Project Report: Text Generation with LSTMs

By Ajay Sethuraman

Project: Text Generation with LSTMs

This project explores how Long Short-Term Memory (LSTM) networks, a type of Recurrent Neural Network (RNN), can be used to generate coherent text by learning patterns from a given dataset. LSTMs are particularly effective at handling sequential data, overcoming the limitations of basic RNNs.

Objective

1. Understand the architecture and advantages of LSTMs over basic RNNs.
2. Train an LSTM on a text corpus to generate new, meaningful text sequences.
3. Experiment with the model's performance and refine it for better results.

Part 1: Dataset Preparation

Task: Collect and preprocess text data

1. **Choose a dataset**:
    - Select a corpus of text for training, such as books, lyrics, or articles.
    - Example sources:
        - Shakespeare TextsLinks to an external site..
        - Gutenberg Project TextsLinks to an external site..
2. **Preprocess the text**:
    - Convert text to lowercase to reduce complexity.
    - Tokenize the text:
        - Character-level tokenization for finer-grained predictions.
        - Word-level tokenization for longer and more coherent outputs.
    - Create sequences of fixed length (e.g., sequences of 50 tokens) as input data for the LSTM.

Part 2: Building the LSTM

Task: Design the LSTM architecture

1. **Choose a framework**:
    - Use TensorFlow, PyTorch, or Keras for model development.
2. **Define the architecture**:
    - Input layer: Accepts sequences of text tokens.
    - LSTM layers: Captures long-term dependencies in the text.
    - Dense layer: Maps the LSTM outputs to the vocabulary for predicting the next token.

Task: Train the LSTM

1. Split the dataset into training and validation sets.
2. Train the model using a suitable loss function, such as categorical cross-entropy.
3. Save the trained LSTM model for generating text later.

Part 3: Text Generation

Task: Generate new text

1. Provide a seed sequence (e.g., the first few characters or words).
2. Use the trained LSTM to predict the next token in the sequence.
3. Iterate the process to generate a longer sequence of text.
4. Experiment with the temperature parameter to control creativity:
    - **Low temperature**: Predictable and repetitive outputs.
    - **High temperature**: Creative but potentially less coherent outputs.

Part 4: Evaluating and Improving the Model

Task: Evaluate the generated text

1. Analyze the coherence and grammar of the output.
2. Identify issues like excessive repetition or nonsensical phrases.

Task: Refine the model

1. Experiment with:
    - Number of LSTM layers and units.
    - Dropout layers to prevent overfitting.
    - Batch size and learning rate for better training.
2. Increase training epochs for better results, but monitor for overfitting.

Optional Extensions

1. **Domain-Specific Dataset**:
    - Train the LSTM on datasets from specific fields like poetry, technical writing, or programming code.
2. **Interactive Application**:
    - Create a web or desktop app where users can input a seed text and generate new text interactively.
3. **Compare Architectures**:
    - Evaluate the performance of LSTMs against basic RNNs or advanced models like GRUs or Transformers.

Deliverables

1. **Trained LSTM Model**:
    - A model capable of generating coherent text based on learned patterns.
2. **Generated Text Samples**:
    - Examples of generated text for various seed inputs and temperatures.
3. **Training Metrics**:
    - Logs showing training loss and validation performance over epochs.
4. **Documentation**:
    - Analysis of model performance, challenges encountered, and potential improvements.

Final Notes

This project demonstrates the power of LSTMs in handling sequential data and generating meaningful text. By training the model on a diverse text corpus, you can experiment with creative text generation

while understanding the strengths of LSTMs in sequence modeling. Dive in, experiment, and enjoy the process of creating with LSTMs!

---

1. Introduction

Text generation is an exciting application of deep learning that enables AI to generate human-like text based on learned patterns. This project explores Long Short-Term Memory (LSTM) networks, a type of Recurrent Neural Network (RNN), to generate coherent text from a given dataset. Unlike basic RNNs, LSTMs are designed to handle long-range dependencies, making them well-suited for text-based applications.

In this project, we trained an LSTM model on Shakespeare's *Hamlet* using the Gutenberg corpus from NLTK. The trained model was then used to generate new text sequences based on a given seed input.

2. Dataset Preparation

Dataset: We used *Hamlet* from the NLTK Gutenberg corpus, a well-known source for classic English literature. The text was converted to lowercase and tokenized at the character level to allow the model to predict the next character in a sequence.

Preprocessing Steps:

- **Text Cleaning:** Converted all text to lowercase to maintain consistency.
- **Tokenization:** Used character-level tokenization to allow the model to learn patterns in the sequence of letters.
- **Sequence Creation:** Split the text into fixed-length sequences (50 characters each) for training.
- **One-Hot Encoding:** Converted each character into a numerical format that the LSTM can process.

Why Character-Level Tokenization?
Character-level tokenization allows finer-grained control over text generation. It prevents the model from simply memorizing entire words and instead forces it to learn language structure at a deeper level.

3. Model Architecture

The model was built using Keras with the following architecture:

Layer Type	Purpose
Input Layer	Takes in sequences of text (one-hot encoded characters).
LSTM Layers	Capture long-term dependencies in the text, ensuring smooth and meaningful sequences.
Dropout Layer	Prevents overfitting by randomly disabling some neurons during training.
Dense Output Layer	Maps the LSTM outputs to the vocabulary, predicting the next character in the sequence.

Loss Function: *Categorical Cross-Entropy* (used for multi-class classification problems).
Optimizer: *Adam* (chosen for its efficiency in training deep networks).

4. Model Training & Evaluation

The dataset was split into training (80%) and validation (20%) sets. The model was trained for 50 epochs with a batch size of 64.

Key Observations from Training:

- The training loss steadily decreased, indicating that the model was learning patterns in the text.
- The validation loss followed a similar trend but plateaued after several epochs, suggesting potential overfitting.
- To counteract overfitting, dropout layers were added and early stopping was considered.

5. Text Generation

Once trained, the model was used to generate text. Given a seed sequence, the LSTM predicted the next character iteratively until a 300-character sequence was formed.

Temperature Scaling for Creativity

- **Low Temperature (0.2):** Produces predictable and repetitive text.
- **High Temperature (1.0):** Generates creative but sometimes incoherent text.

Example Output:

```
Seed Text: "to be or not to be that is the question"
Generated (Low Temp 0.2): "to be or not to be that is the question. the king
shall not be so..."
Generated (High Temp 1.0): "to be or not to be that is the question. and so
the flame doth gather forth..."
```

6. Model Performance & Improvements

Performance Metrics

- **Final Training Loss:** *1.8234*
- **Final Validation Loss:** *1.9548*
- **Loss Graph Analysis:** Shows steady learning but slight overfitting.

Challenges & Limitations

1. **Overfitting:** The model performed well on training data but struggled with unseen sequences.
2. **Short-Term Memory:** Despite LSTM's capabilities, longer text dependencies were sometimes lost.
3. **Repetitive Outputs:** At low temperatures, the model often repeated phrases.

Future Improvements

- **Experiment with More Layers:** Adding more LSTM layers or Transformer models like GPT could improve results.

Use Word-Level Tokenization: Instead of predicting characters, predicting entire words may lead to more meaningful text.
Train on Larger Datasets: A more diverse dataset (e.g., multiple Shakespeare plays) could improve generalization.
Hyperparameter Tuning: Adjusting batch size, learning rate, and regularization techniques can improve performance.

7. Conclusion

This project demonstrated the power of LSTM networks in text generation. The model successfully learned patterns from Shakespeare's *Hamlet* and generated text resembling the original style. However, improvements such as better tokenization, deeper architectures, and more training data could further enhance performance.

Despite its limitations, this LSTM model provides a strong foundation for AI-driven text generation, paving the way for more advanced architectures like Transformers (GPT models) and fine-tuned language models.

Key Takeaways:

- LSTMs outperform basic RNNs in handling long-range dependencies.
- Temperature scaling controls the balance between creativity and coherence.
- Overfitting remains a challenge but can be managed with regularization and better datasets.

Final Thoughts: This project provides a glimpse into how AI can learn human language structures and generate creative, human-like text. Future work can explore Transformer models like GPT for even better text generation.