Next Word Prediction using LSTM

---

1. Introduction

1.1 Problem Statement

The objective of this project is to build a predictive text system that suggests the next word(s) based on the input provided by the user. Such systems have applications in predictive typing, autocomplete features in search engines, and virtual assistants.

1.2 Project Objective

- Implement a neural network model using LSTM to predict the next word in a given text sequence.
- Utilize a classic English text dataset for training the model.
- Build a user interface to accept input and display the generated sentence.

---

2. Dataset Description

2.1 Dataset Source

The dataset used for this project is a text file (`1661-0.txt`) containing the full text of a classic novel.

2.2 Data Statistics

- **Total Words**: *Calculated dynamically from the script.*
- **Content Overview**: The dataset is read as a single text file, split into sentences, and tokenized for model input.

---

3. Methodology

3.1 Preprocessing

3.1.1 Tokenization

The dataset is tokenized using Keras's `Tokenizer` class to convert text into numerical sequences:

- Each word is assigned a unique index.
- Punctuation is ignored during tokenization.

3.1.2 Input Sequence Creation

For each sentence, tokenized sequences are generated. For example:

- Sentence: "I love machine learning"
- Sequences:
  - [I, love]
  - [I, love, machine]
  - [I, love, machine, learning]

These sequences are padded using `pad_sequences` to ensure uniform input length for the model.

3.2 Model Architecture

3.2.1 Embedding Layer

An embedding layer converts word indices into dense vectors of fixed size (100 dimensions in this case).

3.2.2 LSTM Layer

A Long Short-Term Memory (LSTM) layer with 150 units captures dependencies and patterns in the sequential data.

3.2.3 Dense Output Layer

The Dense layer uses a softmax activation function to predict the next word's probability distribution over the vocabulary.

---

4. Implementation

4.1 Data Preprocessing

- **Tokenization**: `Tokenizer.fit_on_texts()`
- **Sequence Padding**: `pad_sequences()`

4.2 Model Definition

- **Embedding**: Converts word indices to embeddings of size 100.
- **LSTM**: Captures sequence dependencies.

- **Dense Layer**: Outputs the probability of the next word.

```
python
Copy code
model = Sequential()
model.add(Embedding(len(tokenizer.word_index) + 1, 100,
input_length=max_length - 1))
model.add(LSTM(150))
model.add(Dense(len(tokenizer.word_index) + 1, activation='softmax'))
```

4.3 Training

- Optimizer: Adam
- Loss Function: Categorical Crossentropy
- Metric: Accuracy
- Training: `model.fit(x, y, epochs=60, batch_size=32)`

---

5. Results

5.1 Evaluation

The model was trained on the dataset for 60 epochs. Below is an example of predictions:

- **Input Word**: "machine"
- **Predicted Sentence**: "machine learning is the best"

5.2 Performance

The model provides accurate next-word predictions for common sequences. However, its accuracy may decline for rare or unseen words.

---

6. Use Cases

6.1 Predictive Typing

Helps users type faster by predicting the next word.

6.2 Autocomplete

Improves user experience in search engines.

6.3 Virtual Assistants

Enhances conversational capabilities by predicting user intents.

---

7. Conclusion

7.1 Summary

The project successfully implemented a next-word prediction system using LSTM. It is capable of generating contextually relevant sentences for input words.

7.2 Future Enhancements

- Use larger datasets for better generalization.
- Fine-tune the model with domain-specific text.
- Extend predictions to handle phrases and idiomatic expressions.

---

8. Appendix

8.1 Key Terms

- **Tokenization**: Breaking text into smaller units (tokens) such as words or sentences.
- **Embedding Layer**: Maps words to dense vectors of fixed size.
- **LSTM**: A type of recurrent neural network (RNN) capable of learning long-term dependencies.
- **Padding**: Adding zeros to sequences to ensure equal length.
- **Softmax**: A function that converts raw scores to probabilities.

8.2 Code Repository

The complete code can be found in the project folder.

---

9. References

- TensorFlow Documentation
- Keras API Documentation
- Project Dataset: Gutenberg Project