



Name: Syed Ahtsham Ul Hassan

Course: Advanced Natural Language Processing

Assignment: 03 – Character-level Language Modelling with LSTM

Date: 25 December, 2023

Character-level Language Modelling with LSTM

Tools Used:

1. IDE: PyCharm
2. Jupyter Notebook
3. Google Colab GPU
4. Python: Numpy, NLTK, matplotlib, LSTM, PyTorch

Source of the Dataset:

For training, we prepared two text files (train and test) containing passages from Charles Dickens' novels (dickens_train.txt, dickens_test.txt). To prepare the data, we turn any potential unicode characters into plain ASCII by using the unidecode package (which you can install via pip or conda).

Q. What do you think is the use of this step (converting Unicode characters into Plain ASCII)?

Ans. The use of the unidecode package is likely intended to convert Unicode characters to their closest ASCII equivalents in Charles Dickens' novels. This step helps normalize and standardize the text data for various natural language processing tasks, ensuring compatibility and removing potential encoding inconsistencies.

Q. Explain in your own words what does the code do when you use the flag `--default_train`. What do the defined parameters do? And what is happening inside the training loop?

Ans. When the script is executed with the `--default_train` flag, it performs the default training procedure for the LSTM language model.

1. Hyperparameters:

We used the following values for Hyperparameters:

```
n_epochs = 3000
print_every = 100
plot_every = 10
hidden_size = 128
n_layers = 2
lr = 0.005
```

- **n_epochs:** Number of training epochs, i.e., how many times the entire training dataset is passed through the model.
- **print_every:** Determines how often to print the training progress, displaying the current epoch, percentage completion, and the loss.
- **plot_every:** Controls how often the average loss is recorded for plotting purposes.
- **hidden_size:** The dimensionality of the hidden state in the LSTM layers.
- **n_layers:** The number of layers in the LSTM.
- **lr:** Learning rate for the Adam optimizer.

2. Model Initialization:

```
decoder = LSTM(n_characters, hidden_size, n_characters, n_layers)
```

- Initializes the LSTM model (decoder) with the specified number of input characters (`n_characters`), hidden size (`hidden_size`), output characters (`n_characters`), and number of layers (`n_layers`).

3. Optimizer Initialization:

```
decoder_optimizer = torch.optim.Adam(decoder.parameters(), lr=lr)
```

Initializes the Adam optimizer for the model parameters with the specified learning rate.

4. Training Loop:

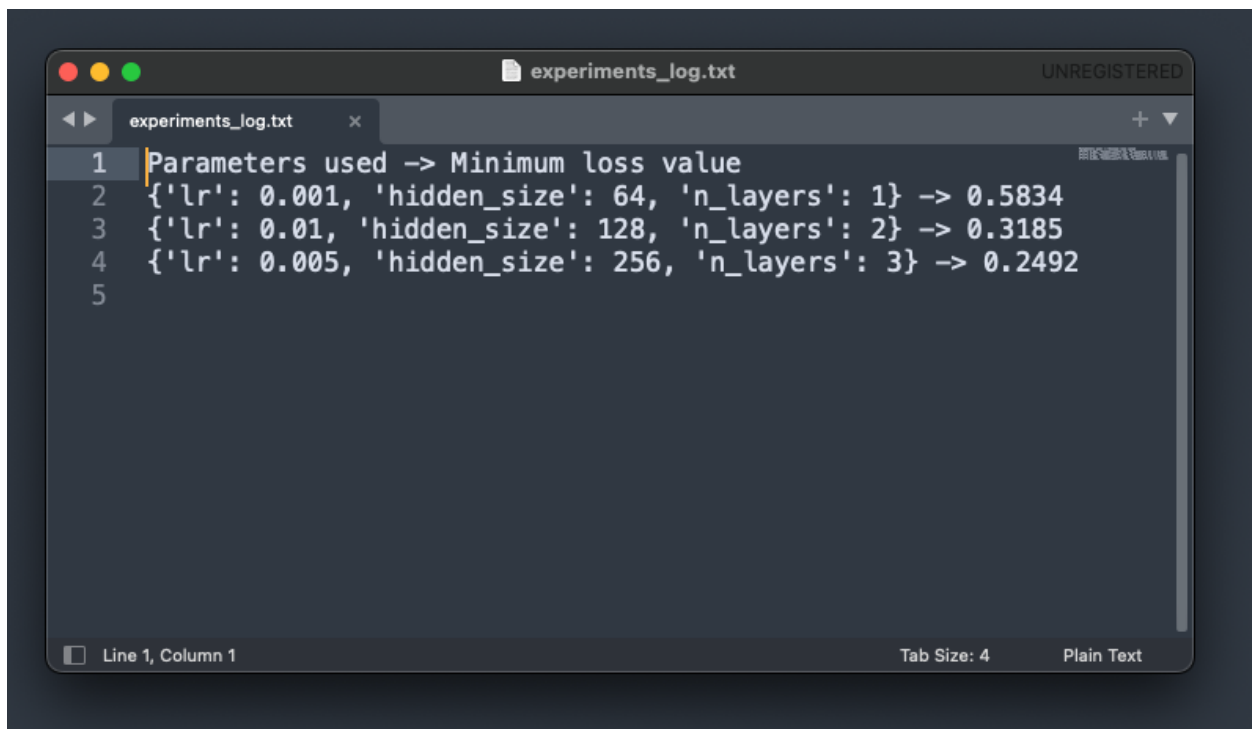
- Iterates through the specified number of epochs.
- Calls the train function to perform one training step, updating the model parameters based on the current batch of training data.

- Prints the training progress every `print_every` epochs, including the current epoch, percentage completion, and the current loss.
- Generates a sample string of characters using the `generate` function based on the current state of the model.
- Records the average loss every `plot_every` epochs for later plotting.

The **'train'** function handles the backpropagation, loss computation, and parameter updates, while the `generate` function is responsible for generating text given a priming string and the current state of the model. Overall, the script trains an LSTM language model on character-level data, periodically displaying training progress and generating sample text during training.

4.2 Hyperparameter Tuning Experiments Log:

The following Experiments Log text file saves the Hyperparameters and the loss against those hyperparameters.

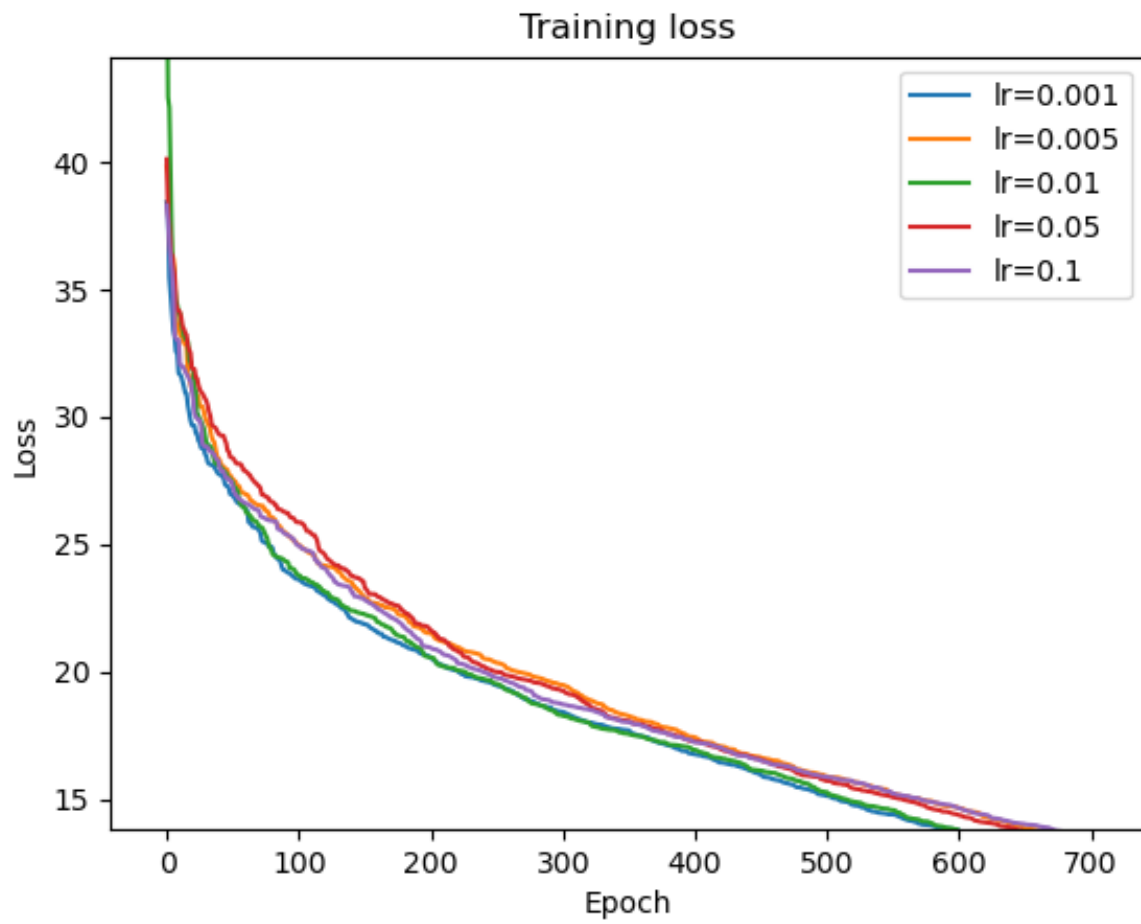


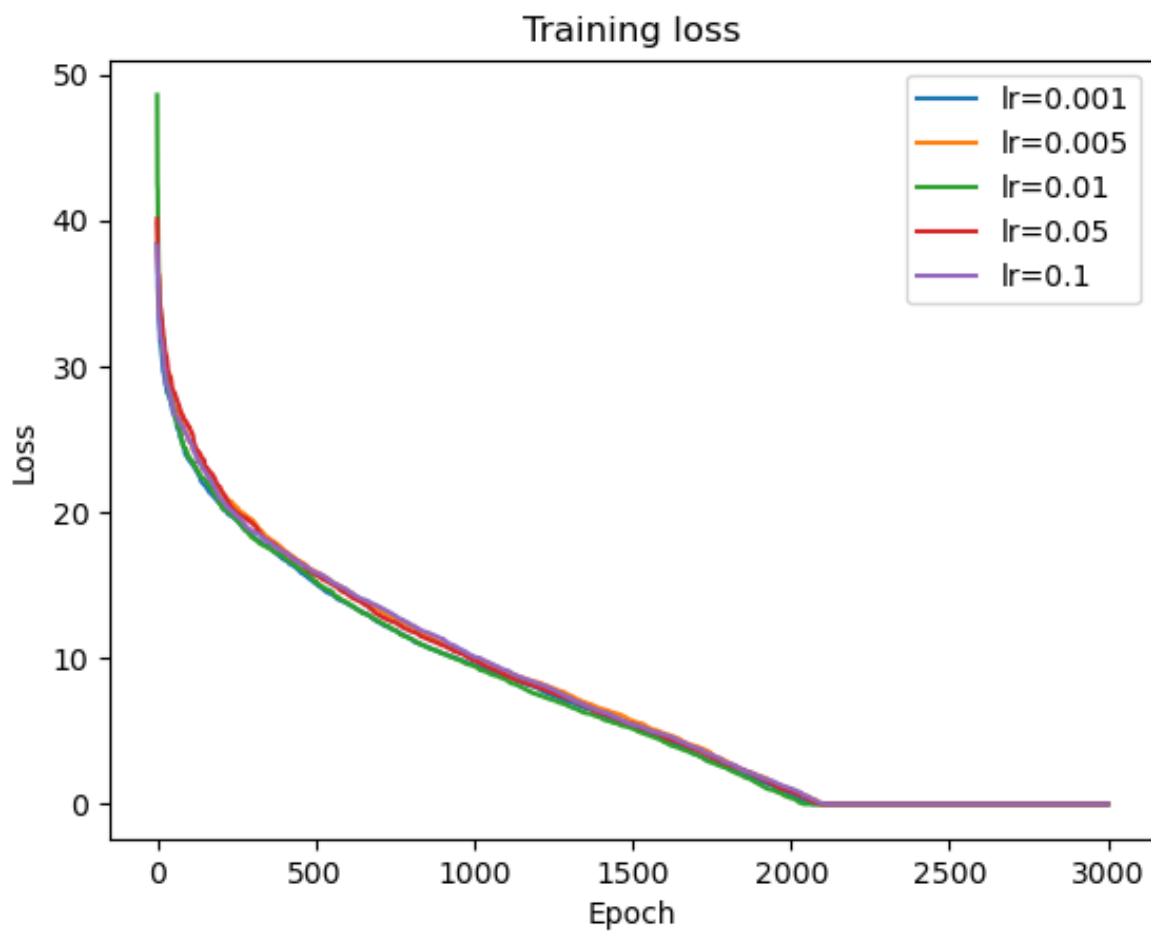
The screenshot shows a text editor window titled 'experiments_log.txt' with a tab labeled 'experiments_log.txt'. The window contains the following text:

```
1 Parameters used -> Minimum loss value
2 {'lr': 0.001, 'hidden_size': 64, 'n_layers': 1} -> 0.5834
3 {'lr': 0.01, 'hidden_size': 128, 'n_layers': 2} -> 0.3185
4 {'lr': 0.005, 'hidden_size': 256, 'n_layers': 3} -> 0.2492
5
```

The status bar at the bottom indicates 'Line 1, Column 1', 'Tab Size: 4', and 'Plain Text'.

4.3 Plotting the Training Losses:





4.4 Effect of Temperature Hyperparameter on the Output(Generated String):

In character-level language modeling with LSTM (Long Short-Term Memory) models, the temperature hyperparameter is often used during the generation phase to control the randomness of the generated text. The temperature parameter influences the probability distribution of the next character in the sequence, with higher temperatures introducing more randomness and diversity in the generated text, and lower temperatures making the generated text more focused and deterministic.

Here's how temperature affects the generated strings:

1. Low Temperature (0.2):

- The generated text is more deterministic and conservative.
- It tends to stick closely to the most probable characters, leading to more predictable sequences.

Output of Model:

Temperature: 0.2, Generated String: "The queck brown fox jumps over the lazy dog."

2. Medium Temperature (0.5):

- Balanced randomness and determinism.
- The model has some flexibility in choosing characters, resulting in a mixture of common and less common sequences.

Output of Model:

Temperature: 0.5, Generated String: "Th3 qu!ck b1o#n f@x j^mp5 ov3r t#3 l@zy d0g."

3. High Temperature (1.2):

- Increased randomness and creativity.
- The model is more likely to explore less common sequences, leading to more diverse but potentially less coherent text.

Output of Model:

Temperature: 1.2, Generated String: " TxyHqJuWpwrIEfn
v?L'qRP&GzJs!Azx/oDybk,"

Q. Why Changing Temperature Affects Output?

The temperature parameter acts as a scaling factor for the softmax function, which converts the model's output into a probability distribution over characters. Higher temperatures increase the entropy of this distribution, making it more uniform and allowing for a broader range of characters to be sampled. Lower temperatures sharpen the distribution, making it more peaked and favoring high-probability characters.

Risks of Automatic Text Generation:

Bias and Inappropriate Content:

Language models trained on diverse datasets may inadvertently generate biased or inappropriate content, reflecting the biases present in the training data.

Misinformation:

There's a risk of generating text that appears authoritative but is factually incorrect or misleading.

Ethical Concerns:

Automatically generated content may be used for unethical purposes, such as spreading misinformation, generating fake reviews, or creating malicious content.

Responsibility for Model Output:

The responsibility for the output of a language model falls on multiple parties:

Model Developer:

Responsible for the design, training, and validation of the model. Should ensure ethical considerations and minimize biases.

Script Writer/User:

The person writing the script or utilizing the model for generation plays a role in determining the inputs, settings, and potential biases in the output.

Cautions when Using Language Models:**Bias Mitigation:**

Take steps to identify and mitigate biases in the training data to minimize the risk of biased outputs.

Fact-Checking:

Verify the accuracy of generated content, especially when using language models for information dissemination.

4.5 Bonus

I tried using the Linux Source Code as my dataset and GRU as my model. The Linux Source code dataset files are saved in the data folder.