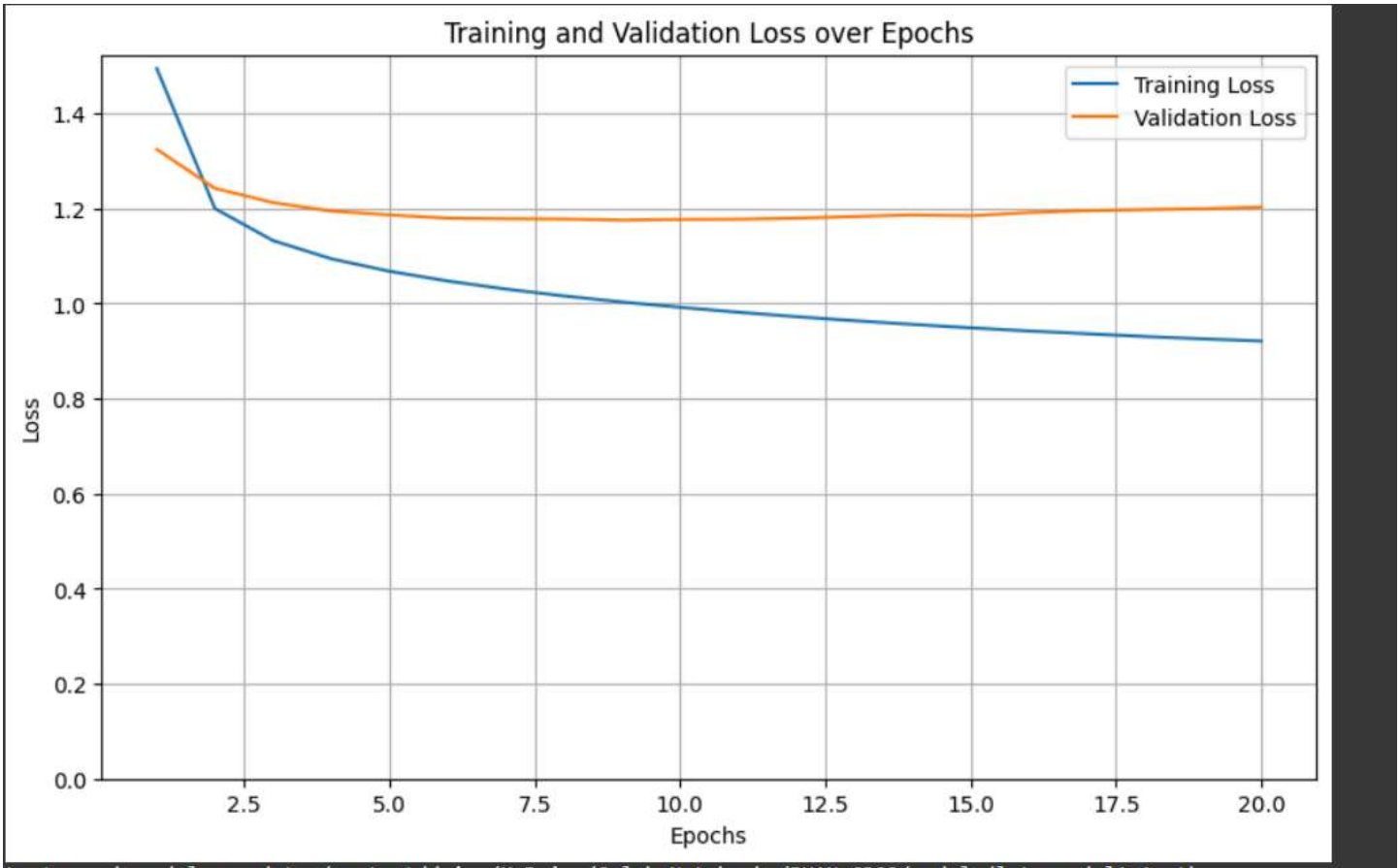
```
[ ]  train_loader = DataLoader(train_dataset, batch_size=2048, shuffle=True)
     val_loader = DataLoader(val_dataset, batch_size=2048)
```

```
     # Define model parameters
     vocab_size = len(chars)
     embed_size = 128
     hidden_size = 512

     model1_1 = LSTMModel(vocab_size, embed_size, hidden_size)

     # Define loss function and optimizer
     criterion = nn.CrossEntropyLoss()
     optimizer1_1 = optim.Adam(model1_1.parameters(), lr=0.001)
```

```python
# Train the model
trained_model1_1, best_model_state_dict1_1, train_losses, val_losses = train_model(
    model1_1, train_loader, val_loader, criterion, optimizer1_1, num_epochs=20, device=device
)

# Save the last epoch model
last_epoch_model_path = models_folder / "lstm_model1_1.pth"
torch.save(trained_model1_1.state_dict(), last_epoch_model_path)
print(f"Last epoch model saved to {last_epoch_model_path}")

# Save the best model based on validation loss
best_model_path = models_folder / "lstm_model1_1_best.pth"
torch.save(best_model_state_dict1_1, best_model_path)
print(f"Best validation loss model saved to {best_model_path}")
```

9.12 Total trainable model parameters are increased to 1,344,943.

```
Total number of trainable parameters in the model: 1344943
Epoch 1/20, Train Loss: 1.4945, Val Loss: 1.3245
Epoch 2/20, Train Loss: 1.1998, Val Loss: 1.2423
Epoch 3/20, Train Loss: 1.1324, Val Loss: 1.2124
Epoch 4/20, Train Loss: 1.0942, Val Loss: 1.1945
Epoch 5/20, Train Loss: 1.0679, Val Loss: 1.1865
Epoch 6/20, Train Loss: 1.0474, Val Loss: 1.1796
Epoch 7/20, Train Loss: 1.0305, Val Loss: 1.1783
Epoch 8/20, Train Loss: 1.0160, Val Loss: 1.1776
Epoch 9/20, Train Loss: 1.0033, Val Loss: 1.1755
Epoch 10/20, Train Loss: 0.9920, Val Loss: 1.1768
Epoch 11/20, Train Loss: 0.9817, Val Loss: 1.1773
Epoch 12/20, Train Loss: 0.9723, Val Loss: 1.1795
Epoch 13/20, Train Loss: 0.9639, Val Loss: 1.1830
Epoch 14/20, Train Loss: 0.9561, Val Loss: 1.1865
Epoch 15/20, Train Loss: 0.9489, Val Loss: 1.1848
Epoch 16/20, Train Loss: 0.9423, Val Loss: 1.1913
Epoch 17/20, Train Loss: 0.9365, Val Loss: 1.1954
Epoch 18/20, Train Loss: 0.9306, Val Loss: 1.1976
Epoch 19/20, Train Loss: 0.9257, Val Loss: 1.1994
Epoch 20/20, Train Loss: 0.9210, Val Loss: 1.2029
```




```
Last epoch model saved to /content/drive/MyDrive/Colab Notebooks/BUAN 6382/models/lstm_model1_1.pth
Best validation loss model saved to /content/drive/MyDrive/Colab Notebooks/BUAN 6382/models/lstm_model1_1_best.pth
```

9.13 We see that training loss is decreasing with each epoch, whereas the validation loss initially decreases and then starts to rise from the 10th epoch, suggesting overfitting.

9.14 We also observe that the train loss is higher than the validation loss at the end of the first epoch. This must be due to the fact that in the start of the first epoch, the model is initialized with random weights and as the training progresses, with each batch, the weights are updated. This leads to poor performance on the initial training batches thus increasing the overall training loss. By the time the validation dataset is evaluated, the model is trained to a certain degree and hence we see a lower validation loss.

```
[ ]  # Define model parameters
     model_path = models_folder / "lstm_model1_1_best.pth"
     vocab_size = len(chars)
     embed_size = 128
     hidden_size = 512
     num_layers = 1
     dropout_rate = 0

     model1_1 = load_model(LSTMModel1, model_path, vocab_size, embed_size, hidden_size, num_layers, dropout_rate, device)
```

9.15 The model is saved at the end of training in order to reuse it later. The load_model function is used to load the model into memory.

9.16 Generation of sample text from the MODEL1_1.

```
[ ]  seed_text = "The soldiers marched forward and at a".lower()  # Convert to lowercase
     generated_text = generate_text(model1_1, seed_text, length=200)
     print(generated_text)
```

```
the soldiers marched forward and at a distance which was already at the same time and the countess was a significance
```

```
of the conversation with his hand and said to him and the countess was a significance of the commander in chiefs attemp
```

9.17 We see that the model has performed much better than MODEL0_1. The model has learned to spell the words correctly, has proper spacing between words, uses a great variety of words in the generation, although does not make much sense.

```
[ ]  # Evaluate perplexity on the validation set
     val_perplexity = calculate_perplexity(model1_1, val_loader)
     print(f"Validation Perplexity: {val_perplexity:.2f}")

     entropy = calculate_entropy(generated_text)
     print(f"Entropy of Generated Text: {entropy:.2f}")
```

```
Validation Perplexity: 3.33
Entropy of Generated Text: 3.88
```

9.18 The MODEL1_1 has performed similarly to MODEL1. The perplexity has reduced from 3.31 to 3.33 while the entropy rises from 3.59 to 3.88. Thus using a sequence length of 100 instead of 200 has minimal effect on model performance.

10 Model Training: Section 3

10.1 In this section, we will yet further increase the size of the model. We will build upon MODEL1_1 and increase the hidden dimension from 512 to 1024. This will be model 2_1.

Model 2_1:

Single layer LSTM.

Training Sequence Length 100

Hidden size 1024

Character level Embedding size 128

Batch Size 2048

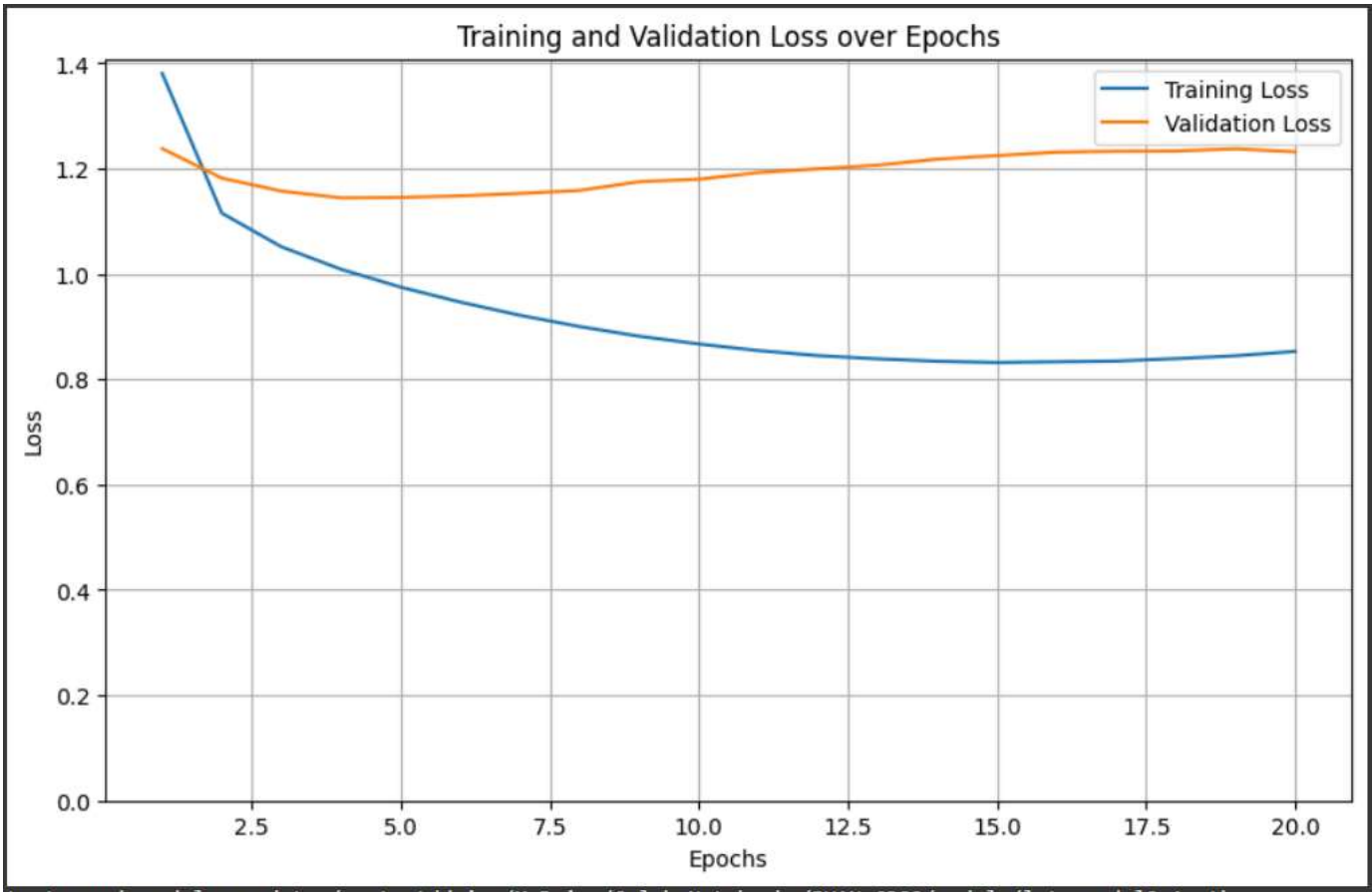
10.2 Rest all the model parameters, optimizer, learning rate and loss function the same as MODEL1_1.

```
# Define model parameters
vocab_size = len(chars)
embed_size = 128
hidden_size = 1024

model2_1 = LSTMModel2(vocab_size, embed_size, hidden_size)

# Define loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer2_1 = optim.Adam(model2_1.parameters(), lr=0.001)
```

```python
# Train the model
trained_model2_1, best_model_state_dict2_1, train_losses, val_losses = train_model(
    model2_1, train_loader, val_loader, criterion, optimizer2_1, num_epochs=20, device=device
)

# Save the last epoch model
last_epoch_model_path = models_folder / "lstm_model2_1.pth"
torch.save(trained_model2_1.state_dict(), last_epoch_model_path)
print(f"Last epoch model saved to {last_epoch_model_path}")

# Save the best model based on validation loss
best_model_path = models_folder / "lstm_model2_1_best.pth"
torch.save(best_model_state_dict2_1, best_model_path)
print(f"Best validation loss model saved to {best_model_path}")
```

10.3 Total trainable model parameters are increased to 4,780,975.

```
Total number of trainable parameters in the model: 4780975
Epoch 1/20, Train Loss: 1.3807, Val Loss: 1.2379
Epoch 2/20, Train Loss: 1.1153, Val Loss: 1.1818
Epoch 3/20, Train Loss: 1.0514, Val Loss: 1.1572
Epoch 4/20, Train Loss: 1.0087, Val Loss: 1.1443
Epoch 5/20, Train Loss: 0.9748, Val Loss: 1.1451
Epoch 6/20, Train Loss: 0.9464, Val Loss: 1.1481
Epoch 7/20, Train Loss: 0.9214, Val Loss: 1.1527
Epoch 8/20, Train Loss: 0.9000, Val Loss: 1.1584
Epoch 9/20, Train Loss: 0.8817, Val Loss: 1.1750
Epoch 10/20, Train Loss: 0.8670, Val Loss: 1.1797
Epoch 11/20, Train Loss: 0.8544, Val Loss: 1.1924
Epoch 12/20, Train Loss: 0.8448, Val Loss: 1.1995
Epoch 13/20, Train Loss: 0.8387, Val Loss: 1.2064
Epoch 14/20, Train Loss: 0.8342, Val Loss: 1.2179
Epoch 15/20, Train Loss: 0.8318, Val Loss: 1.2246
Epoch 16/20, Train Loss: 0.8332, Val Loss: 1.2311
Epoch 17/20, Train Loss: 0.8345, Val Loss: 1.2329
Epoch 18/20, Train Loss: 0.8392, Val Loss: 1.2333
Epoch 19/20, Train Loss: 0.8446, Val Loss: 1.2374
Epoch 20/20, Train Loss: 0.8528, Val Loss: 1.2319
```




```
Last epoch model saved to /content/drive/MyDrive/Colab Notebooks/BUAN 6382/models/lstm_model2_1.pth
Best validation loss model saved to /content/drive/MyDrive/Colab Notebooks/BUAN 6382/models/lstm_model2_1_best.pth
```

10.4 We see that training loss is decreasing with each epoch, whereas the validation loss initially decreases and then starts to rise from the 4th epoch, suggesting overfitting. Also, after some more epochs, the training and validation loss increase and decreases respectively.

10.5 We also observe that the train loss is higher than the validation loss at the end of the first epoch. This must be due to the fact that in the start of the first epoch, the model is initialized with random weights and as the training progresses, with each batch, the weights are updated. This leads to poor performance on the initial training batches thus increasing the overall training loss. By the time the validation dataset is evaluated, the model is trained to a certain degree and hence we see a lower validation loss.

```
[ ]  # Define model parameters
     model_path = models_folder / "lstm_model2_1_best.pth"
     vocab_size = len(chars)
     embed_size = 128
     hidden_size = 1024
     num_layers = 1
     dropout_rate = 0

     model2_1 = load_model(LSTMModel1, model_path, vocab_size, embed_size, hidden_size, num_layers, dropout_rate, device)
```

```
Model loaded from /content/drive/MyDrive/Colab Notebooks/BUAN 6382/models/lstm_model2_1_best.pth
```

10.6 The model is saved at the end of training in order to reuse it later. The load_model function is used to load the model into memory.

10.7 Generation of sample text from the MODEL2_1.

```
[ ]  seed_text = "The soldiers marched forward and at a".lower()  # Convert to lowercase
     generated_text = generate_text(model2_1, seed_text, length=200)
     print(generated_text)
```

```
the soldiers marched forward and at any moment the count was sent to him and the count was sent to her so far said the
count who was sitting in the corner of the street the count was seriously as if in any pained hole in his hand
```

10.8 We see that the model has performed similar to MODEL1_1. The model has learned to spell the words correctly, has proper spacing between words and uses a great variety of words in the generation.

10.9 One thing of qualitative observation, the generated text seems to be of a higher quality that seen in the previous models.

```
[ ]  # Evaluate perplexity on the validation set
     val_perplexity = calculate_perplexity(model2_1, val_loader)
     print(f"Validation Perplexity: {val_perplexity:.2f}")

     entropy = calculate_entropy(generated_text)
     print(f"Entropy of Generated Text: {entropy:.2f}")
```

```
Validation Perplexity: 3.42
Entropy of Generated Text: 3.82
```

10.10 The Perplexity of this model on validation dataset is 3.42 and Entropy of the generated text is 3.82.

10.11 Following the trend of building and training bigger models, we will now modify the MODEL2_1 to have a hidden dimension of size 2048. This will be MODEL2_2.

Model 2_2:

Single layer LSTM.

Training Sequence Length 50

Hidden size 2048

Character level Embedding size 128

Batch Size 2096

10.12 Rest all the model parameters, optimizer, learning rate and loss function the same as MODEL2_1.

10.13 To train this model we will use float16 datatype for the model parameters to reduce training time and memory consumption.

```
[ ]  from torch.amp import autocast, GradScaler
     import matplotlib.pyplot as plt

     def train_model(model, train_loader, val_loader, criterion, optimizer, num_epochs, device):
         # Move model to the device
         model.to(device)

         # Calculate the number of trainable parameters
         total_params = sum(p.numel() for p in model.parameters() if p.requires_grad)
         print(f"Total number of trainable parameters in the model: {total_params}")

         # Initialize GradScaler for mixed precision training
         scaler = GradScaler(init_scale=2.0, enabled=True)

         # Lists to store losses
         train_losses = []
         val_losses = []

         # Best validation loss and model
         best_val_loss = float('inf')
         best_model_state_dict = None

         # Training loop
         for epoch in range(num_epochs):
             model.train()
             train_loss = 0
             for inputs, targets in train_loader:
                 inputs, targets = inputs.to(device), targets.to(device)

                 # Forward pass with mixed precision
                 with autocast(device_type='cuda', dtype=torch.float16):
                     outputs, _ = model(inputs)
                     loss = criterion(outputs, targets)

                 # Backward pass
                 optimizer.zero_grad()
                 scaler.scale(loss).backward()
                 scaler.step(optimizer)
                 scaler.update()

                 train_loss += loss.item()
```

```
             # Validation loop
             model.eval()
             val_loss = 0
             with torch.no_grad():
                 for inputs, targets in val_loader:
                     inputs, targets = inputs.to(device), targets.to(device)

                     # Forward pass with mixed precision
                     with autocast(device_type='cuda', dtype=torch.float16):
                         outputs, _ = model(inputs)
                         loss = criterion(outputs, targets)

                     val_loss += loss.item()

             # Calculate average losses for the epoch
             train_loss /= len(train_loader)
             val_loss /= len(val_loader)

             # Store losses
             train_losses.append(train_loss)
             val_losses.append(val_loss)

             # Check for best validation loss
             if val_loss < best_val_loss:
                 best_val_loss = val_loss
                 best_model_state_dict = model.state_dict()  # Save the state dict of the best model

             print(f"Epoch {epoch+1}/{num_epochs}, Train Loss: {train_loss:.4f}, Val Loss: {val_loss:.4f}")
```

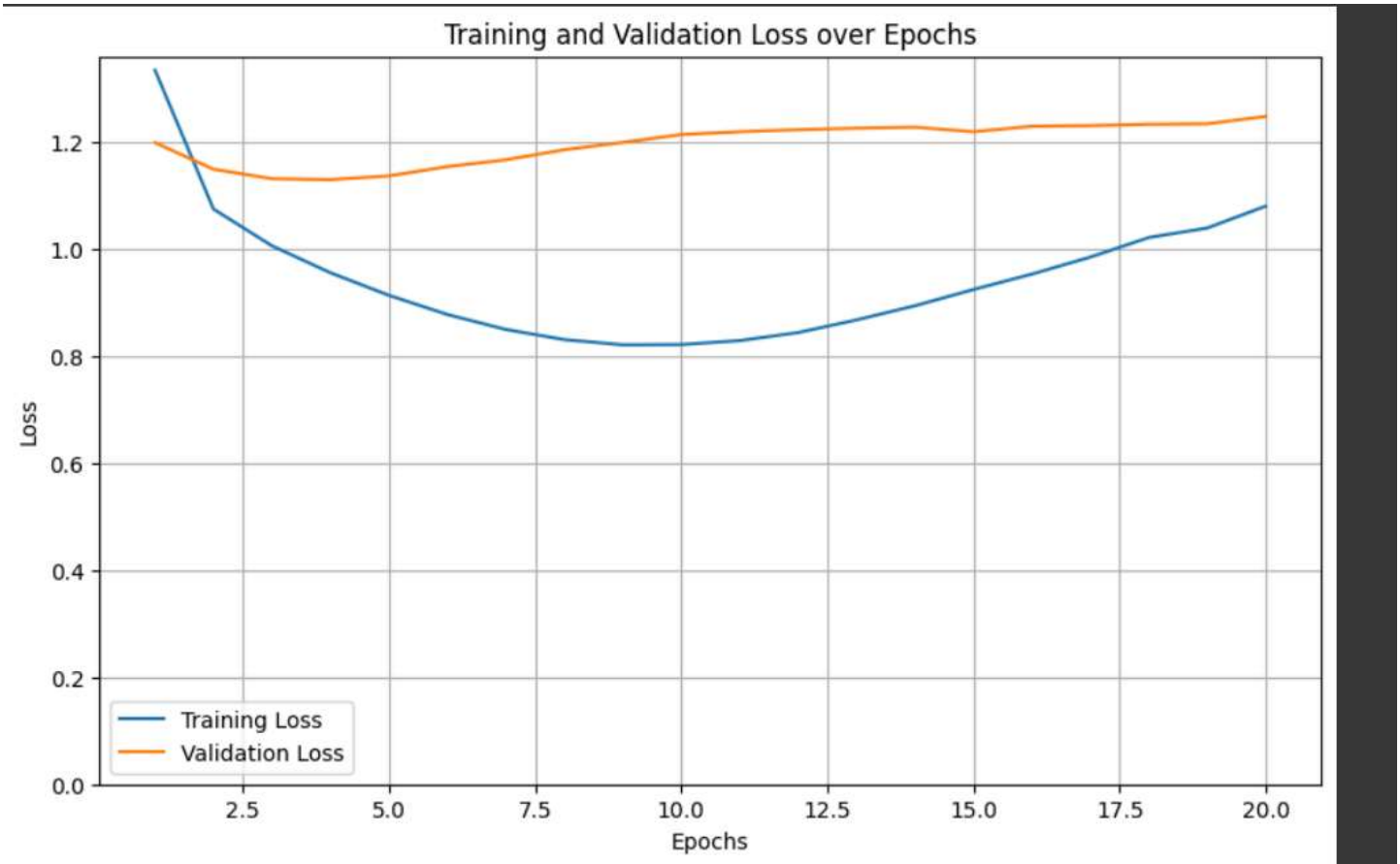
```
    # Plot the training and validation losses
    plt.figure(figsize=(10, 6))
    plt.plot(range(1, num_epochs + 1), train_losses, label='Training Loss')
    plt.plot(range(1, num_epochs + 1), val_losses, label='Validation Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.ylim(0)  # Start y-axis from 0
    plt.title('Training and Validation Loss over Epochs')
    plt.legend()
    plt.grid(True)
    plt.show()

    # Return the model and the best model state dict
    return model, best_model_state_dict, train_losses, val_losses
```

```
# Define model parameters
vocab_size = len(chars)
embed_size = 128
hidden_size = 2048

model2_2 = LSTMModel(vocab_size, embed_size, hidden_size)

# Define loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer2_2 = optim.Adam(model2_2.parameters(), lr=0.001)
```

```
# Train the model
trained_model2_2, best_model_state_dict2_2, train_losses, val_losses = train_model(
    model2_2, train_loader, val_loader, criterion, optimizer2_2, num_epochs=20, device=device
)

# Save the last epoch model
last_epoch_model_path = models_folder / "lstm_model2_2.pth"
torch.save(trained_model2_2.state_dict(), last_epoch_model_path)
print(f"Last epoch model saved to {last_epoch_model_path}")

# Save the best model based on validation loss
best_model_path = models_folder / "lstm_model2_2_best.pth"
torch.save(best_model_state_dict2_2, best_model_path)
print(f"Best validation loss model saved to {best_model_path}")
```

10.14 Total trainable model parameters are increased to 17,944,495.

```
Total number of trainable parameters in the model: 17944495
Epoch 1/20, Train Loss: 1.3345, Val Loss: 1.1991
Epoch 2/20, Train Loss: 1.0750, Val Loss: 1.1494
Epoch 3/20, Train Loss: 1.0062, Val Loss: 1.1317
Epoch 4/20, Train Loss: 0.9559, Val Loss: 1.1300
Epoch 5/20, Train Loss: 0.9137, Val Loss: 1.1369
Epoch 6/20, Train Loss: 0.8779, Val Loss: 1.1543
Epoch 7/20, Train Loss: 0.8500, Val Loss: 1.1670
Epoch 8/20, Train Loss: 0.8310, Val Loss: 1.1858
Epoch 9/20, Train Loss: 0.8212, Val Loss: 1.1995
Epoch 10/20, Train Loss: 0.8217, Val Loss: 1.2142
Epoch 11/20, Train Loss: 0.8291, Val Loss: 1.2195
Epoch 12/20, Train Loss: 0.8441, Val Loss: 1.2233
Epoch 13/20, Train Loss: 0.8678, Val Loss: 1.2261
Epoch 14/20, Train Loss: 0.8943, Val Loss: 1.2280
Epoch 15/20, Train Loss: 0.9246, Val Loss: 1.2195
Epoch 16/20, Train Loss: 0.9534, Val Loss: 1.2297
Epoch 17/20, Train Loss: 0.9852, Val Loss: 1.2308
Epoch 18/20, Train Loss: 1.0218, Val Loss: 1.2334
Epoch 19/20, Train Loss: 1.0395, Val Loss: 1.2343
Epoch 20/20, Train Loss: 1.0805, Val Loss: 1.2481
```




```
Last epoch model saved to /content/drive/MyDrive/Colab Notebooks/BUAN 6382/models/lstm_model2_2.pth
Best validation loss model saved to /content/drive/MyDrive/Colab Notebooks/BUAN 6382/models/lstm_model2_2_best.pth
```

10.15 We see that training loss is decreasing with each epoch, whereas the validation loss initially decreases and then starts to rise from the 4th epoch, suggesting overfitting. Also, after some more epochs, the training and validation loss increase and decreases respectively.

10.16 We also observe that the train loss is higher than the validation loss at the end of the first epoch. This must be due to the fact that in the start of the first epoch, the model is initialized with random weights and as the training progresses, with each batch, the weights are updated. This leads to poor performance on the initial training batches thus increasing the overall training loss. By the time the validation dataset is evaluated, the model is trained to a certain degree and hence we see a lower validation loss.

10.17 The model is saved at the end of training in order to reuse it later. The load_model function is used to load the model into memory.

```
[ ]  # Define model parameters
     model_path = models_folder / "lstm_model2_2_best.pth"
     vocab_size = len(chars)
     embed_size = 128
     hidden_size = 2048
     num_layers = 1
     dropout_rate = 0

     model2_2 = load_model(LSTMModel2, model_path, vocab_size, embed_size, hidden_size, num_layers, dropout_rate, device)
```

```
Model loaded from /content/drive/MyDrive/Colab Notebooks/BUAN 6382/models/lstm_model2_2_best.pth
```

10.18 Generation of sample text from the MODEL2_2.

```
[ ]  seed_text = "The soldiers marched forward and at a".lower()  # Convert to lowercase
     generated_text = generate_text(model2_2, seed_text, length=200)
     print(generated_text)
```

```
the soldiers marched forward and at a stage of the commander of the regiment and the commander of the regiment and
the staff and the soldiers were standing in the commander of the regiment and the commander of the regiment and
```

10.19 We see that the model has performed worse than MODEL2_1. The model keeps repeating words ‘commander’ and ‘regiment’.

```
[ ]  # Evaluate perplexity on the validation set
     val_perplexity = calculate_perplexity(model2_2, val_loader)
     print(f"Validation Perplexity: {val_perplexity:.2f}")

     entropy = calculate_entropy(generated_text)
     print(f"Entropy of Generated Text: {entropy:.2f}")
```

```
Validation Perplexity: 3.48
Entropy of Generated Text: 3.75
```

10.20 Validation perplexity for the model is 3.48 and entropy of the text generated from the model is 3.75.

11 Model Training: Section 4

11.1 Next, we will try to go with a deeper model with multiple layers.

11.2 One observation that can be seen from the previous models is that the model tends to overfit to the training data after the first few epochs. So we will try to have some dropout between the layers as well. A dropout of 0.5 is selected.

11.3 We will modify MODEL1_1 to have a two layer LSTM architecture. This will be MODEL4

Model 4:

Two layer LSTM.

Hidden size 512

Character level Embedding size 128

Batch Size 3200

Dropout 0.5

11.4 We will have a batch size of 3200 to maximize the memory utilization of our GPU.

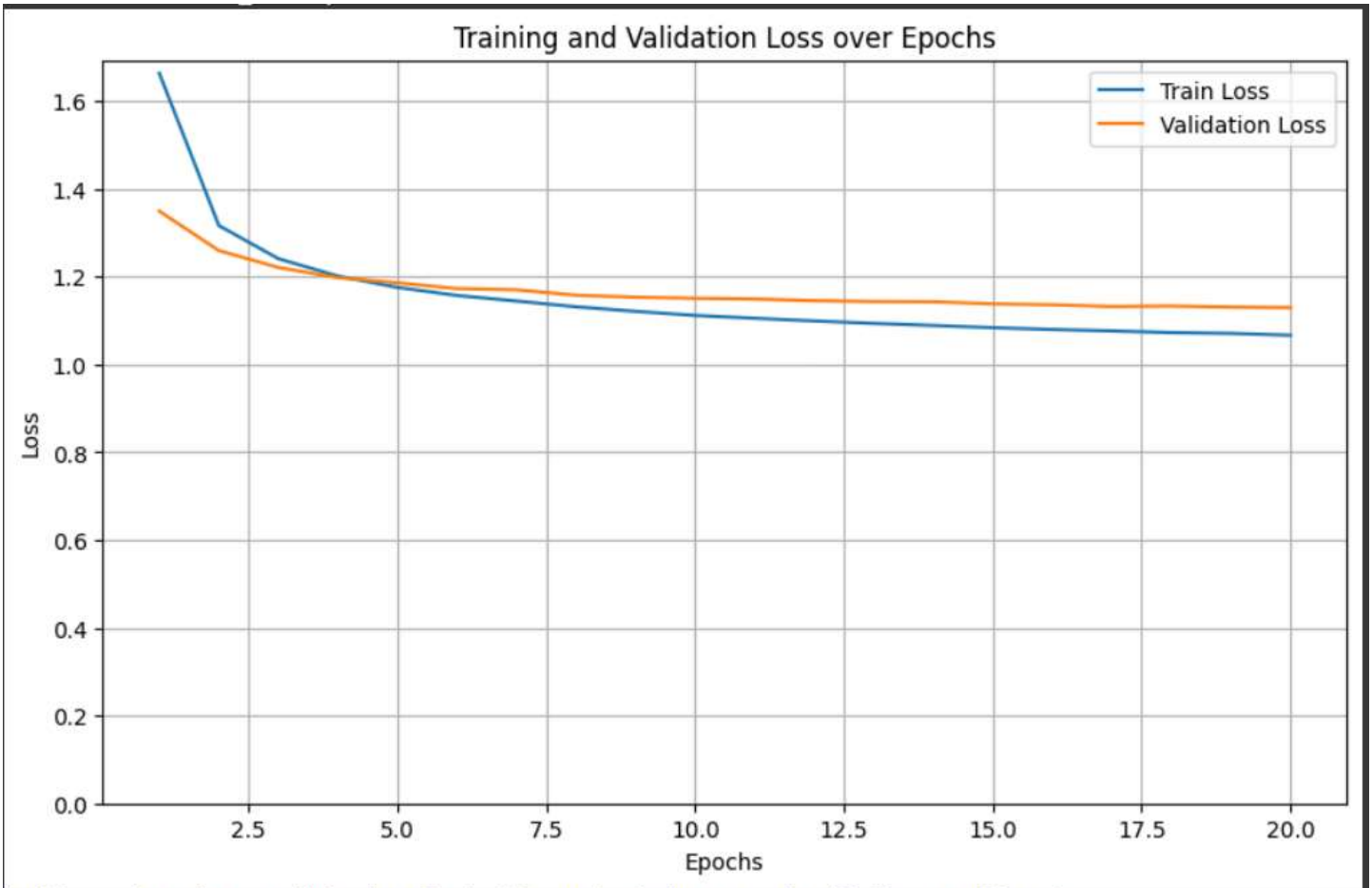
```
[ ]  train_loader = DataLoader(train_dataset, batch_size=3200, shuffle=True)
     val_loader = DataLoader(val_dataset, batch_size=3200)
```

```
# Define model parameters
vocab_size = len(chars)
embed_size = 128
hidden_size = 512
num_layers = 2
dropout_rate = 0.5

# Instantiate the model with dropout
model4 = LSTMModel(vocab_size, embed_size, hidden_size, num_layers, dropout=dropout_rate)

# Define loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer4 = optim.Adam(model4.parameters(), lr=0.001)
```

```
# Train the model
trained_model, train_losses, val_losses = train_model(
    model4, train_loader, val_loader, criterion, optimizer4, num_epochs=20, device=device
)

# Save the model to Google Drive
model_save_path = models_folder / "lstm_model4.pth"
torch.save(trained_model.state_dict(), model_save_path)
print(f"Model saved to {model_save_path}")
```

11.5 Total trainable model parameters are increased to 3,446,191.

```
Total number of trainable parameters in the model: 3446191
Epoch 1/20, Train Loss: 1.6630, Val Loss: 1.3491
Epoch 2/20, Train Loss: 1.3162, Val Loss: 1.2592
Epoch 3/20, Train Loss: 1.2408, Val Loss: 1.2207
Epoch 4/20, Train Loss: 1.2012, Val Loss: 1.1976
Epoch 5/20, Train Loss: 1.1753, Val Loss: 1.1858
Epoch 6/20, Train Loss: 1.1570, Val Loss: 1.1727
Epoch 7/20, Train Loss: 1.1443, Val Loss: 1.1699
Epoch 8/20, Train Loss: 1.1312, Val Loss: 1.1575
Epoch 9/20, Train Loss: 1.1208, Val Loss: 1.1530
Epoch 10/20, Train Loss: 1.1115, Val Loss: 1.1506
Epoch 11/20, Train Loss: 1.1051, Val Loss: 1.1489
Epoch 12/20, Train Loss: 1.0989, Val Loss: 1.1452
Epoch 13/20, Train Loss: 1.0934, Val Loss: 1.1431
Epoch 14/20, Train Loss: 1.0887, Val Loss: 1.1426
Epoch 15/20, Train Loss: 1.0836, Val Loss: 1.1379
Epoch 16/20, Train Loss: 1.0795, Val Loss: 1.1357
Epoch 17/20, Train Loss: 1.0763, Val Loss: 1.1319
Epoch 18/20, Train Loss: 1.0725, Val Loss: 1.1331
Epoch 19/20, Train Loss: 1.0707, Val Loss: 1.1305
Epoch 20/20, Train Loss: 1.0664, Val Loss: 1.1293
Model saved to lstm_model.pth
```



11.6 We see that training loss and validation loss decreases as we train the model for more epochs.

11.7 Also, the two losses are very close to each other suggesting that the overfitting is being curbed by using dropout.

11.8 We also observe that the train loss is higher than the validation loss at the end of the first epoch. This must be due to the fact that in the start of the first epoch, the model is initialized with random weights and as the training progresses, with each batch, the weights are updated. This leads to poor performance on the initial training batches thus increasing the overall training loss. By the time the validation dataset is evaluated, the model is trained to a certain degree and hence we see a lower validation loss.

11.9 The model is saved at the end of training in order to reuse it later. The load_model funtion is used to load the model into memory.

```
[ ]  # Model parameters
     model_path = models_folder / "lstm_model4.pth"
     vocab_size = len(chars)
     embed_size = 128
     hidden_size = 512
     num_layers = 2
     dropout_rate = 0.5

     # Load the model
     model4 = load_model(LSTMModel4, model_path, vocab_size, embed_size, hidden_size, num_layers, dropout_rate, device)
```

11.10 Generation of sample text from the MODEL4.

```
     seed_text = "The soldiers marched forward and at a".lower()  # Convert to lowercase
     generated_text = generate_text(model4, seed_text, length=200)
     print(generated_text)
```

```
the soldiers marched forward and at a state of the staff of the staff of the staff of the staff of the staff of the street
```

```
the count was a stranger and the same thing the count was a stranger and the same thing the count was
```

11.11 The model did not perform so well. It keeps on repeating the words “of the staff” and the overall diversity of words used is lacking.

```
[16] # Evaluate perplexity on the validation set
     val_perplexity = calculate_perplexity(model4, val_loader)
     print(f"Validation Perplexity: {val_perplexity:.2f}")

     entropy = calculate_entropy(generated_text)
     print(f"Entropy of Generated Text: {entropy:.2f}")

     Validation Perplexity: 3.10
     Entropy of Generated Text: 3.69
```

11.12 Validation perplexity for the model is 3.10 and entropy of the text generated from the model is 3.69.

12 Model Training: Section 5

12.1 BPE Tokenization

12.1.1 We have tried the next set of models using a few set of model improvement techniques, one being Byte-pair encoding for Tokenization.

12.1.2 In Byte Pair Encoding (BPE) tokenization, frequently occurring character pairs in a text are merged into new tokens, creating a vocabulary that represents both common words and subword units.

12.1.3 Since from the previous models we were able to decipher that larger sequence lengths - 200 or 150 aren't giving us good results, we shall start our models from lower sequential lengths i.e. 25 and try our chances with 50 and 100 combinations.

12.1.4 We can use various Vocab sizes for Byte- pair encoding. For our experiments we have used tokenizer vocab sizes of 1000, 3000, 5000.

12.1.5 We have also experimented with different combinations of embedding size and also hidden layers' size based on the outputs.

12.1.6 Since we have used 1 layer in the character level tokenization models, we shall try using 2 layer models, i.e. adding 1 additional hidden layer. Also to avoid any overfitting to the train data, we will use dropouts between the layers.

12.1.7 We have used a constant batch size throughout the BPE set of models for better comparisons.

12.1.8 We have used a constant learning rate throughout the BPE models and after finding the best model, we shall implement a learner rate scheduler on the best model with best of the hyper parameters we have used.

12.1.9 We have used 20 epochs for this model. If the validation loss continues to decrease even after the 20th epoch we shall further train the model to higher epochs.

Model3_6:

```
Vocab size : 1000
Sequence length : 25
embed_size = 256
hidden_size = 1024
num_layers = 2
dropout_rate = 0.3
```

- Vocab size : 1000
- Sequence length : 25
- embed_size = 256

- hidden_size = 1024
- num_layers = 2
- dropout_rate = 0.3
- Number of epochs = 20
- Batch_size = 1024
- Learning rate = 0.001

12.1.10 In this model we have started with a vocab size of 1000 and a smaller sequence length of 25.

```
# Build a BPE tokenizer
tokenizer = Tokenizer(BPE(unk_token="<unk>"))
trainer = BpeTrainer(
    vocab_size=1000,
    min_frequency=1,
    special_tokens=["<unk>", "<pad>", "<s>", "</s>"]
)
tokenizer.pre_tokenizer = Whitespace()

# Train the tokenizer
tokenizer.train_from_iterator([cleaned_text], trainer)

# Tokenize the text
encoded = tokenizer.encode(cleaned_text)
text_as_int = np.array(encoded.ids)

# Define sequence length and prepare input-output pairs
sequence_length = 25
sequences = []
targets = []
```

12.1.11 After this we will be creating input-output pairs from the data, create a Custom dataset class to store the input sequence and output target in the same object. Splitting the data into Train and Validation datasets.

12.1.12 We will use the same ‘train_model’ as in the previous sections.

12.1.13 The ‘generate_text’ function has been slightly modified to accept the tokenizer as an argument to generate text from the model, one token at a time.

```
# Function to generate text
def generate_text(model, tokenizer, seed_text, length, device):
    model.eval()
    generated_ids = tokenizer.encode(seed_text).ids
    input_ids = torch.tensor(generated_ids, dtype=torch.long).unsqueeze(0).to(device)

    hidden = None

    for _ in range(length):
        with torch.no_grad():
            output, hidden = model(input_ids, hidden)
            probs = torch.nn.functional.softmax(output, dim=-1)
            # Sample from the distribution or take the argmax
            next_token_id = torch.multinomial(probs, num_samples=1).item()

        # Append the predicted token id to the generated_ids
        generated_ids.append(next_token_id)

        # Update input_ids to contain the new token
        input_ids = torch.tensor([[next_token_id]], dtype=torch.long).to(device)

    # Decode the generated ids to text
    generated_text = tokenizer.decode(generated_ids)
    return generated_text
```

12.1.14 Initially to start with, We have an embedding size of 256, hidden layer size of 1024 with 2 hidden layers and drop out of 0.3. Here, we are trying to implement a deeper network (2 hidden layers) as a part of model improvement. Based on the outputs we will make necessary changes in the parameters to reach to a better model at every step.

```
# Define model parameters
vocab_size = tokenizer.get_vocab_size()
embed_size = 256
hidden_size = 1024
num_layers = 2
dropout_rate = 0.3

model3_6 = LSTMModel3(vocab_size, embed_size, hidden_size, num_layers=num_layers,
                      dropout=dropout_rate)

# Define loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer3_6 = optim.Adam(model3_6.parameters(), lr=0.001)

# Set device (GPU if available)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

12.1.15 We have saved the best model based on the validation loss and will use it further for text generation.

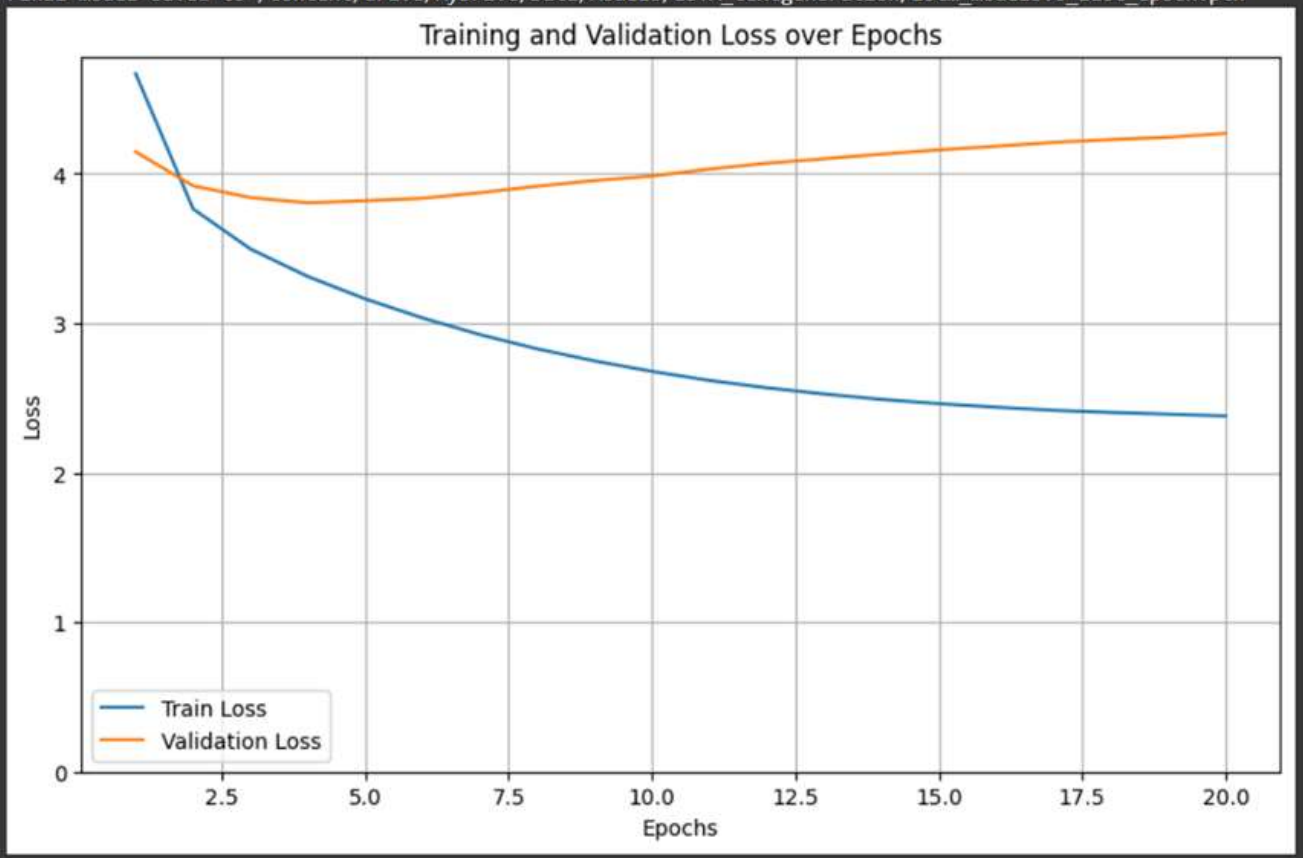
```python
model_folder.mkdir(parents=True, exist_ok=True)  # Ensure the folder exists

# Define paths for saving the best and last epoch models
best_model_path = model_folder / "lstm_model3.6_best.pth"
final_model_path = model_folder / "lstm_model3.6_last_epoch.pth"

# Train the model
trained_model, best_model_state_dict, train_losses, val_losses = train_model(
    model=model3_6,
    train_loader=train_loader,
    val_loader=val_loader,
    criterion=criterion,
    optimizer=optimizer3_6,
    num_epochs=20,
    device=device,
    best_model_path=best_model_path,
    final_model_path=final_model_path
)

# Print the locations of the saved models
print(f"Last epoch model saved to {final_model_path}")
print(f"Best validation loss model saved to {best_model_path}")
```

```
Total number of trainable parameters in the model: 14928872
Epoch 1/20 --> Train Loss: 4.6693, Val Loss: 4.1465
Best model saved with validation loss: 4.1465
Epoch 2/20 --> Train Loss: 3.7653, Val Loss: 3.9192
Best model saved with validation loss: 3.9192
Epoch 3/20 --> Train Loss: 3.4970, Val Loss: 3.8411
Best model saved with validation loss: 3.8411
Epoch 4/20 --> Train Loss: 3.3130, Val Loss: 3.8059
Best model saved with validation loss: 3.8059
Epoch 5/20 --> Train Loss: 3.1633, Val Loss: 3.8197
Best model saved with validation loss: 3.8059
Epoch 6/20 --> Train Loss: 3.0355, Val Loss: 3.8360
Best model saved with validation loss: 3.8059
Epoch 7/20 --> Train Loss: 2.9242, Val Loss: 3.8735
Best model saved with validation loss: 3.8059
Epoch 8/20 --> Train Loss: 2.8288, Val Loss: 3.9168
Best model saved with validation loss: 3.8059
Epoch 9/20 --> Train Loss: 2.7481, Val Loss: 3.9548
Best model saved with validation loss: 3.8059
Epoch 10/20 --> Train Loss: 2.6783, Val Loss: 3.9846
Best model saved with validation loss: 3.8059
```

```
Epoch 11/20 --> Train Loss: 2.6176, Val Loss: 4.0316
Best model saved with validation loss: 3.8059
Epoch 12/20 --> Train Loss: 2.5686, Val Loss: 4.0703
Best model saved with validation loss: 3.8059
Epoch 13/20 --> Train Loss: 2.5270, Val Loss: 4.1004
Best model saved with validation loss: 3.8059
Epoch 14/20 --> Train Loss: 2.4911, Val Loss: 4.1313
Best model saved with validation loss: 3.8059
Epoch 15/20 --> Train Loss: 2.4625, Val Loss: 4.1599
Best model saved with validation loss: 3.8059
Epoch 16/20 --> Train Loss: 2.4387, Val Loss: 4.1841
Best model saved with validation loss: 3.8059
Epoch 17/20 --> Train Loss: 2.4179, Val Loss: 4.2100
Best model saved with validation loss: 3.8059
Epoch 18/20 --> Train Loss: 2.4036, Val Loss: 4.2288
Best model saved with validation loss: 3.8059
Epoch 19/20 --> Train Loss: 2.3922, Val Loss: 4.2444
Best model saved with validation loss: 3.8059
Epoch 20/20 --> Train Loss: 2.3800, Val Loss: 4.2704
Best model saved with validation loss: 3.8059
```

12.1.16 We can see that the model's validation loss increases after the fourth epoch. This gives us the information that there is no point in increasing the number of epochs beyond 20.

```
Final model saved to /content/drive/MyDrive/Data/Models/LSTM_textgeneration/lstm_model3.6_last_epoch.pth
```




```python
# Function to load the model
def load_model(model_class, model_path, vocab_size, embed_size, hidden_size, num_layers,
               dropout_rate, device):
    model = model_class(vocab_size, embed_size, hidden_size, num_layers=num_layers,
                        dropout=dropout_rate)
    model.load_state_dict(torch.load(model_path, map_location=device))
    model.to(device)
    model.eval()
    return model

# Load the best model
model_path = os.path.join(model_folder, "lstm_model3.6_best.pth")
model3_6_loaded = load_model(LSTMModel3, model_path, vocab_size, embed_size, hidden_size,
                             num_layers, dropout_rate, device)
```

12.1.17 Generation of sample text from the MODEL3_6.

```python
# Generate text using the loaded model
seed_text = "The king and kingdom".lower()  # Convert to lowercase
generated_text = generate_text(model3_6_loaded, tokenizer, seed_text, length=200, device=device)
print(generated_text)
```

```
('the king and king d om of this mor al mo od of all de ath pl ans we should '
 'be go ver road and now at first i would did be lie ve it and that it may be '
 'happ ened too l ater why what i can and thats re very ni ce but i con sid er '
 'that at once in v ited him a f ine fellow he ap qui red why he should not do '
 'd g ers as l or d said dólokhov bol k ón ski he ad mi red his for ced de cam '
 'p in his go ld ent repe ated kutúzov well but i su p po se you ans w er on '
 'ac count he d rew a very end b ri ll i ant ha p pier that he af fe cted him '
 'he had made over his gu est s has brought a morning a cont r ast a ir th rew '
 'oh i am a fr a id there was nothing in d ang er that and f if te en wr it he '
 'gl ad ly right inst e ad of spe aking or dered a word to pro vis ions')
```

12.1.18 We can see from the output that most words have unnecessary spaces between them, i.e. the model is not able to predict complete words without any space for most of the uncommon words like ‘moral’, ‘mood’ ‘plans’ etc. The generation is grammatically inaccurate and lacks cohesion. The reasons can be due to the sequence length, vocab size, the embedding size and drop out. Though we can’t try all the combinations, we will make sure to try some of them.

```
# Evaluate perplexity on the validation set
val_perplexity = calculate_perplexity(model3_6_loaded, val_loader)
print(f"Validation Perplexity: {val_perplexity:.2f}")

# Evaluate entropy on the generated text
entropy = calculate_entropy(generated_text)
print(f"Entropy of Generated Text: {entropy:.2f}")
```

```
Validation Perplexity: 44.46
Entropy of Generated Text: 3.91
```

12.1.19 We got the perplexity of this as 44.46 (which is high) and entropy of 3.91. Since the generation is not giving us any meaningful insights, we will try out different sequence lengths with the same vocab size and see if there is any change. Maybe giving a higher sequence length will help the model capture the context better for coherent generation.

Model3_7:

```
Model3_7:
Vocab size : 1000
Sequence length : 50
embed_size = 512
hidden_size = 1024
num_layers = 2
dropout_rate = 0.3
```

- Vocab size : 1000
- Sequence length : 50
- embed_size = 512
- hidden_size = 1024
- num_layers = 2
- dropout_rate = 0.3
- Number of epochs = 20

- Batch_size = 1024
- Learning rate = 0.001

12.1.20 In this model we have tried to see the effect of increase in sequence length to 50.

```
# Build a BPE tokenizer
tokenizer = Tokenizer(BPE(unk_token="<unk>"))
trainer = BpeTrainer(
    vocab_size=1000,
    min_frequency=1,
    special_tokens=["<unk>", "<pad>", "<s>", "</s>"]
)
tokenizer.pre_tokenizer = Whitespace()

# Train the tokenizer
tokenizer.train_from_iterator([cleaned_text], trainer)

# Tokenize the text
encoded = tokenizer.encode(cleaned_text)
text_as_int = np.array(encoded.ids)

# Define sequence length and prepare input-output pairs
sequence_length = 50
sequences = []
targets = []
```

12.1.21 Additionally, as an experiment we have increased the Embedding size to 512 to see if the performance increases, as we know that more embedding size more information capture. As we increase the sequence length from 25 to 50 we are expecting that we are giving more tokens to the model to generate the next token, the model performance should increase.

```
# Define model parameters
vocab_size = tokenizer.get_vocab_size()
embed_size = 512
hidden_size = 1024
num_layers = 2
dropout_rate = 0.3

model3_7 = LSTMModel3(vocab_size, embed_size, hidden_size, num_layers=num_layers, dropout=dropout_rate)

# Define loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer3_7 = optim.Adam(model3_7.parameters(), lr=0.001)

# Set device (GPU if available)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

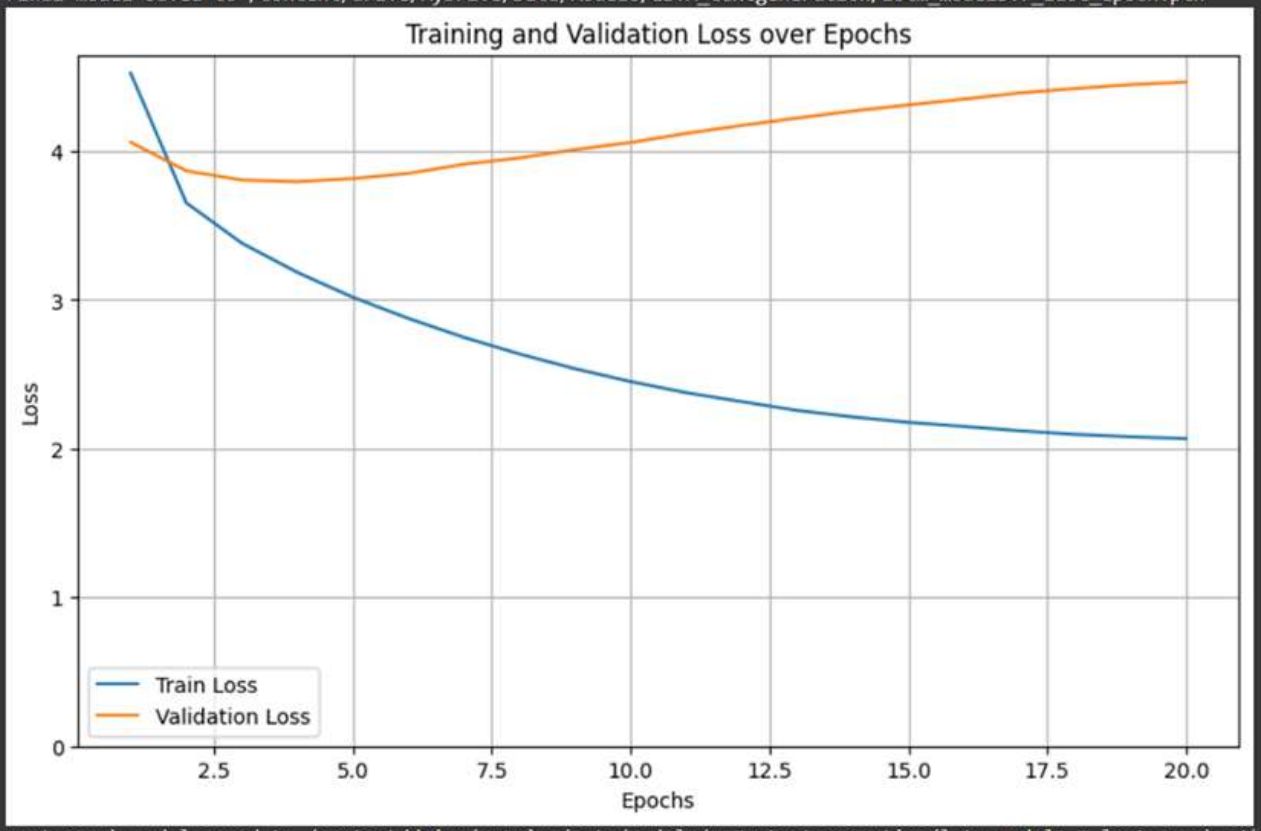
```python
model_folder.mkdir(parents=True, exist_ok=True)  # Ensure the folder exists

# Define paths for saving the best and last epoch models
best_model_path = model_folder / "lstm_model3.7_best.pth"
final_model_path = model_folder / "lstm_model3.7_last_epoch.pth"

# Train the model
trained_model, best_model_state_dict, train_losses, val_losses = train_model(
    model=model3_7,
    train_loader=train_loader,
    val_loader=val_loader,
    criterion=criterion,
    optimizer=optimizer3_7,
    num_epochs=20,
    device=device,
    best_model_path=best_model_path,
    final_model_path=final_model_path
)

# Print the locations of the saved models
print(f"Last epoch model saved to {final_model_path}")
print(f"Best validation loss model saved to {best_model_path}")
```

```
Total number of trainable parameters in the model: 16233448
Epoch 1/20 --> Train Loss: 4.5231, Val Loss: 4.0592
Best model saved with validation loss: 4.0592
Epoch 2/20 --> Train Loss: 3.6523, Val Loss: 3.8673
Best model saved with validation loss: 3.8673
Epoch 3/20 --> Train Loss: 3.3813, Val Loss: 3.8056
Best model saved with validation loss: 3.8056
Epoch 4/20 --> Train Loss: 3.1846, Val Loss: 3.7944
Best model saved with validation loss: 3.7944
Epoch 5/20 --> Train Loss: 3.0174, Val Loss: 3.8155
Best model saved with validation loss: 3.7944
Epoch 6/20 --> Train Loss: 2.8751, Val Loss: 3.8499
Best model saved with validation loss: 3.7944
Epoch 7/20 --> Train Loss: 2.7480, Val Loss: 3.9120
Best model saved with validation loss: 3.7944
Epoch 8/20 --> Train Loss: 2.6362, Val Loss: 3.9537
Best model saved with validation loss: 3.7944
Epoch 9/20 --> Train Loss: 2.5368, Val Loss: 4.0103
Best model saved with validation loss: 3.7944
Epoch 10/20 --> Train Loss: 2.4509, Val Loss: 4.0576
Best model saved with validation loss: 3.7944
Epoch 11/20 --> Train Loss: 2.3761, Val Loss: 4.1190
Best model saved with validation loss: 3.7944
Epoch 12/20 --> Train Loss: 2.3158, Val Loss: 4.1726
Best model saved with validation loss: 3.7944
Epoch 13/20 --> Train Loss: 2.2567, Val Loss: 4.2219
Best model saved with validation loss: 3.7944
Epoch 14/20 --> Train Loss: 2.2128, Val Loss: 4.2694
Best model saved with validation loss: 3.7944
Epoch 15/20 --> Train Loss: 2.1766, Val Loss: 4.3103
Best model saved with validation loss: 3.7944
Epoch 16/20 --> Train Loss: 2.1491, Val Loss: 4.3495
Best model saved with validation loss: 3.7944
```

```
Epoch 17/20 --> Train Loss: 2.1200, Val Loss: 4.3908
Best model saved with validation loss: 3.7944
Epoch 18/20 --> Train Loss: 2.0957, Val Loss: 4.4204
Best model saved with validation loss: 3.7944
Epoch 19/20 --> Train Loss: 2.0798, Val Loss: 4.4467
Best model saved with validation loss: 3.7944
Epoch 20/20 --> Train Loss: 2.0673, Val Loss: 4.4640
Best model saved with validation loss: 3.7944
```

12.1.22 We can see that the model's validation loss increases after the fourth epoch. This gives us the information that there is no point in increasing the number of epochs beyond 20. But the loss value is lesser than the previous model. Hopefully, we have a coherent generation as compared to the previous model.

```
Final model saved to /content/drive/MyDrive/Data/Models/LSTM_textgeneration/lstm_model3.7_last_epoch.pth
```




```
Last epoch model saved to /content/drive/MyDrive/Data/Models/LSTM_textgeneration/lstm_model3.7_last_epoch.pth
Best validation loss model saved to /content/drive/MyDrive/Data/Models/LSTM_textgeneration/lstm_model3.7_best.pth
```

```python
# Function to load the model
from pprint import pprint
def load_model(model_class, model_path, vocab_size, embed_size, hidden_size, num_layers, dropout_rate, device):
    model = model_class(vocab_size, embed_size, hidden_size, num_layers=num_layers, dropout=dropout_rate)
    model.load_state_dict(torch.load(model_path, map_location=device))
    model.to(device)
    model.eval()
    return model

# Load the best model
model_path = os.path.join(model_folder, "lstm_model3.7_best.pth")
model3_7_loaded = load_model(LSTMModel3, model_path, vocab_size, embed_size, hidden_size, num_layers,
                             dropout_rate, device)

# Generate text using the loaded model
seed_text = "The king and kingdom".lower()  # Convert to lowercase
generated_text = generate_text(model3_7_loaded, tokenizer, seed_text, length=200, device=device)
pprint(generated_text)
```

12.1.23 Generation of sample text from the MODEL3_7.

```
('the king and king d om that did not ar ouse his days kutúzov s su ite and '
 'that he had not ex pos ed the le ad er one of whom was your ex ce ll enc y '
 'who has been su ff ering to gr as p their com r ad es people des cri bed '
 'napoleon must be str ong er one and the cor p or al gl anc ing h ar st ness '
 'ent ering his mind f right ened and who had had bes ide the fe w step s he '
 'went to the door that ru shed but this h ard had to be f illed from them '
 'with fe ar as possible he was in his pa use and point ed to the in f ant ry '
 'officer on see ing the appro ach ing step h ard among his ha ir w she n ast '
 'ard down up into the ho llow and looked through the foot men laugh ing and '
 'how something they have known from god s we can st ru g g le and sto pped '
 'the ch ar g é d in just look with a p h y s ic ity ness')
```

12.1.24 Through manual inspection, it can be seen that the generated text has less unnecessary spaces in the words (or incomplete word generation) as compared to the previous model and is slightly better in context and coherence as compared to the previous generation, but it still has a lot more unnecessary spaces generated.

12.1.25 So, to know if the sequence length is actually contributing to the improved model generation we will increase the sequential length further to see for improvements.

12.1.26 We should notice a point here that the model is overfitting from the 2nd epoch itself. Hence, we will try to increase drop out to 0.4 to reduce overfitting in our next model.

```
# Evaluate perplexity on the validation set
val_perplexity = calculate_perplexity(model3_7_loaded, val_loader)
print(f"Validation Perplexity: {val_perplexity:.2f}")

# Evaluate entropy on the generated text
entropy = calculate_entropy(generated_text)
print(f"Entropy of Generated Text: {entropy:.2f}")
```

```
Validation Perplexity: 43.89
Entropy of Generated Text: 3.92
```

12.1.27 We got the perplexity of this as 44.89 which is more than previous model and entropy of 3.92, which is almost indifferent from the previous model.

Model3_8:

```
Model3_8:
Vocab size : 1000
Sequence length : 100
embed_size = 512
hidden_size = 1024
num_layers = 2
dropout_rate = 0.4
```

- Vocab size : 1000
- Sequence length : 100
- embed_size = 512

- hidden_size = 1024
- num_layers = 2
- dropout_rate = 0.4
- Number of epochs = 20
- Batch_size = 1024
- Learning rate = 0.001

12.1.28 In this model we try to increase the sequential length, as we saw some improvement in increasing the sequence length previously. We are hoping that an increase in sequence length to 100 will give a better result.

```
# Build a BPE tokenizer
tokenizer = Tokenizer(BPE(unk_token="<unk>"))
trainer = BpeTrainer(
    vocab_size=1000,
    min_frequency=1,
    special_tokens=["<unk>", "<pad>", "<s>", "</s>"]
)
tokenizer.pre_tokenizer = Whitespace()

# Train the tokenizer
tokenizer.train_from_iterator([cleaned_text], trainer)

# Tokenize the text
encoded = tokenizer.encode(cleaned_text)
text_as_int = np.array(encoded.ids)

# Define sequence length and prepare input-output pairs
sequence_length = 100
sequences = []
targets = []
```

12.1.29 As discussed earlier, to reduce overfitting, we increase the dropout to 0.4 and keep the rest of the parameters as it is.

```
# Define model parameters
vocab_size = tokenizer.get_vocab_size()
embed_size = 512
hidden_size = 1024
num_layers = 2
dropout_rate = 0.4

model3_8 = LSTMModel3(vocab_size, embed_size, hidden_size, num_layers=num_layers, dropout=dropout_rate)

# Define loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer3_8 = optim.Adam(model3_8.parameters(), lr=0.001)

# Set device (GPU if available)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

```python
model_folder.mkdir(parents=True, exist_ok=True)  # Ensure the folder exists

# Define paths for saving the best and last epoch models
best_model_path = model_folder / "lstm_model3.8_best.pth"
final_model_path = model_folder / "lstm_model3.8_last_epoch.pth"

# Train the model
trained_model, best_model_state_dict, train_losses, val_losses = train_model(
    model=model3_8,
    train_loader=train_loader,
    val_loader=val_loader,
    criterion=criterion,
    optimizer=optimizer3_8,
    num_epochs=20,
    device=device,
    best_model_path=best_model_path,
    final_model_path=final_model_path
)

# Print the locations of the saved models
print(f"Last epoch model saved to {final_model_path}")
print(f"Best validation loss model saved to {best_model_path}")
```

```
Total number of trainable parameters in the model: 16233448
Epoch 1/20 --> Train Loss: 4.6213, Val Loss: 4.1366
Best model saved with validation loss: 4.1366
Epoch 2/20 --> Train Loss: 3.8201, Val Loss: 3.9103
Best model saved with validation loss: 3.9103
Epoch 3/20 --> Train Loss: 3.5920, Val Loss: 3.8242
Best model saved with validation loss: 3.8242
Epoch 4/20 --> Train Loss: 3.4523, Val Loss: 3.7802
Best model saved with validation loss: 3.7802
Epoch 5/20 --> Train Loss: 3.3491, Val Loss: 3.7542
Best model saved with validation loss: 3.7542
Epoch 6/20 --> Train Loss: 3.2677, Val Loss: 3.7553
Best model saved with validation loss: 3.7542
```

12.1.30 If we observe the train and validation loss, the validation loss stops improving after the 4th epoch and the model starts overfitting from the 2nd epoch itself. This suggests that maybe we should try with higher drop out and also that there is no point in increasing the number of epochs. Since the model was not improving, we stopped the model at 6th epoch, where the model at 4th epoch has the least validation loss and is hence saved as the best model for further inference.

```python
# Function to load the model
from pprint import pprint
def load_model(model_class, model_path, vocab_size, embed_size, hidden_size, num_layers, dropout_rate, device):
    model = model_class(vocab_size, embed_size, hidden_size, num_layers=num_layers, dropout=dropout_rate)
    model.load_state_dict(torch.load(model_path, map_location=device))
    model.to(device)
    model.eval()
    return model

# Load the best model
model_path = os.path.join(model_folder, "lstm_model3.8_best.pth")
model3_8_loaded = load_model(LSTMModel3, model_path, vocab_size, embed_size, hidden_size, num_layers, dropout_rate, device)

# Generate text using the loaded model
seed_text = "The king and kingdom".lower()  # Convert to lowercase
generated_text = generate_text(model3_8_loaded, tokenizer, seed_text, length=200, device=device)
pprint(generated_text)
```

12.1.31 Generation of sample text from the MODEL3_8.

```
('the king and king d om est ic and at im ic is m s of fe red for a du ll mil '
 'it ary ser vi ce ne cess ary to per m me to set ac c us to med after the es '
 'ca pe from napoleon came out from the v ill ag es and re turning to his s is '
 'ter where s he had di ed a h that here is dólokhov for the emperor s follow '
 's he said it is the last no on es li ved to ch an ge a good can not like my '
 'ran k to him but like a since re person al m and a v in ity here b en ni g '
 'sen cor re ct an im al to us he added your hon or hur ra h and inter ru pt '
 'ed russian the enemy is expe cted we can tell you to h ard er what is it it '
 'you know that our selves always end uring to fire to ch an ge this self and '
 'then have been rid icul ous that fo l ks are in g re en ser ving words un '
 'cle but to day')
```

12.1.32 If we look at the text generation, it is contextually relevant and creative in generation as compared to the other two models. Yet, we are not sure whether increase in sequence length is actually contributing to improved performance. To ensure this we will try out increasing the sequence length with higher vocab sizes as well.

```python
[ ] # Evaluate perplexity on the validation set
    val_perplexity = calculate_perplexity(model3_8_loaded, val_loader)
    print(f"Validation Perplexity: {val_perplexity:.2f}")

    # Evaluate entropy on the generated text
    entropy = calculate_entropy(generated_text)
    print(f"Entropy of Generated Text: {entropy:.2f}")
```

```
Validation Perplexity: 42.19
Entropy of Generated Text: 3.90
```

12.1.33 We got the perplexity of this as 42.19 and entropy of 3.90, both of which are again not much different from our previous models. With this we can come to a conclusion that maybe just increasing the sequence length is not contributing enough to improving the model performance(by seeing the generated text and the perplexity and entropy values). So we better try to focus on other hyper parameters like vocab size.

Model3_0:

```
Model3_0:
Vocab size : 3000
Sequence length : 25
embed_size = 256
hidden_size = 512
num_layers = 2
dropout_rate = 0.5
```

- Vocab size : 3000
- Sequence length : 25
- embed_size = 256
- hidden_size = 512
- num_layers = 2
- dropout_rate = 0.5
- Number of epochs = 20
- Batch_size = 1024
- Learning rate = 0.001

12.1.34 In this model we tried changing four parameters- Vocab size - 3000. Since we are increasing the vocab size we will try to decrease the embedding size to 256 as each token is more likely to have a unique vector, so a smaller embedding size might still suffice for capturing token relationships. Here, we are trying to strike an optimal balance between the two.

12.1.35 We have reduced the hidden size to 512 as we have increased the vocab to 3000. For the vocab size 3000, we again start with a sequence length of 25.

```
# Build a BPE tokenizer
tokenizer = Tokenizer(BPE(unk_token="<unk>"))
trainer = BpeTrainer(
    vocab_size=3000,
    min_frequency=1,
    special_tokens=["<unk>", "<pad>", "<s>", "</s>"]
)
tokenizer.pre_tokenizer = Whitespace()

# Train the tokenizer
tokenizer.train_from_iterator([cleaned_text], trainer)

# Tokenize the text
encoded = tokenizer.encode(cleaned_text)
text_as_int = np.array(encoded.ids)

# Define sequence length and prepare input-output pairs
sequence_length = 25
sequences = []
targets = []
```

12.1.36 We can also see that for the reason stated in the previous experiment (model overfitting) we increased the drop out value to 0.5.

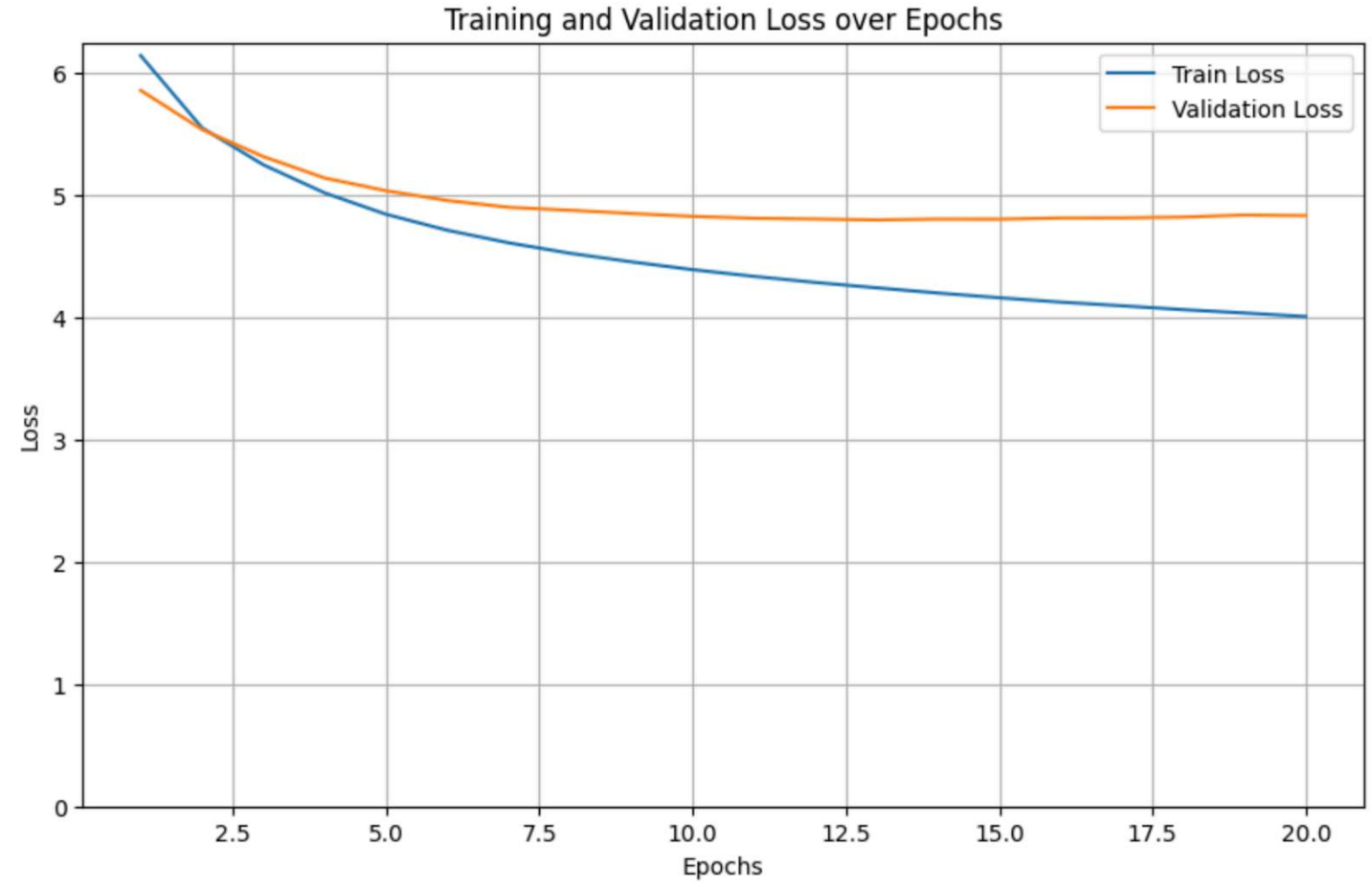
```
# Define model parameters
vocab_size = tokenizer.get_vocab_size()
embed_size = 256
hidden_size = 512
num_layers = 2
dropout_rate = 0.5

model3_0 = LSTMModel3(vocab_size, embed_size, hidden_size, num_layers=num_layers, dropout=dropout_rate)

# Define loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer3_0 = optim.Adam(model3_0.parameters(), lr=0.001)

# Set device (GPU if available)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

```
model_folder.mkdir(parents=True, exist_ok=True)  # Ensure the folder exists

# Define paths for saving the best and last epoch models
best_model_path = model_folder / "lstm_model3.0_best.pth"
final_model_path = model_folder / "lstm_model3.0_last_epoch.pth"

# Train the model
trained_model, best_model_state_dict, train_losses, val_losses = train_model(
    model=model3_0,
    train_loader=train_loader,
    val_loader=val_loader,
    criterion=criterion,
    optimizer=optimizer3_0,
    num_epochs=20,
    device=device,
    best_model_path=best_model_path,
    final_model_path=final_model_path
)

# Print the locations of the saved models
print(f"Last epoch model saved to {final_model_path}")
print(f"Best validation loss model saved to {best_model_path}")
```

```
Total number of trainable parameters in the model: 5985208
Epoch 1/20 --> Train Loss: 6.1440, Val Loss: 5.8596
Best model saved with validation loss: 5.8596
Epoch 2/20 --> Train Loss: 5.5529, Val Loss: 5.5392
Best model saved with validation loss: 5.5392
Epoch 3/20 --> Train Loss: 5.2506, Val Loss: 5.3174
Best model saved with validation loss: 5.3174
Epoch 4/20 --> Train Loss: 5.0204, Val Loss: 5.1425
Best model saved with validation loss: 5.1425
Epoch 5/20 --> Train Loss: 4.8457, Val Loss: 5.0382
Best model saved with validation loss: 5.0382
Epoch 6/20 --> Train Loss: 4.7149, Val Loss: 4.9585
Best model saved with validation loss: 4.9585
Epoch 7/20 --> Train Loss: 4.6122, Val Loss: 4.9027
Best model saved with validation loss: 4.9027
Epoch 8/20 --> Train Loss: 4.5267, Val Loss: 4.8784
Best model saved with validation loss: 4.8784
Epoch 9/20 --> Train Loss: 4.4565, Val Loss: 4.8523
Best model saved with validation loss: 4.8523
Epoch 10/20 --> Train Loss: 4.3925, Val Loss: 4.8284
Best model saved with validation loss: 4.8284
```

```
Best model saved with validation loss: 4.8284
Epoch 11/20 --> Train Loss: 4.3376, Val Loss: 4.8130
Best model saved with validation loss: 4.8130
Epoch 12/20 --> Train Loss: 4.2879, Val Loss: 4.8068
Best model saved with validation loss: 4.8068
Epoch 13/20 --> Train Loss: 4.2440, Val Loss: 4.7996
Best model saved with validation loss: 4.7996
Epoch 14/20 --> Train Loss: 4.2013, Val Loss: 4.8062
Best model saved with validation loss: 4.7996
Epoch 15/20 --> Train Loss: 4.1626, Val Loss: 4.8053
Best model saved with validation loss: 4.7996
Epoch 16/20 --> Train Loss: 4.1268, Val Loss: 4.8154
Best model saved with validation loss: 4.7996
Epoch 17/20 --> Train Loss: 4.0967, Val Loss: 4.8162
Best model saved with validation loss: 4.7996
Epoch 18/20 --> Train Loss: 4.0654, Val Loss: 4.8227
Best model saved with validation loss: 4.7996
Epoch 19/20 --> Train Loss: 4.0375, Val Loss: 4.8395
Best model saved with validation loss: 4.7996
Epoch 20/20 --> Train Loss: 4.0088, Val Loss: 4.8345
Best model saved with validation loss: 4.7996
```

12.1.37 If we observe the losses, the validation loss has decreased till 13th epoch (which is a good improvement) and increased again. The model started overfitting from the 3rd epoch. So, increasing the number of epochs will not help in this model.



```
# Function to load the model
def load_model(model_class, model_path, vocab_size, embed_size, hidden_size, num_layers, dropout_rate, d
    model = model_class(vocab_size, embed_size, hidden_size, num_layers=num_layers, dropout=dropout_rate
    model.load_state_dict(torch.load(model_path, map_location=device))
    model.to(device)
    model.eval()
    return model

# Load the best model
model_path = os.path.join(model_folder, "lstm_model3.0_best.pth")
model3_0_loaded = load_model(LSTMModel3, model_path, vocab_size, embed_size, hidden_size, num_layers,
                             dropout_rate, device)

# Generate text using the loaded model
seed_text = "The soldiers marched forward and at a".lower()  # Convert to lowercase
generated_text = generate_text(model3_0_loaded, tokenizer, seed_text, length=200, device=device)
pprint(generated_text)
```

12.1.38 Generation of sample text from the MODEL3_0.

```
('the soldiers mar ched forward and at a bound ful note and the prisoners fell '
 'asleep there was in an unknown and smoke f um bled and thirty cr ack ling of '
 'the soldiers looked at him and at once led up to the front officers now it '
 'was hearing everyone shouted the soldiers beside the prisoners and began '
 'shot ting it and ran to the wolf and putting the words an adjutant who were '
 'de fi n ated from that fl è ches beyond its st ream that had a close battle '
 'to the village where they had gone with the battalion for the hour the time '
 'he learned that the russian army were prepared for polit eness the fire were '
 'already still bus y and like a general galloped out of the d ev z z ling '
 'road s and e en galloped a i understand and stri king that appeared in such '
 'a state of great success and these some beginning the young man were bound '
 'out and too a fool in nesvítski he himself kill ed the pie ces for time as '
 'on bennigsen s three years before it was a great mass man eu vers everything '
 'tou l u ck asked a we')
```

12.1.39 If we look at the text generation, the model is able to generate more words correctly, i.e. without any unnecessary spaces. The reason can be that we have increased the vocab size. The text has a better grammar as compared to the previous models but has low coherency.

```
# Evaluate perplexity on the validation set
val_perplexity = calculate_perplexity(model3_0_loaded, val_loader)
print(f"Validation Perplexity: {val_perplexity:.2f}")

# Evaluate entropy on the generated text
entropy = calculate_entropy(generated_text)
print(f"Entropy of Generated Text: {entropy:.2f}")
```

```
Validation Perplexity: 120.79
Entropy of Generated Text: 4.01
```

12.1.40 If we see, the perplexity value is 120.79 and entropy is 4.01. Entropy has shown an improvement but perplexity has increased a lot. Maybe an increase in sequence length will provide some improvements.

12.1.41 As discussed earlier, we will try to increase the sequence length to see if there is any improvement.

Model3_1:

```
Model3_1:
Vocab size : 3000
Sequence length : 50
embed_size = 256
hidden_size = 512
num_layers = 2
dropout_rate = 0.5
```

- Vocab size : 3000
- Sequence length : 50
- embed_size = 256
- hidden_size = 512
- num_layers = 2
- dropout_rate = 0.5
- Number of epochs = 20
- Batch_size = 1024
- Learning rate = 0.001

12.1.42 In this model we have just changed a single parameter, i.e. the sequence length as 50.

```
# Build a BPE tokenizer
tokenizer = Tokenizer(BPE(unk_token="<unk>"))
trainer = BpeTrainer(
    vocab_size=3000,
    min_frequency=1,
    special_tokens=["<unk>", "<pad>", "<s>", "</s>"]
)
tokenizer.pre_tokenizer = Whitespace()

# Train the tokenizer
tokenizer.train_from_iterator([cleaned_text], trainer)

# Tokenize the text
encoded = tokenizer.encode(cleaned_text)
text_as_int = np.array(encoded.ids)

# Define sequence length and prepare input-output pairs
sequence_length = 50
sequences = []
targets = []
```

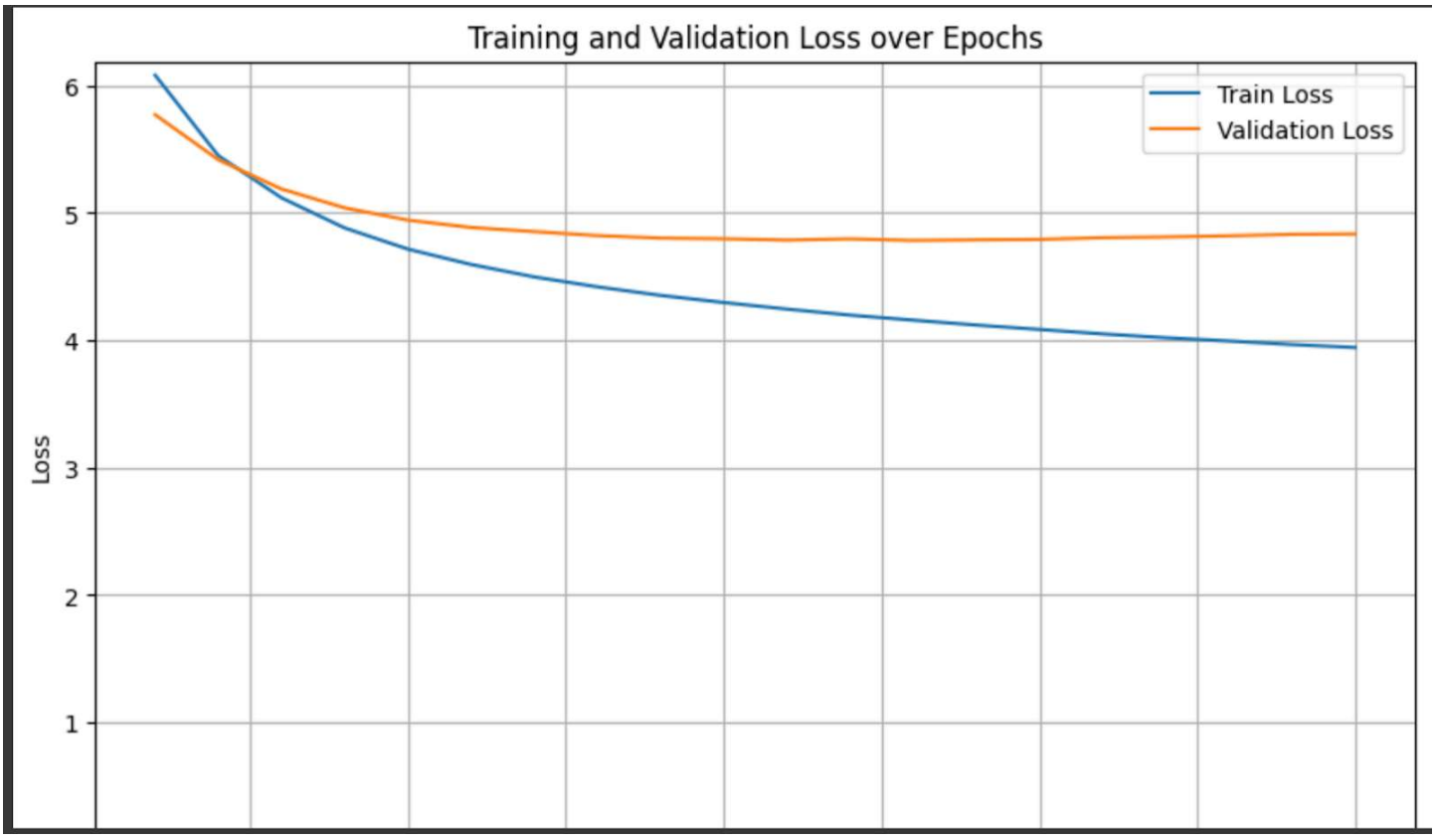
```python
# Define model parameters
vocab_size = tokenizer.get_vocab_size()
embed_size = 256
hidden_size = 512
num_layers = 2
dropout_rate = 0.5

model3_1 = LSTMModel3(vocab_size, embed_size, hidden_size, num_layers=num_layers, dropout=dropout_rate)

# Define loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer3_1 = optim.Adam(model3_1.parameters(), lr=0.001)

# Set device (GPU if available)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

```python
model_folder.mkdir(parents=True, exist_ok=True)  # Ensure the folder exists

# Define paths for saving the best and last epoch models
best_model_path = model_folder / "lstm_model3.1_best.pth"
final_model_path = model_folder / "lstm_model3.1_last_epoch.pth"

# Train the model
trained_model, best_model_state_dict, train_losses, val_losses = train_model(
    model=model3_1,
    train_loader=train_loader,
    val_loader=val_loader,
    criterion=criterion,
    optimizer=optimizer3_1,
    num_epochs=20,
    device=device,
    best_model_path=best_model_path,
    final_model_path=final_model_path
)

# Print the locations of the saved models
print(f"Last epoch model saved to {final_model_path}")
print(f"Best validation loss model saved to {best_model_path}")
```

```
Total number of trainable parameters in the model: 5985208
Epoch 1/20 --> Train Loss: 6.0814, Val Loss: 5.7713
Best model saved with validation loss: 5.7713
Epoch 2/20 --> Train Loss: 5.4495, Val Loss: 5.4176
Best model saved with validation loss: 5.4176
Epoch 3/20 --> Train Loss: 5.1189, Val Loss: 5.1882
Best model saved with validation loss: 5.1882
Epoch 4/20 --> Train Loss: 4.8822, Val Loss: 5.0400
Best model saved with validation loss: 5.0400
Epoch 5/20 --> Train Loss: 4.7164, Val Loss: 4.9440
Best model saved with validation loss: 4.9440
Epoch 6/20 --> Train Loss: 4.5956, Val Loss: 4.8862
Best model saved with validation loss: 4.8862
Epoch 7/20 --> Train Loss: 4.4979, Val Loss: 4.8536
Best model saved with validation loss: 4.8536
Epoch 8/20 --> Train Loss: 4.4198, Val Loss: 4.8224
Best model saved with validation loss: 4.8224
Epoch 9/20 --> Train Loss: 4.3530, Val Loss: 4.8029
Best model saved with validation loss: 4.8029
Epoch 10/20 --> Train Loss: 4.2966, Val Loss: 4.7973
Best model saved with validation loss: 4.7973
```

```
Epoch 11/20 --> Train Loss: 4.2452, Val Loss: 4.7872
Best model saved with validation loss: 4.7872
Epoch 12/20 --> Train Loss: 4.1981, Val Loss: 4.7965
Best model saved with validation loss: 4.7872
Epoch 13/20 --> Train Loss: 4.1581, Val Loss: 4.7845
Best model saved with validation loss: 4.7845
Epoch 14/20 --> Train Loss: 4.1197, Val Loss: 4.7890
Best model saved with validation loss: 4.7845
Epoch 15/20 --> Train Loss: 4.0854, Val Loss: 4.7921
Best model saved with validation loss: 4.7845
Epoch 16/20 --> Train Loss: 4.0504, Val Loss: 4.8061
Best model saved with validation loss: 4.7845
Epoch 17/20 --> Train Loss: 4.0200, Val Loss: 4.8118
Best model saved with validation loss: 4.7845
Epoch 18/20 --> Train Loss: 3.9946, Val Loss: 4.8203
Best model saved with validation loss: 4.7845
Epoch 19/20 --> Train Loss: 3.9666, Val Loss: 4.8323
Best model saved with validation loss: 4.7845
Epoch 20/20 --> Train Loss: 3.9437, Val Loss: 4.8350
Best model saved with validation loss: 4.7845
```

12.1.43 If we see the losses, the validation loss has started decreasing till 13th epoch and increasing thereafter, which is a good indication. The model starts overfitting from the 3rd epoch. Therefore , increasing the number of epochs will be of no use.




```python
# Load the best model
model_path = os.path.join(model_folder, "lstm_model3.1_best.pth")
model3_1_loaded = load_model(LSTMModel3, model_path, vocab_size, embed_size, hidden_size, num_layers, dropout_rate, device)

# Generate text using the loaded model
seed_text = "The soldiers marched forward and at a".lower()  # Convert to lowercase
generated_text = generate_text(model3_1_loaded, tokenizer, seed_text, length=200, device=device)
pprint(generated_text)
```

12.1.44 Generation of sample text from the MODEL3_1.

```
('the soldiers mar ched forward and at a re col lection of medi ences ly ha '
 'ped in approaching no way this was over thought natásha prince andrew knew '
 'that neither it that had been of the previ ous day he saw a con o ul w ry de '
 'empt y splendid surprise of gr att le and d um bling who in the mid st of '
 'their ad vice was moved by the spee ch by att ending fo od and he had set '
 'again to ob serve one another had just been quite that time it looked and st '
 'ream ed with capt ure cut off the sha g gy of the object and for a long time '
 'the artillery when brought again a visit napoleon brought back in front of '
 'them in front but instead of any single right of the french post to arri ve '
 'by the french but those ga ther ing enormous cossack napoleon did not know '
 'his own retreat well only rose to his army the marsh als ways the emperor '
 'forget ed that the must remain s of the bri ef part of place destro y all '
 'the marsh als thought of the action or rece ive forces that produ')
```

12.1.45 If we closely see the generated text, the text generated with 25 sequence length has less unnecessary spaces as compared to that of 50 sequence length, though it has better context and grammar. So increase in sequence length has resulted in some improvements but also has more fragmented words.

```
# Evaluate perplexity on the validation set
val_perplexity = calculate_perplexity(model3_1_loaded, val_loader)
print(f"Validation Perplexity: {val_perplexity:.2f}")

# Evaluate entropy on the generated text
entropy = calculate_entropy(generated_text)
print(f"Entropy of Generated Text: {entropy:.2f}")
```

```
Validation Perplexity: 119.00
Entropy of Generated Text: 4.03
```

12.1.46 We can with a high vocab size of 5000 and lower frequency, less than 25, try a model. Also, another reason is the vanishing gradient issue which might be more when we take higher sequence lengths. Hence we plan to take a vocab size of 5000 and sequence length of 15 instead of 25 as an experiment.

Model3_9:

```
Model3_9:
Vocab size : 5000
Sequence length : 15
embed_size = 256
hidden_size = 1024
num_layers = 2
dropout_rate = 0.5
min_frequency = 5
```

- Vocab size : 5000
- Sequence length : 15
- embed_size = 256
- hidden_size = 1024
- num_layers = 2
- dropout_rate = 0.5
- Number of epochs = 20
- Batch_size = 1024
- Learning rate = 0.001

12.1.47 Here, in this model we have tried to experiment with a high vocab size of 5000 and a lower sequence length of 15.

12.1.48 We are trying to get better output, hence we try to increase the hidden size to 1024 as a final experiment in BPE tokenized models. If we find any good results in this model we will try to proceed with BPE, else, we will try out improving the character level generation model.

```
# Build a BPE tokenizer
tokenizer = Tokenizer(BPE(unk_token="<unk>"))
trainer = BpeTrainer(
    vocab_size=5000,
    min_frequency=1,
    special_tokens=["<unk>", "<pad>", "<s>", "</s>"]
)
tokenizer.pre_tokenizer = Whitespace()

# Train the tokenizer
tokenizer.train_from_iterator([cleaned_text], trainer)

# Tokenize the text
encoded = tokenizer.encode(cleaned_text)
text_as_int = np.array(encoded.ids)

# Define sequence length and prepare input-output pairs
sequence_length = 15
sequences = []
targets = []

for i in range(len(text_as_int) - sequence_length):
    sequences.append(text_as_int[i:i + sequence_length])
    targets.append(text_as_int[i + sequence_length])

sequences = np.array(sequences)
targets = np.array(targets)
```

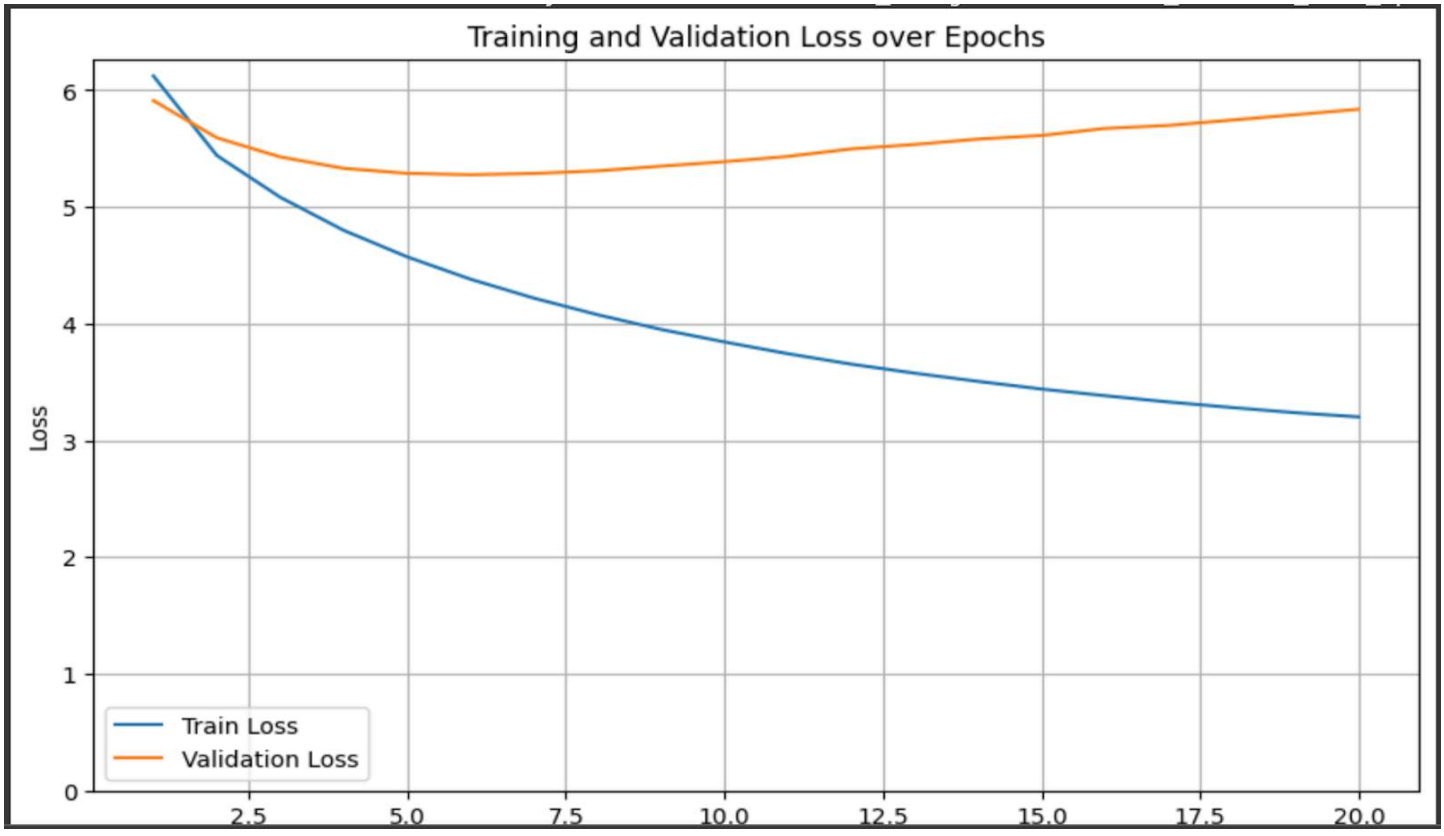
```python
# Define model parameters
vocab_size = tokenizer.get_vocab_size()
embed_size = 256
hidden_size = 1024
num_layers = 2
dropout_rate = 0.5

model3_9 = LSTMModel3(vocab_size, embed_size, hidden_size, num_layers=num_layers, dropout=dropout

# Define loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer3_9 = optim.Adam(model3_9.parameters(), lr=0.001)

# Set device (GPU if available)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

```python
model_folder.mkdir(parents=True, exist_ok=True)  # Ensure the folder exists

# Define paths for saving the best and last epoch models
best_model_path = model_folder / "lstm_model3.9_best.pth"
final_model_path = model_folder / "lstm_model3.9_last_epoch.pth"

# Train the model
trained_model, best_model_state_dict, train_losses, val_losses = train_model(
    model=model3_9,
    train_loader=train_loader,
    val_loader=val_loader,
    criterion=criterion,
    optimizer=optimizer3_9,
    num_epochs=20,
    device=device,
    best_model_path=best_model_path,
    final_model_path=final_model_path
)

# Print the locations of the saved models
print(f"Last epoch model saved to {final_model_path}")
print(f"Best validation loss model saved to {best_model_path}")
```

```
Total number of trainable parameters in the model: 20052872
Epoch 1/20 --> Train Loss: 6.1221, Val Loss: 5.9118
Best model saved with validation loss: 5.9118
Epoch 2/20 --> Train Loss: 5.4411, Val Loss: 5.5929
Best model saved with validation loss: 5.5929
Epoch 3/20 --> Train Loss: 5.0830, Val Loss: 5.4286
Best model saved with validation loss: 5.4286
Epoch 4/20 --> Train Loss: 4.7995, Val Loss: 5.3309
Best model saved with validation loss: 5.3309
Epoch 5/20 --> Train Loss: 4.5725, Val Loss: 5.2882
Best model saved with validation loss: 5.2882
Epoch 6/20 --> Train Loss: 4.3809, Val Loss: 5.2762
Best model saved with validation loss: 5.2762
Epoch 7/20 --> Train Loss: 4.2187, Val Loss: 5.2873
Best model saved with validation loss: 5.2762
Epoch 8/20 --> Train Loss: 4.0771, Val Loss: 5.3095
Best model saved with validation loss: 5.2762
Epoch 9/20 --> Train Loss: 3.9511, Val Loss: 5.3499
Best model saved with validation loss: 5.2762
Epoch 10/20 --> Train Loss: 3.8438, Val Loss: 5.3876
Best model saved with validation loss: 5.2762
```

```
Epoch 11/20 --> Train Loss: 3.7428, Val Loss: 5.4329
Best model saved with validation loss: 5.2762
Epoch 12/20 --> Train Loss: 3.6535, Val Loss: 5.4976
Best model saved with validation loss: 5.2762
Epoch 13/20 --> Train Loss: 3.5759, Val Loss: 5.5355
Best model saved with validation loss: 5.2762
Epoch 14/20 --> Train Loss: 3.5059, Val Loss: 5.5819
Best model saved with validation loss: 5.2762
Epoch 15/20 --> Train Loss: 3.4400, Val Loss: 5.6127
Best model saved with validation loss: 5.2762
Epoch 16/20 --> Train Loss: 3.3832, Val Loss: 5.6731
Best model saved with validation loss: 5.2762
Epoch 17/20 --> Train Loss: 3.3291, Val Loss: 5.6985
Best model saved with validation loss: 5.2762
Epoch 18/20 --> Train Loss: 3.2817, Val Loss: 5.7446
Best model saved with validation loss: 5.2762
Epoch 19/20 --> Train Loss: 3.2375, Val Loss: 5.7908
Best model saved with validation loss: 5.2762
Epoch 20/20 --> Train Loss: 3.2015, Val Loss: 5.8374
Best model saved with validation loss: 5.2762
```

12.1.49 If we look at the losses, the training loss has decreased continuously, but the validation loss decreased until 6th epoch and increased thereafter. The model starts to overfit from the beginning itself.



```
# Function to load the model
def load_model(model_class, model_path, vocab_size, embed_size, hidden_size, num_layers, dropout_rate
    model = model_class(vocab_size, embed_size, hidden_size, num_layers=num_layers, dropout=dropout_
    model.load_state_dict(torch.load(model_path, map_location=device))
    model.to(device)
    model.eval()
    return model

# Load the best model
model_path = os.path.join(model_folder, "lstm_model3.9_best.pth")
model3_9_loaded = load_model(LSTMModel3, model_path, vocab_size, embed_size, hidden_size, num_layers,

# Generate text using the loaded model
seed_text = "The king and kingdom".lower()  # Convert to lowercase
generated_text = generate_text(model3_9_loaded, tokenizer, seed_text, length=200, device=device)
pprint(generated_text)
```

```
('the soldiers marched forward and at a distance the french disper sed just '
 'dismounted and shouting wi ping down hosp ically and looked about him and '
 'leave them the princess smoo thed herself her eyes fixed on her or set him '
 'on earth there so on entering the whole room from her left over a tro y ka '
 'gloom and hussars on the other morning a blu sh get a battery something rose '
 'to the dust and puff ing with that cost news that they were remained first '
 'then dessalles you know the girls had died of the holy a denísovs young man '
 'of u ries or my benefactor who is a fool of me all michael ivánovich was he '
 'so kutúzovs a sense of retreating a emperor and foreign er who ventured to '
 'enter the last danger more entrusted to his mother to the emperor or having '
 'given him every glo ves of life and such anxiety both reproach s on such own '
 'grief he said in a fr ist voice princess mary making her hand and gave a '
 'bass all his cheeks and that someone was thought in his drawing room as the '
 'next prince andrew lived at last had some in definite time with')
```

12.1.50 A look at the text generation, we see that there are still unnecessary spaces in the words persisting. But in the BPE models performed till now, this model has better coherence, grammar, and contextual relevance.

```
# Evaluate perplexity on the validation set
val_perplexity = calculate_perplexity(model3_9_loaded, val_loader)
print(f"Validation Perplexity: {val_perplexity:.2f}")

# Evaluate entropy on the generated text
entropy = calculate_entropy(generated_text)
print(f"Entropy of Generated Text: {entropy:.2f}")
```

```
Validation Perplexity: 190.29
Entropy of Generated Text: 4.02
```

12.1.51 Though we have tried so many models, we still have persisting issues especially in spacing. We can improve this by trying out higher vocab sizes and again different sequence lengths. Having said all this, the model generated at character level is much better in terms of coherence, grammar, creativity and other aspects.

12.1.52 For this project it is better to continue with the character level text generation rather than trying out further experiments in BPE. We may have to carry out other experiments like increasing vocab size, different sequence lengths, different hidden layers, different drop out values etc which is computationally intensive.

13 Model Training: Section 6: Finetuning the best model

13.1 Based on our analysis, the character-level text generation model has demonstrated better performance compared to the BPE tokenized models. Building upon these promising results, we are implementing additional optimization strategies to further enhance the model's performance.

13.2 To achieve this, we are adding dynamic learning rate adjustment through the **ReduceLROnPlateau scheduler**. The scheduler monitors the validation loss and reduces the learning rate when the model's improvement stagnates.

13.3 The 'train_model' function is modified to incorporate the **ReduceLROnPlateau scheduler**. The ReduceLROnPlateau has the below configuration:

13.3.1 **Patience value of 2 epochs**: The scheduler waits for two epochs without improvement before adjusting the learning rate
13.3.2 **Reduction factor of 0.5**: When triggered, the learning rate is halved.
13.3.3 **Minimum learning rate threshold of 1e-5**: This prevents the learning rate from becoming too small, which could lead to negligible/no updates.

```
import matplotlib.pyplot as plt
import torch

def train_model(model, train_loader, val_loader, criterion, optimizer, num_epochs, device, scheduler=None, min_lr=None):
    """
    Train the given model and visualize training and validation loss over epochs.

    Args:
        model: The PyTorch model to train.
        train_loader: DataLoader for training data.
        val_loader: DataLoader for validation data.
        criterion: Loss function.
        optimizer: Optimizer for training.
        num_epochs: Number of training epochs.
        device: Device to train on (e.g., "cpu" or "cuda").
        scheduler: Learning rate scheduler (optional).
        min_lr: Minimum learning rate below which the scheduler should not reduce (optional).

    Returns:
        model: Trained model (last epoch).
        best_model_state_dict: State dictionary of the model with the best validation loss.
        train_losses: List of training losses for each epoch.
        val_losses: List of validation losses for each epoch.
    """
    # Move model to the device
    model.to(device)
```

```python
    # Calculate the number of trainable parameters
    total_params = sum(p.numel() for p in model.parameters() if p.requires_grad)
    print(f"Total number of trainable parameters in the model: {total_params}")

    # Lists to store losses
    train_losses = []
    val_losses = []

    # Best validation loss and model
    best_val_loss = float('inf')
    best_model_state_dict = None

    # Training loop
    for epoch in range(num_epochs):
        model.train()
        train_loss = 0
        for inputs, targets in train_loader:
            inputs, targets = inputs.to(device), targets.to(device)

            # Forward pass
            outputs, _ = model(inputs)
            loss = criterion(outputs, targets)

            # Backward pass
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            train_loss += loss.item()
```

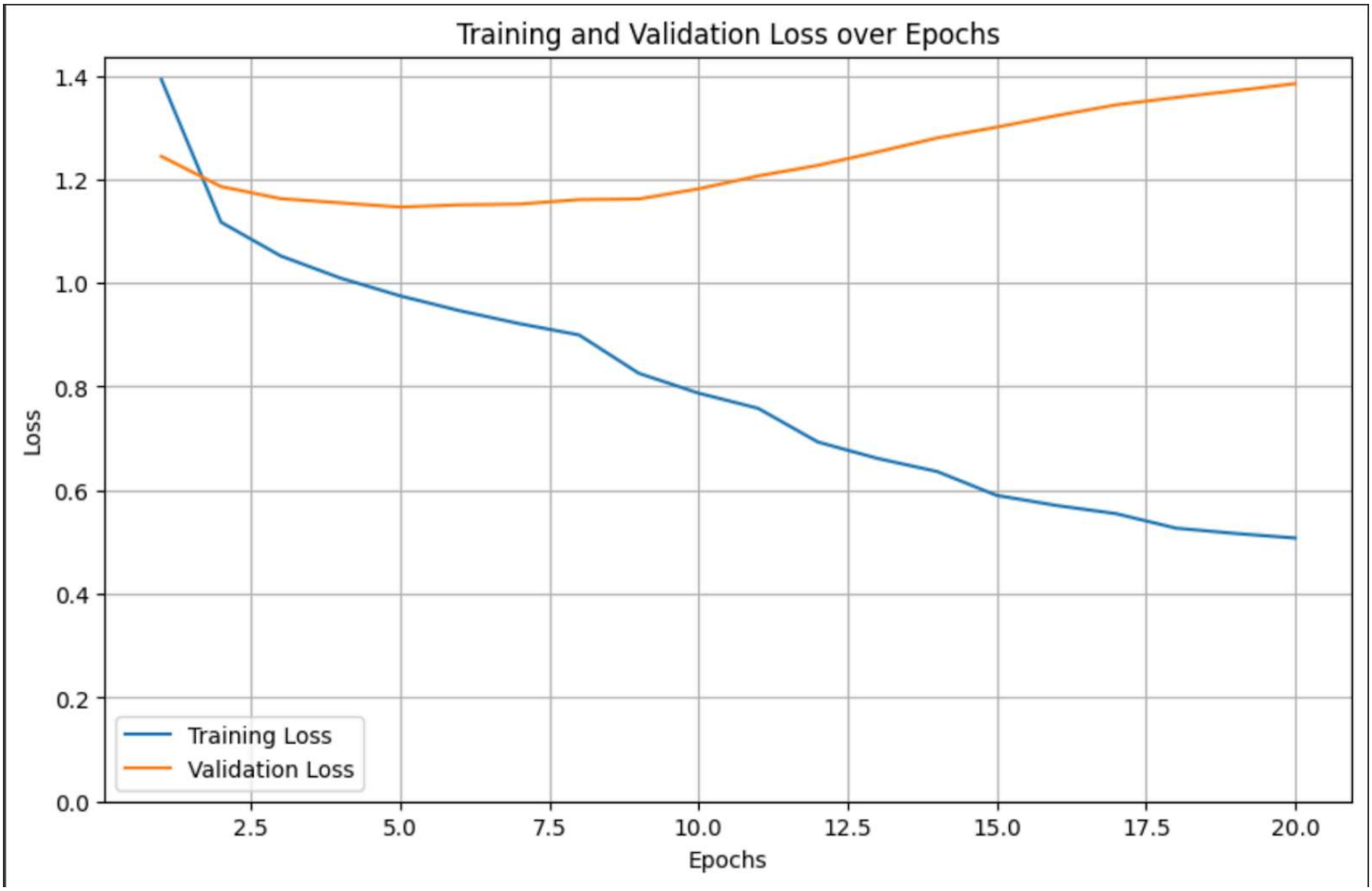
```python
        # Validation loop
        model.eval()
        val_loss = 0
        with torch.no_grad():
            for inputs, targets in val_loader:
                inputs, targets = inputs.to(device), targets.to(device)
                outputs, _ = model(inputs)
                loss = criterion(outputs, targets)
                val_loss += loss.item()

        # Calculate average losses for the epoch
        train_loss /= len(train_loader)
        val_loss /= len(val_loader)

        # Update learning rate scheduler if provided
        if scheduler:
            if isinstance(scheduler, torch.optim.lr_scheduler.ReduceLROnPlateau):
                scheduler.step(val_loss)  # Pass validation loss as the metric
            else:
                scheduler.step()  # For other types of schedulers

            # Ensure the learning rate does not drop below min_lr
            for param_group in optimizer.param_groups:
                if min_lr is not None and param_group['lr'] < min_lr:
                    param_group['lr'] = min_lr
```

```
    # Store losses
    train_losses.append(train_loss)
    val_losses.append(val_loss)

    # Check for best validation loss
    if val_loss < best_val_loss:
        best_val_loss = val_loss
        best_model_state_dict = model.state_dict()  # Save the state dict of the best model

    print(f"Epoch {epoch+1}/{num_epochs}, Train Loss: {train_loss:.4f}, Val Loss: {val_loss:.4f},
              LR: {optimizer.param_groups[0]['lr']:.6f}")

# Plot the training and validation losses
plt.figure(figsize=(10, 6))
plt.plot(range(1, num_epochs + 1), train_losses, label='Training Loss')
plt.plot(range(1, num_epochs + 1), val_losses, label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.ylim(0)  # Start y-axis from 0
plt.title('Training and Validation Loss over Epochs')
plt.legend()
plt.grid(True)
plt.show()

# Return the model and the best model state dict
return model, best_model_state_dict, train_losses, val_losses
```

```
# Define model parameters
vocab_size = len(chars)  # Character-level vocabulary size
embed_size = 128  # Embedding size
hidden_size = 1024  # Hidden size of the LSTM

model2_2 = LSTMModel2(vocab_size, embed_size, hidden_size)

# Define loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer2_2 = optim.Adam(model2_2.parameters(), lr=0.001)

# Scheduler to decrease learning rate when validation loss plateaus
# Define ReduceLROnPlateau scheduler
scheduler = ReduceLROnPlateau(optimizer2_2, mode='min', factor=0.5, patience=2, min_lr=1e-5, verbose=True)

# Train the model
trained_model2_2, best_model_state_dict2_2, train_losses, val_losses = train_model(
    model=model2_2,
    train_loader=train_loader,
    val_loader=val_loader,
    criterion=nn.CrossEntropyLoss(),
    optimizer=optimizer2_2,
    num_epochs=20,
    device=torch.device('cuda' if torch.cuda.is_available() else 'cpu'),
    scheduler=scheduler,
    min_lr=1e-5  # Minimum learning rate
)
```

```
# Save the last epoch model
last_epoch_model_path = model_folder / "lstm_model2_2.pth"
torch.save(trained_model2_2.state_dict(), last_epoch_model_path)
print(f"Last epoch model saved to {last_epoch_model_path}")

# Save the best model based on validation loss
best_model_path = model_folder / "lstm_model2_2_best.pth"
torch.save(best_model_state_dict2_2, best_model_path)
print(f"Best validation loss model saved to {best_model_path}")
```

```
Total number of trainable parameters in the model: 4780975
Epoch 1/20, Train Loss: 1.3926, Val Loss: 1.2443, LR: 0.001000
Epoch 2/20, Train Loss: 1.1172, Val Loss: 1.1859, LR: 0.001000
Epoch 3/20, Train Loss: 1.0522, Val Loss: 1.1627, LR: 0.001000
Epoch 4/20, Train Loss: 1.0095, Val Loss: 1.1545, LR: 0.001000
Epoch 5/20, Train Loss: 0.9751, Val Loss: 1.1463, LR: 0.001000
Epoch 6/20, Train Loss: 0.9464, Val Loss: 1.1506, LR: 0.001000
Epoch 7/20, Train Loss: 0.9215, Val Loss: 1.1521, LR: 0.001000
Epoch 8/20, Train Loss: 0.8997, Val Loss: 1.1608, LR: 0.000500
Epoch 9/20, Train Loss: 0.8258, Val Loss: 1.1620, LR: 0.000500
Epoch 10/20, Train Loss: 0.7875, Val Loss: 1.1815, LR: 0.000500
Epoch 11/20, Train Loss: 0.7582, Val Loss: 1.2065, LR: 0.000250
Epoch 12/20, Train Loss: 0.6932, Val Loss: 1.2268, LR: 0.000250
Epoch 13/20, Train Loss: 0.6615, Val Loss: 1.2527, LR: 0.000250
Epoch 14/20, Train Loss: 0.6362, Val Loss: 1.2798, LR: 0.000125
Epoch 15/20, Train Loss: 0.5901, Val Loss: 1.3007, LR: 0.000125
Epoch 16/20, Train Loss: 0.5708, Val Loss: 1.3229, LR: 0.000125
Epoch 17/20, Train Loss: 0.5549, Val Loss: 1.3436, LR: 0.000063
Epoch 18/20, Train Loss: 0.5270, Val Loss: 1.3578, LR: 0.000063
Epoch 19/20, Train Loss: 0.5167, Val Loss: 1.3708, LR: 0.000063
Epoch 20/20, Train Loss: 0.5079, Val Loss: 1.3846, LR: 0.000031
```



13.4 By analyzing the training and validation losses over epochs, we observe that in the initial phases(epochs 1-7,LR=0.001), the model showed a good convergence. As the learning rate reduced to 0.0005, the validation loss started increasing slowly leading to significant deterioration in the validation performance over the next few epochs even with smaller learning rates.

13.5 For further experiments in future, we can try different reduction factors,early stopping etc. to increase the performance. Before we decide if we want to continue with finetuning, lets see the generated output of the best model achieved in this process.

```
# Define model parameters
model_path = model_folder / "lstm_model2_2_best.pth"
vocab_size = len(chars)
embed_size = 128
hidden_size = 1024
num_layers = 1
dropout_rate = 0

model2_2 = load_model(LSTMModel2, model_path, vocab_size, embed_size, hidden_size, num_layers, dropout_rate, device)
```

```
seed_text = "The soldiers marched forward and at a".lower()  # Convert to lowercase
generated_text = generate_text(model2_2, seed_text, length=200)
pprint(generated_text)
```

```
('the soldiers marched forward and at a silver spoon the regiment had been '
 'sent to see the commander in chief who was talking about and begin the first '
 'time the first sleigh driven his horse and rode away with a smile of '
 'pleasure at the si')
```

```
# Evaluate perplexity on the validation set
val_perplexity = calculate_perplexity(model2_2, val_loader)
print(f"Validation Perplexity: {val_perplexity:.2f}")

# Evaluate entropy on the generated text
entropy = calculate_entropy(generated_text)
print(f"Entropy of Generated Text: {entropy:.2f}")
```

```
Validation Perplexity: 3.96
Entropy of Generated Text: 3.96
```

13.6 While the quantitative performance metrics is almost in the same range as our base model without the learning rate scheduler, manual evaluation of the generated text shows enhanced coherence.This improvement in text coherence, despite similar numerical metrics, highlights that in case of language model evaluation, traditional loss metrics may not fully capture all aspects of model performance.

13.7 Though the character-level model showed promising performance it is still exhibiting common text generation issues including not creative,nonsensical outputs,out of context sentence formation and sometimes repetitive patterns when generating longer sequences. To address these limitations, we implemented and evaluated a multi-strategy decoding approach combining temperature sampling, top-k sampling, and beam search.

13.8 We decided to implement the combined strategy because using each strategy on its own had few limitations as below:

1. **Temperature Sampling**

- Controls randomness in the next token selection process
- Higher temperatures (>1.0) increase randomness and creativity
- Lower temperatures (<1.0) make selections more deterministic
- **When used alone:** While good at introducing variety, it may still produce incoherent sequences due to unrestricted sampling from the entire vocabulary

2. **Top-k Sampling**

- Restricts token selection to the k most likely next tokens
- Prevents selection of highly improbable tokens

- **When used alone:** Improves local coherence but may still struggle with maintaining global narrative consistency.

3. **Beam Search**

- Maintains multiple possible sequences and selects the most probable complete sequence
- Ensures global coherence across the generated text
- **When used alone:** Often produces overly deterministic and generic outputs lacking creative variation.

13.9 Below is our new generate_text function which generates text based on our provided combination of temperature,top_k value and beam width.

```python
import torch
import torch.nn.functional as F
import numpy as np

def generate_text(model, seed_text, vocab_size, idx_to_char, char_to_idx, max_length=100,
                  temperature=1.0, top_k=5, beam_width=3):
    """
    Generates text using the LSTM model with temperature, top-k sampling, and beam search.

    Args:
        model: The LSTM model.
        seed_text: The initial string to seed the text generation.
        vocab_size: Total vocabulary size.
        idx_to_char: Mapping from indices to characters.
        char_to_idx: Mapping from characters to indices.
        max_length: Maximum length of the generated sequence.
        temperature: Controls randomness in predictions.
        top_k: Number of top probable tokens to sample from.
        beam_width: Number of beams to use in beam search.

    Returns:
        Generated text as a string.
    """
    model.eval()  # Set model to evaluation mode
    device = next(model.parameters()).device
    # Convert seed text to input indices
    input_indices = [char_to_idx[char] for char in seed_text if char in char_to_idx]
    input_tensor = torch.tensor(input_indices, dtype=torch.long, device=device).unsqueeze(0)

    generated_text = seed_text  # Initialize with seed text
```

```
    for _ in range(max_length - len(seed_text)):
        new_beams = []
        for text, input_tensor, cum_log_prob in beams:
            # Forward pass through the model
            with torch.no_grad():
                output = model(input_tensor)  # Output: (1, seq_len, vocab_size)
                logits = output[0]  # Extract the first element (logits) from the tuple
                logits = logits / temperature  # Apply temperature scaling
                probs = F.softmax(logits, dim=-1)  # Convert logits to probabilities

                # Apply top-k sampling
                top_k_probs, top_k_indices = torch.topk(probs, top_k, dim=-1)
                top_k_probs = top_k_probs.squeeze().cpu().numpy()
                top_k_indices = top_k_indices.squeeze().cpu().numpy()

                # Normalize top-k probabilities
                top_k_probs /= top_k_probs.sum()

                # Generate new beams
                for i in range(len(top_k_probs)):
                    idx = top_k_indices[i]
                    prob = top_k_probs[i]
                    next_char = idx_to_char[idx]

                    # Update beam with new character
                    new_text = text + next_char
                    new_input_tensor = torch.cat([input_tensor, torch.tensor([[idx]],
                                                                             device=device)], dim=1)
                    new_log_prob = cum_log_prob + np.log(prob)  # Accumulate log-probability

                    new_beams.append((new_text, new_input_tensor, new_log_prob))
```

```
        # Keep top `beam_width` beams
        beams = sorted(new_beams, key=lambda x: x[2], reverse=True)[:beam_width]

    # Return the most probable sequence
    return beams[0][0]
```

13.10 We experimented four different hyperparameter combinations for two of our best performing models (character-level base model and the model obtained from learningrate scheduler experiment), systematically varying:

1. Temperature values
2. Top-k thresholds
3. Beam width

13.11The goal was to identify configurations that maximize both coherence and creativity in the generated text.

13.12 Below are the combination of parameters we experimented with:

```
param_array = [{
    "max_length":200,
    "temperature":0.8,
    "top_k":5,
    "beam_width":3
},
{
    "max_length":200,
    "temperature": 1.2,
    "top_k" : 10,
    "beam_width" :5
},
            {
    "max_length":200,
    "temperature": 1.5,
    "top_k" : 25,
    "beam_width" :3
},
{
    "max_length":200,
    "temperature": 2,
    "top_k" : 30,
    "beam_width" :5
}
]
```

```
seed_text = "the soldiers marched forward and at a".lower()

def hyperParameter_GenerateText(model, param_array, bpe=False, tokenizer=None, idx_to_char=None,
                                char_to_idx=None, vocab_size=None, device="cpu",max_length=None):
    """
    Generate text for multiple hyperparameter settings.

    Args:
        model: The LSTM model.
        param_array: A list of dictionaries containing hyperparameter combinations.
        bpe: Boolean indicating if BPE tokenizer is used.
        tokenizer: Tokenizer instance (required if bpe=True).
        idx_to_char: Mapping from indices to characters (required if bpe=False).
        char_to_idx: Mapping from characters to indices (required if bpe=False).
        vocab_size: Total vocabulary size (required if bpe=False).
        device: Device to run the model on ("cpu" or "cuda").
    """
    for param in param_array:
        if max_length is not None:
          max_length = max_length
        else:
          max_length = param["max_length"]
        temperature = param["temperature"]
        top_k = param["top_k"]
        beam_width = param["beam_width"]

        updatedParams = {
            "max_length": max_length,
            "temperature": temperature,
            "top_k": top_k,
            "beam_width": beam_width
```

```
    }

    if bpe:
        # BPE-based text generation
        if tokenizer is None:
            raise ValueError("Tokenizer must be provided for BPE generation!")

        generated_text = generate_text_forBPE(
            model=model,          # Your LSTM model
            seed_text=seed_text, # Seed text
            tokenizer=tokenizer, # Tokenizer instance
            max_length=max_length,  # Max generated length
            temperature=temperature,
            top_k=top_k,
            beam_width=beam_width,
            device=device
        )
    else:
        # Character-level text generation
        if idx_to_char is None or char_to_idx is None or vocab_size is None:
            raise ValueError("idx_to_char, char_to_idx, and vocab_size must be provided")

        generated_text = generate_text(
            model=model,          # Your LSTM model
            seed_text=seed_text, # Seed text
            vocab_size=vocab_size,
            idx_to_char=idx_to_char,
            char_to_idx=char_to_idx,
            max_length=max_length,
            temperature=temperature,
            top_k=top_k,
            beam_width=beam_width
        )

    print(f"Generated Text for param: {updatedParams}")
    print(generated_text)
    val_perplexity = calculate_perplexity(model2_1, val_loader)
    print(f"Validation Perplexity: {val_perplexity:.2f}")

    entropy = calculate_entropy(generated_text)
    print(f"Entropy of Generated Text: {entropy:.2f}")
    print("-" * 80)  # Separator for readability
```

13.13 Experiment with character-level base model:

```
# Define model parameters
model_path = model_folder / "lstm_model2_1_best.pth"
vocab_size = len(chars)
embed_size = 128
hidden_size = 1024
num_layers = 1
dropout_rate = 0

model2_1 = load_model(LSTMModel1, model_path, vocab_size, embed_size, hidden_size, num_layers, dropout_rate, device)
```

```
hyperParameter_GenerateText(model = model2_1,
                            param_array = param_array,
                            bpe=False,
                            idx_to_char=index_to_char,
                            char_to_idx=char_to_index,
                            vocab_size=vocab_size,
                            device=device)
```

```
Generated Text for param: {'max_length': 200, 'temperature': 0.8, 'top_k': 5, 'beam_width': 3}
('the soldiers marched forward and at anna pávlovnas reception and the count '
 'was sent to her and that her fathers head was speaking to her that she was '
 'not to blame for her and to whom he had been told ')
Validation Perplexity: 3.42
Entropy of Generated Text: 3.86
--------------------------------------------------------------------------------
Generated Text for param: {'max_length': 200, 'temperature': 1.2, 'top_k': 10, 'beam_width': 5}
('the soldiers marched forward and at anna pávlovnas reception and the '
 'countess had been told that she had been told that there was nothing for '
 'herself and that her father would not be able to know what')
Validation Perplexity: 3.42
Entropy of Generated Text: 3.91
--------------------------------------------------------------------------------
Generated Text for param: {'max_length': 200, 'temperature': 1.5, 'top_k': 25, 'beam_width': 3}
('the soldiers marched forward and at anna mikháylovna to the countess and the '
 'countess had been told that there was nothing for herself and that her '
 'father would have said that he would not be able to ')
Validation Perplexity: 3.42
Entropy of Generated Text: 3.92
--------------------------------------------------------------------------------
Generated Text for param: {'max_length': 200, 'temperature': 2, 'top_k': 30, 'beam_width': 5}
('the soldiers marched forward and at anna mikháylovnas household had been '
 'told that it was impossible for herself and that he would never remember how '
 'he had never before knowing that he would never un')
Validation Perplexity: 3.42
Entropy of Generated Text: 4.05
--------------------------------------------------------------------------------
```

13.14 Experiment with Learning Rate Scheduler model:

```
# Define model parameters
model_path = model_folder / "lstm_model2_2.pth"
vocab_size = len(chars)
embed_size = 128
hidden_size = 1024
num_layers = 1
dropout_rate = 0

model2_2 = load_model(LSTMModel1, model_path, vocab_size, embed_size, hidden_size, num_layers, dropout_rate, device)
```

```
Generated Text for param: {'max_length': 200, 'temperature': 0.8, 'top_k': 5, 'beam_width': 3}
('the soldiers marched forward and at a trot along the road to the right and '
 'the french soldiers were dragging away the soldiers shouting and straggling '
 'from the french colonel who was standing and the ')
Validation Perplexity: 3.42
Entropy of Generated Text: 3.89
--------------------------------------------------------------------------------
Generated Text for param: {'max_length': 200, 'temperature': 1.2, 'top_k': 10, 'beam_width': 5}
('the soldiers marched forward and at a trot along the highroad to the left of '
 'the road where the battle of borodinó was being dragged along the road to '
 'mozháysk three hundred rubles were heard from the')
Validation Perplexity: 3.42
Entropy of Generated Text: 3.93
--------------------------------------------------------------------------------
Generated Text for param: {'max_length': 200, 'temperature': 1.5, 'top_k': 25, 'beam_width': 3}
('the soldiers marched forward and at another time two men were retreating '
 'along the highroad to the left of the road through the forest there was '
 'nothing terrible on the contrary there was nothing for ')
Validation Perplexity: 3.42
Entropy of Generated Text: 3.79
--------------------------------------------------------------------------------
Generated Text for param: {'max_length': 200, 'temperature': 2, 'top_k': 30, 'beam_width': 5}
('the soldiers marched forward and at austerlitz next morning prince bagratión '
 'bowed his head without looking at rostopchín and denísov where have you '
 'heard it yourself said pierre without raising his e')
Validation Perplexity: 3.42
Entropy of Generated Text: 4.18
--------------------------------------------------------------------------------
```

13.15 By looking at the generated outputs for both models, we see that the learning rate model has generated creative text that sticks to the context. So lets analyze how the model has performed for different parameter combinations:

Combination 1
Parameters:

- Temperature: 0.8 (low)
- Top-k: 5 (restrictive)
- Beam width: 3 (moderate)

Result Analysis:

- Generated coherent military-themed text about soldiers marching
- Maintained narrative consistency

Combination 2
Parameters:

- Temperature: 1.2 (moderate)
- Top-k: 10 (moderate)
- Beam width: 5 (higher)

Result Analysis:

- Introduced more specific details (e.g., "battle of borodin", "mozhaysk")
- Wider beam search helped maintain coherence despite higher temperature

Combination 3
Parameters:

- Temperature: 1.5 (high)
- Top-k: 25 (permissive)
- Beam width: 3 (moderate)

Result Analysis:

- More creative scene construction with complex narrative elements
- Shows balance between creativity and coherence

Combination 4
Parameters:

- Temperature: 2.0 (very high)
- Top-k: 30 (very permissive)
- Beam width: 5 (higher)

13.16 **Result Analysis:**

- Introduced character dialogue and names ("pierre", "rostophin")
- Very creative output but still maintained narrative coherence due to wider beam width

13.17 **<u>Impact Analysis by Parameter</u>**

Temperature Impact

- Low (0.8): Produced straightforward, conservative text
- Moderate (1.2): Introduced more specific details
- High (1.5-2.0): Generated more creative and diverse outputs
- Higher temperatures consistently led to more varied vocabulary and complex narrative structures

Top-k Impact

- Restrictive (5): Limited vocabulary but ensured consistency
- Moderate (10): Balanced vocabulary diversity with coherence
- Permissive (25-30): Allowed for more creative word choices while maintaining context
- Larger k-values enabled more diverse storytelling elements when paired with higher temperatures

Beam Width Impact

- Width 3: Adequate for maintaining coherence with lower temperatures
- Width 5: Provided better coherence control with higher temperatures/top-k values
- Wider beams helped maintain narrative consistency even with more aggressive exploration parameters.

13.18 The implementation of combined decoding strategies significantly improved our model's text generation capabilities.

13.19 Overall, this strategy increased the model's contextual relevance, reduced repetition and maintained narrative coherence.

13.20 So after evaluating the generated texts, we decided to keep the below combination of hyperparameters and check the model robustness for different seed texts.

- Temperature: 1.2 (moderate)
- Top-k: 10 (moderate)
- Beam width: 5 (higher)

```python
def hyperseedText_generation(
    model,
    seed_texts,
    bpe=False,
    tokenizer=None,
    idx_to_char=None,
    char_to_idx=None,
    vocab_size=None,
    max_length=200,
    temperature=1.2,
    top_k=10,
    beam_width=5,
    device="cpu"
):
    """
    Generate text for a list of seed texts.

    Args:
        model: The LSTM model.
        seed_texts: A list of seed strings to generate text from.
        bpe: Boolean indicating if BPE tokenizer is used.
        tokenizer: Tokenizer instance (required if bpe=True).
        idx_to_char: Mapping from indices to characters (required if bpe=False).
        char_to_idx: Mapping from characters to indices (required if bpe=False).
        vocab_size: Total vocabulary size (required if bpe=False).
        max_length: Maximum length of the generated sequence.
        temperature: Controls randomness in predictions.
        top_k: Number of top probable tokens to sample from.
        beam_width: Number of beams to use in beam search.
        device: Device to run the model on ("cpu" or "cuda").
    """
```

```
    entropy_text = None  # To store the first generated text for entropy calculation

    # Generate text for each seed text
    for i, seed_text in enumerate(seed_texts):
        print(f"Generating text for seed {i + 1}: {seed_text}")

        if bpe:
            # BPE-based text generation
            if tokenizer is None:
                raise ValueError("Tokenizer must be provided for BPE generation!")

            generated_text = generate_text_forBPE(
                model=model,          # Your LSTM model
                seed_text=seed_text, # Current seed text
                tokenizer=tokenizer, # Tokenizer instance
                max_length=max_length,  # Max generated length
                temperature=temperature,
                top_k=top_k,
                beam_width=beam_width,
                device=device
            )
        else:
            # Character-level text generation
            if idx_to_char is None or char_to_idx is None or vocab_size is None:
                raise ValueError("idx_to_char, char_to_idx, and vocab_size must be provided")

            generated_text = generate_text(
                model=model,          # Your LSTM model
                seed_text=seed_text, # Current seed text
                vocab_size=vocab_size,
                idx_to_char=idx_to_char,
                char_to_idx=char_to_idx,
                max_length=max_length,
                temperature=temperature,
                top_k=top_k,
                beam_width=beam_width
            )

        # Print the generated text
        print(f"Generated Text for Seed {i + 1}:")
        pprint(generated_text)
        print("-" * 80)  # Separator for readability
```

```
seed_texts = [
    "war is not only a political but".lower(),
    "peace brings moments of joy and".lower(),
    "the king and the kingdom".lower(),
    "on the battlefield, the soldiers".lower(),
    "the meaning of life is questioned".lower()
]
hyperseedText_generation(model = model2_2,seed_texts = seed_texts, bpe=False,
                         idx_to_char=index_to_char, char_to_idx=char_to_index,
                         vocab_size=vocab_size, device=device)
```

```
Generating text for seed 1: war is not only a political but
Generated Text for Seed 1:
('war is not only a political but which is the same order thank god how much i '
 'thank you forever said prince andrew without looking at prince andrew as he '
 'looked at him in silence what is the matter with you my dear she said '
 'touching his eyes and turning to her mother and went out of the room without '
 'understanding what was going on before her and that her father would not be '
 'able to understand all that was going on around her and that her father '
 'would not be able to understand all that was going o')
--------------------------------------------------------------------------------
Generating text for seed 2: peace brings moments of joy and
Generated Text for Seed 2:
('peace brings moments of joy and enthusiasm we shall see the princess did not '
 'reply and went to the drawing room where he had been standing at the other '
 'side of the room prince andrew rode up to the general who was standing '
 'behind the countess and sónya and natásha went up to her mother and went out '
 'of the room and sat down on a sofa without undressing and immediately fell '
 'into her eyes and pressing her hands on the shoulder and without replying '
 'put him down from her hand and said something to he')
--------------------------------------------------------------------------------
Generating text for seed 3: the king and the kingdom
Generated Text for Seed 3:
('the king and the kingdom of heaven knows when the emperor alexander had met '
 'that morning on the twentyfourth of august prince bagratión was impossible '
 'because they went away without any preparation for what was happening in '
 'front of them were standing in the morning of the second of september prince '
 'andrew found himself at the beginning of the commander in chief and the '
 'russian army which had been destroyed by the emperors headquarters in the '
 'morning of the twentyfourth of august at that moment ')
--------------------------------------------------------------------------------
```

```
Generating text for seed 4: on the battlefield, the soldiers
Generated Text for Seed 4:
('on the battlefield, the soldiers of the sunshine merged in their hands they '
 'could not forget the sight of the french army which there was nothing for '
 'him to relate to the french and that the russians had not been foreseen '
 'everything could be descried on them and that they would have been necessary '
 'for him to receive the army to defend moscow to the emperor alexander '
 'napoleon sat down at the other side of the road from borodinó to the right '
 'side of the bridge and the officer ran out into the porc')
--------------------------------------------------------------------------------
Generating text for seed 5: the meaning of life is questioned
Generated Text for Seed 5:
('the meaning of life is questioned he said to himself as he looked at him in '
 'silence what is the matter with you my dear said prince vasíli turning to '
 'prince andrew with a smile that she did not understand what was going on '
 'around her and that her father would not be able to understand all that was '
 'going on around her and that her father would not be able to understand all '
 'that was going on around her and that her father would not be able to '
 'understand all that was going on around her and that he')
--------------------------------------------------------------------------------
```

13.21 From the gerated outputs, we can observe that our model was able to be:

- Consistently perform across various input contexts.
- Maintain coherence regardless of starting point.
- Ability to generate diverse outputs.