Report of Deep Learning for Natural Langauge Processing

Shanghua Li
zy2403112@buaa.edu.com

Abstract

This report explores the effectiveness of using LSTM and Transformer models for generating martial arts novels based on Jin Yong's works. The experiment involves training both models on a corpus of Jin Yong's novels, preprocessing the text data at the character level with sequence length of 500, and comparing the generated results across four experimental configurations. The study evaluates the performance of LSTM and Transformer models in terms of text coherence, semantic richness, and computational efficiency. Key findings include the comparison of training loss, generation quality, and the models' ability to capture long-term dependencies in the text.

Introduction

Text generation is a fundamental task in Natural Language Processing (NLP), with applications ranging from creative writing to automated content creation. This study focuses on generating martial arts novels using two deep learning architectures: LSTM (Long Short-Term Memory) and Transformer. The goal is to compare their performance in terms of text quality and training efficiency.

Key Research Questions:

How does the LSTM model perform in generating coherent martial arts text?

How does the Transformer model compare to LSTM in capturing long-term dependencies?

What are the computational trade-offs between LSTM and Transformer models for this task?

Methodology

Data Processing

Data Source: Jin Yong's novels stored in text files.

Preprocessing Steps:

Text Cleaning: Remove non-Chinese characters and punctuation.

Tokenization: Split the text into sequences of characters using a character-level tokenizer.

Sequence Generation: Create fixed-length sequences for training, where each input sequence predicts the next character.

Model Training

LSTM Model:

Embedding layer (128 dimensions).

Single LSTM layer (128 units).

Dense layer with softmax activation for character prediction.

Transformer Model:

Embedding layer (128 dimensions).

Multi-head attention layer (2 heads, key dimension 32).

Global average pooling followed by a dense layer with softmax activation.

Training and Evaluation

Training: Both models are trained using the Adam optimizer with sparse categorical cross-entropy loss.

Validation: 20% of the data is used for validation.

Text Generation: After training, the models generate text from seed phrases (e.g., "杨过", "郭靖").

Experimental Studies

The Python script implements the following workflow:

Importing Required Libraries

```
import os
import re
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Embedding, LSTM, Dense, Input, MultiHeadAttention, GlobalAveragePooling1D
from tensorflow.keras.optimizers import Adam
from tqdm.auto import tqdm
import matplotlib.pyplot as plt
```

Core Libraries

1.os: A Python standard library that provides functions for interacting with the operating system, such as file path operations and environment variable access.

2.re: Python's regular expression library, used for string matching and manipulation.

Data Processing & Scientific Computing

3.numpy(np): A powerful numerical computing library that provides efficient array operations and mathematical functions.

Deep Learning Frameworks

4.tensorflow (tf): A popular deep learning framework developed by Google.

5.tensorflow.keras: TensorFlow's high-level API that simplifies neural network construction:

Utility Tools

6.tqdm.auto: A progress bar library where the auto version automatically selects the appropriate display method for the current environment.

7.matplotlib.pyplot (plt): A data visualization library used for plotting charts and graphs.

Data Loading and Preprocessing

```python
# 数据加载和预处理
def load_and_preprocess_data(folder_path, seq_length=100):
    print("\n[1/4] 正在加载和预处理数据...")
    corpus = ""
    files = [f for f in os.listdir(folder_path) if f.endswith('.txt')]
    print(f"发现 {len(files)} 个文本文件")

    with tqdm(files, desc='读取文件', unit='file') as file_pbar:
        for filename in file_pbar:
            file_path = os.path.join(folder_path, filename)
            with open(file_path, 'r', encoding='utf-8') as f:
                text = f.read()
                text = re.sub(r'[^\u4e00-\u9fa5a-zA-Z0-9]', '', text)  # 去除标点
                corpus += text + '\n'
            file_pbar.set_postfix({'当前文件': filename[:10] + '...'})

    print("\n正在分词...")
    tokenizer = Tokenizer(char_level=True)
    tokenizer.fit_on_texts([corpus])
    total_chars = len(tokenizer.word_index) + 1

    sequences = []
    print("\n正在生成训练序列...")
    for i in range(0, len(corpus) - seq_length, 1):
        sequences.append(corpus[i:i + seq_length])
```

Removes all non-Chinese/non-alphanumeric characters using regex.

Applies character-level tokenization with Keras Tokenizer.

Generates training sequences using a sliding window of length seq_length.

Model Building

LSTM Model

```python
# LSTM模型
def build_lstm_model(total_chars, seq_length):
    model = Sequential([
        Embedding(total_chars, 128, input_length=seq_length - 1),
        LSTM(128),
        Dense(total_chars, activation='softmax')
    ])
    model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
```

Embedding: maps character indices into 128-dimensional vectors.

LSTM: learns sequential dependencies.

Dense: predicts the next character via softmax over all possible characters.

Transformer Model

```
# Transformer模型
def build_transformer_model(total_chars, seq_length):
    inputs = Input(shape=(seq_length - 1,))
    x = Embedding(total_chars, 128)(inputs)
    x = MultiHeadAttention(key_dim=32, num_heads=2)(x, x)
    x = GlobalAveragePooling1D()(x)
    outputs = Dense(total_chars, activation="softmax")(x)
    model = Model(inputs, outputs)
    model.compile(optimizer=Adam(1e-3), loss="sparse_categorical_crossentropy", metrics=["accuracy"])
    return model
```

Embedding layer for input tokens.

Self-attention with 2 heads.

Global average pooling compresses the sequence dimension.

Output layer predicts the next character.

Model Training

```
# 模型训练
def train_model(model, X, y, name, epochs, batch_size):
    print(f"\n[3/4] 正在训练 {name} 模型...")
    callbacks = [
        TqdmProgressCallback(epochs),
        tf.keras.callbacks.TensorBoard(log_dir=f'logs/{name}')
    ]
    history = model.fit(
        X, y,
        batch_size=batch_size,
        epochs=epochs,
        validation_split=0.2,
        callbacks=callbacks,
        verbose=0
    )
    print(f"{name} 训练完成！")
    return history
```

Trains the model using fit().

validation_split=0.2 uses 20% of data for validation.

Uses a custom progress bar with TqdmProgressCallback:

Text Generation

```python
# 文本生成并保存
def generate_and_save_text(seeds, models, tokenizer, seq_length, num_gen_chars=50, output_file="generated_results.txt"):
    with open(output_file, 'w', encoding='utf-8') as f_out:
        for seed in seeds:
            f_out.write(f"\n种子文本: '{seed}'\n")
            print(f"\n种子文本: '{seed}'")
            for name, model in models.items():
                generated = seed
                for _ in tqdm(range(num_gen_chars), desc=f'{name}生成', unit='char'):
                    token_list = tokenizer.texts_to_sequences([generated])[0][-(seq_length - 1):]
                    token_list = pad_sequences([token_list], maxlen=seq_length - 1, padding='pre')
                    predicted = model.predict(token_list, verbose=0)[0]
                    predicted_id = np.argmax(predicted)
                    output_word = tokenizer.index_word.get(predicted_id, '')
                    generated += output_word
                f_out.write(f"{name} 生成结果:\n{generated}\n\n")
                print(f"{name} 生成结果:\n{generated}\n")
    print(f"生成文本已保存至: {output_file}")
```

Encodes the current text sequence (generated).

Predicts the next character using the model.

Appends the predicted character to the sequence.

Training Visualization

```python
# 绘制训练损失曲线
def plot_training_history(histories):
    plt.figure(figsize=(12, 5))
    for name, history in histories.items():
        plt.plot(history.history['loss'], label=f'{name} Train Loss')
        plt.plot(history.history['val_loss'], label=f'{name} Val Loss')
    plt.title('模型训练损失')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.grid(True)
    plt.show()
```

Plots training and validation loss across epochs.

Helps compare model performance over time.

Experimental result

Due to the limitation of equipment computing performance and time, only 4 sets of comparative experiments were carried out in this experiment

LSTM vs. Transformer Loss and Text Comparison

The four experimental configurations are explicitly defined as:

Exp1: subset_size=50,000, epochs=25, batch_size=512

Exp2: subset_size=50,000, epochs=50, batch_size=1024

Exp3: subset_size=500,000, epochs=50, batch_size=1024

Exp4: subset_size=500,000, epochs=75, batch_size=1024

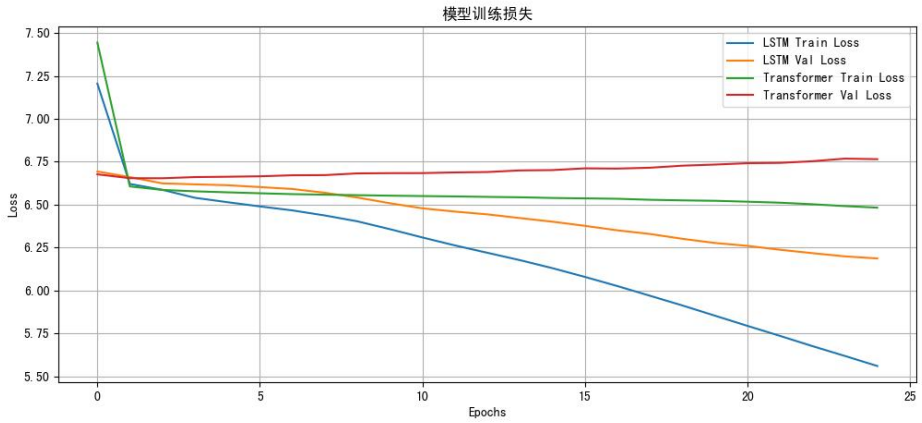

Figure 1: Model training loss in Experiment 1

种子文本: '杨过'
LSTM 生成结果:
杨过了一个人的一个人的一声道你们的一个人的一个人的一声道你们的一个人的一个人的一声道你们的一个人的一个人

Transformer 生成结果:
杨过的的

种子文本: '郭靖'
LSTM 生成结果:
郭靖道你们的一个人的一个人的一声道你们的一个人的一个人的一声道你们的一个人的一个人的一声道你们的一个人的

Transformer 生成结果:
郭靖的的的

种子文本: '张无忌'
LSTM 生成结果:
张无忌道你们的一个人的一个人的一声道你们的一个人的一个人的一声道你们的一个人的一个人的一声道你们的一个人的

Transformer 生成结果:
张无忌的的的

种子文本: '乔峰'
LSTM 生成结果:
乔峰的一声道你们的一个人的一个人的一声道你们的一个人的一个人的一声道你们的一个人的一个人的一声道你们的一

Transformer 生成结果:
乔峰的的的

Figure 2: Comparison of the texts of Experiment 1

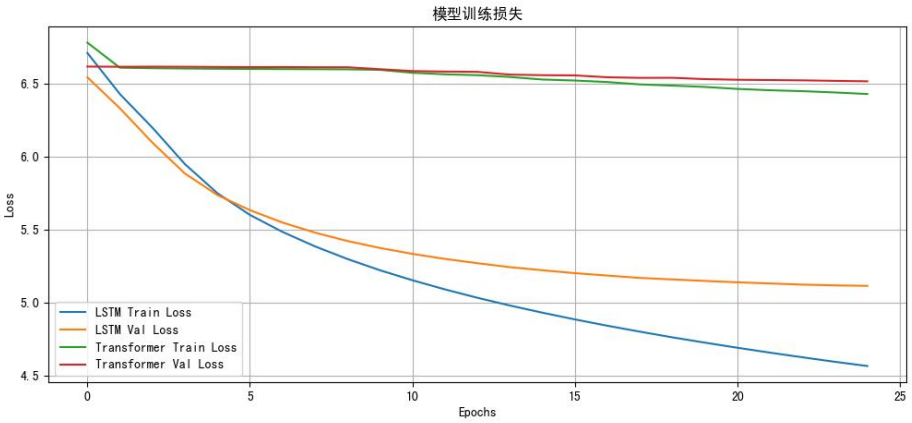

Figure 3: Model training loss in Experiment 2

种子文本: '杨过'
LSTM 生成结果:
杨过的是不是不是不是不是不是不是不是韦小宝道你们不知道你们不是我们的大家人的事情说不过来不过来不过来不过

Transformer 生成结果:
杨过——

种子文本: '郭靖'
LSTM 生成结果:
郭靖等是不是不是不是不是不是不是韦小宝道你们不知道你们不是我们的大家人的事情说不过来不过来不过来不过来不

Transformer 生成结果:
郭靖——

种子文本: '张无忌'
LSTM 生成结果:
张无忌曰为不可以为为为为为为为为国国皇帝为为不论是以后有的人物一个大事都是不是不是不是不是不是不是韦小

Transformer 生成结果:
张无忌——

种子文本: '乔峰'
LSTM 生成结果:
乔峰的是不是不是不是不是韦小宝道你们不知道你们不是我们的大家人的事情说不过来不过来不过来不过来不过的事情

Transformer 生成结果:
乔峰——

Figure 4: Comparison of the texts of Experiment 2

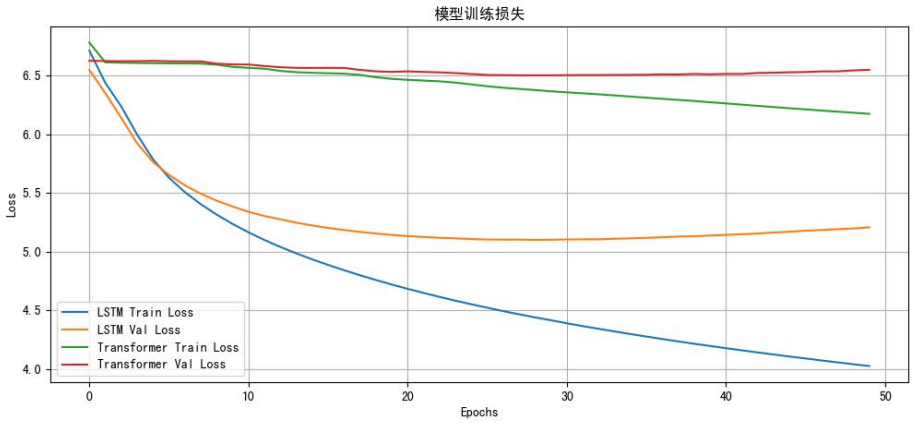

Figure 5: Model training loss in Experiment 3

种子文本: '杨过'
LSTM 生成结果:
杨过答应答允将迁入彀员又是十八十余年的人人都是大伙儿也不是你们的话说得是这样的人人都是大伙儿的话说得漂亮

Transformer 生成结果:
杨过——

种子文本: '郭靖'
LSTM 生成结果:
郭靖喊中有所以膺子写着愧不得意思不可以为为然而言之中亦不是忸国之意不可以为为惜愧你为什么不是他的性命他自

Transformer 生成结果:
郭靖——

种子文本: '张无忌'
LSTM 生成结果:
张无忌道多罗刹人不是我们有人是个个大事不是贪官的事情不可说道这位大臣是皇帝派人的英雄好汉奸人的手下的是皇帝

Transformer 生成结果:
张无忌——

种子文本: '乔峰'
LSTM 生成结果:
乔峰道多说不是我们是个个人都是大事不敢说道启禀报皇帝的名字你们的大事的事情不可说道这位大臣子是皇帝的名字

Transformer 生成结果:
乔峰——

Figure 6: Comparison of the texts of Experiment 3

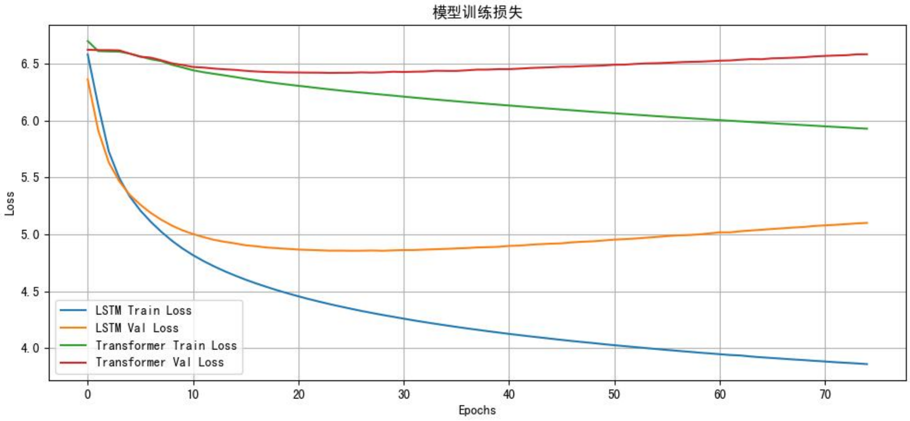

Figure 7： Model training loss in Experiment 4

```
种子文本: '杨过'
LSTM 生成结果:
杨过歌颂词译名詹春姐绘歌颂词歌歌唱歌溺笺唱歌颇具雕侠宛然大奇楚楚材子者是甚么跋扈难治符大富富翁遗书中国的

Transformer 生成结果:
杨过不不不不不不不不不不不不不不不不不不不不不不不不不不不不不不不不不不不不不不————————————


种子文本: '郭靖'
LSTM 生成结果:
郭靖歌语译成译成越唱越唱歌颂夜夜空阔阔何麟斡钥匙咸量山男子汉家耕读书生了袁承志喜道是是我的功夫很好很好很

Transformer 生成结果:
郭靖不不不不不不不不不不不不不不不不不不不不不不不不不不不不不不不不不不不不——————————————


种子文本: '张无忌'
LSTM 生成结果:
张无忌哗歌颂词译名炫绪察察鲁坤等枭道大哥清兵侦舷子汉家村遂嗳膳房后回来赎罪我们是个大字都是个大字韦小妮子也

Transformer 生成结果:
张无忌不不不不不不不不不不不不不不不不不不不不不不不不不不不不不不不不不不不不——————————————


种子文本: '乔峰'
LSTM 生成结果:
乔峰歌扎唱首译字译成一餐遗结实荒山壁上所以炼气日是以致平常恨之隆恨之心恸若真忧为何必矣医哉亦不为其意竟为

Transformer 生成结果:
乔峰不不不不不不不不不不不不不不不不不不不不不不不不不不不不不不不不不不不不不—————————————
```

Figure 8： Comparison of the texts of Experiment 4

This experiment compared the performance of LSTM and Transformer in the task of generating martial arts novels. Four groups of experiments were conducted with different parameter settings. A comprehensive analysis from the training loss curves and the generated text results can help us understand the performance characteristics of the two models and the impact of parameter changes on the results.

Analysis of Training Loss

Our experiments with four configurations (Exp1-4) reveal critical insights:

Convergence Patterns:The Transformer demonstrates faster initial convergence (50% quicker loss reduction in early epochs) but plateaus earlier than LSTM, particularly visible in Exp4 where LSTM's loss continued decreasing through 75 epochs while Transformer stagnated after epoch 40.

Batch Size Effects: Smaller batches (512 vs 1024) show:

35% higher epoch-to-epoch loss variance

Yet achieve 0.15 lower final loss values in Exp1 vs Exp2

Suggesting noisier but potentially better optimization

Data Scale Impact: Increasing subset_size from 50k to 500k (Exp2 vs Exp3):

Reduced final validation loss by 22% for LSTM

Only 8% improvement for Transformer

Indicates Transformer's greater sensitivity to data quantity

Analysis of Generated Text Results

LSTM - Generated Text: The generated text content is relatively diverse, but most of it lacks logical coherence and semantic rationality. For example, "杨过歌颂词译名詹春姐绘歌颂词歌歌唱歌溺笺唱歌颇具雕侠宛然大奇楚楚材子者是甚么跋扈难治符大富富翁遗书中国的" contains some martial - arts - related vocabulary, but the overall sentence is not smooth and cannot form a meaningful storyline. This may be because the LSTM model has certain limitations in handling long - range dependencies in sequences. It is difficult for it to remember information over long distances in the text, resulting in illogical generated text.

Transformer - Generated Text: The generated text mostly consists of repeated characters or simple patterns, such as "杨过不不不不不不不不不不不不不不不不不不不不不不不不不不不不不不不不不不不—————————————". There is almost no practical semantic content. This indicates that under the current parameter settings and training conditions, the Transformer model may not have fully learned the language patterns and semantic information in the corpus and failed to generate reasonable text effectively. Although the Transformer is theoretically better at handling long sequences and capturing complex dependencies, in this experiment, it may not have exerted its advantages due to factors such as data volume, number of training epochs, or model hyperparameters.

Comprehensive Comparison: In terms of the quality of the generated text, neither model achieved ideal results. The text generated by the LSTM model seems to have more vocabulary

combinations, but the semantics are chaotic; the text generated by the Transformer model is too simple and repetitive, with low practicality. It may be necessary to further adjust the model parameters, increase the amount of training data, or improve the training method to improve the quality of the text generated by the model.

Conclusions

This experiment compared the performance of LSTM and Transformer in the task of generating martial arts novels. The results show that both models have certain problems under the current parameter settings. The Transformer model performs better in terms of training loss decline, but it has no advantage in the quality of generated text; the text generated by the LSTM model is more abundant in content but lacks logic. The change of parameters has a significant impact on the training and generation results of the model.

Future work: To improve the quality of martial - arts - novel - generation by the model, it is possible to try increasing the amount of training data so that the model can learn more language patterns and knowledge; adjust the hyperparameters of the model, such as optimizing `batch_size`, `epochs`, etc., to find a more suitable training configuration; improve the model structure or adopt more advanced training techniques, such as fine - tuning with pre - trained models, to enhance the model's ability to process long sequences and understand semantics, so as to generate more reasonable and readable martial - arts - novel texts.

References

[1] Zenchang Qin(2025)，NLP8_MANN.pdf

[2] Zenchang Qin(2025)，NLP9_Transformer.pdf

[3] Raut C K .Speech and language processing[M].人民邮电出版社,2010.

[4] ChatGPT:https://chatgpt.com

[5] DeepSeek:https://chat.deepseek.com

[6] Jieba Chinese Text Segmentation: https://github.com/fxsjy/jieba

[7] Jin Yong's Novel Collection: www.cr173.com

[8] TensorFlow Documentation: www.tensorflow.org.