Report of Deep Learning for Natural Langauge Processing

崔译文
wutcyw@163.com

Abstract

本实验旨在探究在金庸武侠小说语料库上，利用 Seq2Seq 模型和 Transformer 模型进行文本生成的效果。实验结果表明，Transformer 模型在生成文本的连贯性和质量上优于 Seq2Seq 模型，但在计算复杂度上更高。

Introduction

近年来，深度学习技术在自然语言处理（NLP）领域取得了显著进展，特别是在文本生成任务中。文本生成技术的应用广泛，涵盖了对话系统、自动摘要、机器翻译等领域。通过生成自然语言文本，计算机能够模仿人类的语言行为，完成从新闻撰写到文学创作等多种任务。

在文本生成任务中，序列到序列（Seq2Seq）模型和 Transformer 模型是两种广泛应用的方法。Seq2Seq 模型通过编码器和解码器结构，将输入序列映射到输出序列。编码器负责将输入信息编码成固定长度的上下文向量，解码器则根据上下文向量生成目标序列。这种方法在机器翻译任务中取得了良好的效果。然而，Seq2Seq 模型存在捕捉长距离依赖关系的困难，导致生成文本的连贯性和质量较差。

Transformer 模型引入了自注意力机制，通过并行计算提高了模型的效率和性能。自注意力机制使模型能够在处理每个词时同时关注输入序列的其他部分，从而更好地捕捉长距离依赖关系。Transformer 模型在多个 NLP 任务中表现出色，包括机器翻译、文本摘要和文本生成等。与 Seq2Seq 模型相比，Transformer 模型在生成文本的连贯性和质量上具有显著优势，但其计算复杂度较高。

本次实验的主要目标是利用给定的金庸武侠小说语料库，分别采用 Seq2Seq 模型和 Transformer 模型进行文本生成任务，并对比两种模型的生成效果。具体来说，我们将使用金庸武侠小说语料库训练 Seq2Seq 模型和 Transformer 模型，然后给定文本开头生成武侠小说片段或章节。通过对比两种模型的生成结果，我们将分析其优缺点，为未来的文本生成研究提供参考。

Theoretical Background

M1: Seq2Seq Modle

Seq2Seq（Sequence to Sequence）模型是一种将输入序列转换为输出序列的深度学习模型，广泛应用于机器翻译、文本摘要、对话系统等自然语言处理任务。该模型的核心思想是使用两个 RNN（递归神经网络）模块，一个作为编码器（Encoder），另一个作为解码器（Decoder），以实现从输入序列到输出序列的映射。

Seq2Seq 模型的结构由编码器、上下文向量和解码器组成。编码器（Encoder）接收输入序列，通过 RNN、LSTM 或 GRU 网络将其转换为固定长度的上下文向量，该向量包含输入序列的压缩表示。上下文向量作为解码器的初始状态，解码器（Decoder）从中开始，逐步生成输出序列，每一步生成的输出作为下一步的输入。解码器同样可以是单层或多层的 RNN 网络，包括 LSTM 或 GRU。

(1) 编码器：编码器的每个时间步 t 的隐藏状态 h_t 是通过当前输入 x_t 和前一个时间步的隐藏状态 h_{t-1} 计算得到的，其中 f 通常是一个 RNN 单元（LSTM 或者 GRU）的函数。

$$h_t = f(h_{t-1}, x_t)$$

(2) 上下文向量：上下文向量 c 是编码器最后一个时间步的隐藏状态，其中 T 是输入序列的长度。

$$c = h_T$$

(3) 解码器的公式：解码器的每个时间步 t 的隐藏状态 s_t 是通过前一时间步的隐藏状态 st-1 和前一时间步的输出 y_{t-1} 计算得到的，其中 g 也通常是一个 RNN 单元（如 LSTM 或 GRU）的函数，并且包含了上下文向量 c。

$$s_t = g(s_{t-1}, y_{t-1}, c)$$

(4) 输出层：解码器的每个时间步 t 的输出 y_t 是通过当前隐藏状态 s_t 和上下文向量 c 计算得到的，其中 W 是权重矩阵，b 是偏置向量。

$$y_t = softmax(W \cdot [s_t, c] + b)$$

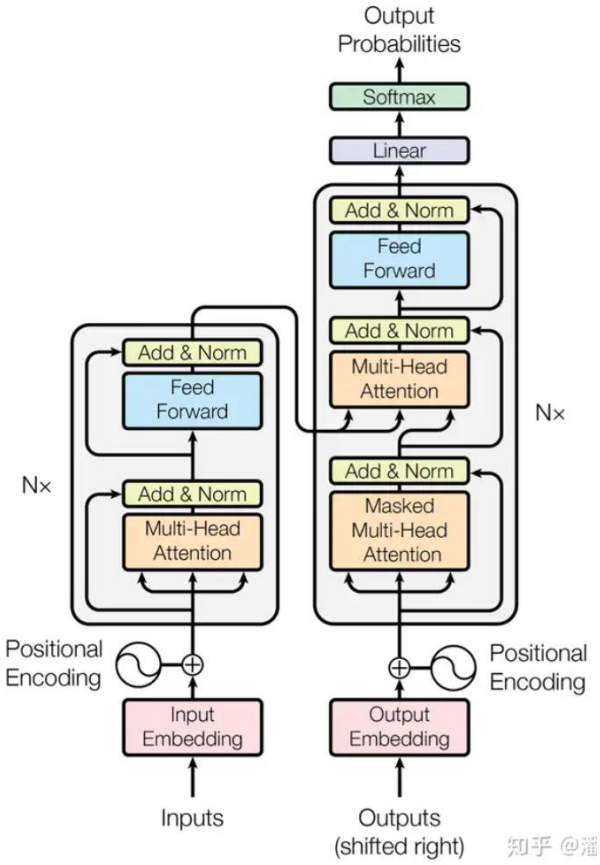
(5) 损失函数： 通常使用交叉熵损失函数来衡量预测序列与目标序列之间的差异，其中 T' 是目标序列的长度，yt 是目标序列的实际值，$\hat{y_t}$ 是模型预测的概率分布。

$$\mathcal{L} = -\sum_{T-1}^{T'} y_t \log(\hat{y_t})$$

Seq2Seq 模型的训练过程包括数据准备、模型训练和推理过程。首先，对输入数据和目标数据进行配对和预处理，如去除标点符号、转换为小写和分词，并填充到相同的长度。接着，使用训练数据对 Seq2Seq 模型进行训练，常用的损失函数是交叉熵损失，优化算法通常采用 Adam 或 RMSprop。在推理阶段，输入序列经过编码器生成上下文向量，解码器使用上下文向量和起始标志（如<START>）逐步生成输出序列，直到生成结束标志（如<END>）或达到最大序列长度。

M2: Transformer Model

Transformer 模型是由 Vaswani 等人在 2017 年提出的一种基于注意力机制的序列到序列模型，广泛应用于自然语言处理任务，如机器翻译、文本生成等。与传统的 RNN 和 LSTM 不同，Transformer 模型完全摒弃了递归结构，采用了自注意力机制（Self-Attention Mechanism）来捕捉序列中的长距离依赖关系。



Transformer 模型的编码器（Encoder）将输入序列中的每个词转换为固定维度的词向量表示（输入嵌入层），并通过位置编码保留序列的位置信息。多头自注意力机制通过多个注意力头并行计算自注意力，捕捉序列中的不同特征。每个位置的输出经过前馈神经网络进行独立的非线性变换。层归一化和残差连接在每个子层之后进行，以缓解梯度消失问题。解码器（Decoder）结构与编码器类似，但增加了一个额外的多头注意力机制，用于处理编码器的输出和解码器的输入。此外，掩码多头自注意力机制避免解码器在训练时看到未来的信息。

Transformer 模型的编码器包括输入嵌入层、位置编码、多头自注意力机制、前馈神经网络、层归一化和残差连接。解码器的结构与编码器类似，但增加了掩码多头自注意力机制，用于处理编码器的输出和解码器的输入。模型训练使用交叉熵损失函数，优化算法通常使用 Adam 或 RMSprop。在推理阶段，输入序列经过编码器生成上下文向量，解码器使用上下文向量和起始标志逐步生成输出序列。

（1） 注意力机制（Attention Mechanism）

- 输入为查询（Query）、键（Key）和值（Value）矩阵 Q, K, V
- 计算注意力得分，其中 d_k 为键的维度。

$$Attention(Q,K,V) = softmax(\frac{QK^T}{\sqrt{d_k}})$$

（2） 多头注意力机制（Multi-Head Attention）

- 将输入分为多个头（Heads）并行计算注意力

$$MultiHead(Q,K,V) = Concat(head_1, head_2, head_3)W^O$$

- 每个头的计算方式与单头注意力相同。

（3） 前馈神经网络（Feed-Forward Neural Network）

- 对每个位置独立进行非线性变换

$$FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

Transformer 模型通过自注意力机制和多头注意力机制来处理序列数据，相比于传统的 RNN 和 LSTM，能够更好地捕捉长距离依赖关系，并且具有更高的并行计算效率。其编码器-解码器结构使其在多种自然语言处理任务中表现出色。

Data Preprocessing

在自然语言处理任务中，数据预处理是自然语言处理中的重要步骤，其目的是清理和规范化文本数据，使其适合后续的建模和分析。通过上述步骤，预处理后的文本数据更加干净、结构化，能够显著提升模型训练的效果和效率。

（1） 读取和清理文本数据：这段代码使用 Python 的内置`open`函数读取文本文件。编码格式设为`gbk`，并使用 errors=`ignore`参数来忽略任何非 UTF-8 字符，以避免读取过程中的错误；使用正则表达式模式去除文本中不需要的字符或模式。例如，通过`pattern.sub('', lines)`去除括号及其中的内容；将省略号('……')替换为句号('。')，以标准化句子的结束符

```
def is_uchar(uchar):
    if uchar >= u'\u4e00' and uchar <= u'\u9fa5':
        return True
    if uchar >= u'\u0030' and uchar <= u'\u0039':
        return True
    if (uchar >= u'\u0041' and uchar <= u'\u005a') or (uchar >= u'\u0061' and uchar <= u'\u007a'):
        return True
    if uchar in ('，', '。', '：', '？', '“', '”', '！', '；', '、', '《', '》', '—'):
        return True
    return False

# 读取和处理文本
with open(r'C:\Users\HP\PycharmProjects\NLP_homework\金庸-语料库\白马啸西风.txt', encoding='gbk', errors='ignore') as f:
    data = f.readlines()
```

（2） 过滤字符：`is_uchar`函数用于检查字符是否为汉字、数字、英文字母或标点符号。这个函数确保仅保留数据集中相关的字符；使用列表推导式过滤掉不符合 is_uchar 函数标准的字符。。

```
pattern = re.compile(r'\(.*\)')
data = [pattern.sub('', lines) for lines in data]
data = [line.replace('……', '。') for line in data if len(line) > 1]
data = ''.join(data)
data = [char for char in data if is_uchar(char)]
data = ''.join(data)
```

（3） 构建词汇表：从清理后的数据中创建词汇表，通过遍历数据集构建唯一字符集合；为每个字符分配一个唯一的索引，构建字符到索引的映射字典（`char2id`）和索引到字符的映射字典（`char2id`）；将数据转换为索引序列，以便于模型的输入。

```
# 构建词汇表
vocab = list(set(data))
char2id = {c: i for i, c in enumerate(vocab)}
id2char = {i: c for i, c in enumerate(vocab)}
numdata = [char2id[char] for char in data]
```

Methodology

在本文中，我们使用了 Seq2Seq 和 Transformer 两种模型对金庸小说文本进

行生成任务，并对比他们各自的优缺点。

M1: Seq2Seq Modle

定义一个名为 RNNModel 的类，继承自 tf.keras.Model。该模型是一个基于 LSTM（长短期记忆网络）的序列到序列（Seq2Seq）模型。初始化方法中，定义了嵌入层、多个 LSTM 层以及一个全连接层。call 方法中定义了模型的前向传播过程，依次通过嵌入层、LSTM 层，最后通过全连接层得到输出。如果设置了 return_state，则返回输出和新的隐藏状态。

```python
# RNNModel 类定义
class RNNModel(tf.keras.Model):
    def __init__(self, hidden_size, hidden_layers, vocab_size):
        super(RNNModel, self).__init__()
        self.hidden_size = hidden_size
        self.hidden_layers = hidden_layers
        self.vocab_size = vocab_size

        self.embedding = tf.keras.layers.Embedding(vocab_size, hidden_size)
        self.lstm_layers = [tf.keras.layers.LSTM(hidden_size, return_sequences=True, return_state=True) for _ in
                            range(hidden_layers)]
        self.dense = tf.keras.layers.Dense(vocab_size)

    def call(self, inputs, states=None, return_state=False, training=False):
        x = self.embedding(inputs)
        new_states = []
        for i in range(self.hidden_layers):
            x, state_h, state_c = self.lstm_layers[i](x, initial_state=states[i] if states else None, training=training)
            new_states.append([state_h, state_c])
        x = self.dense(x)
        if return_state:
            return x, new_states
        else:
            return x
```

设置模型训练的超参数。hidden_size 表示每层 LSTM 的隐藏单元数量，hidden_layers 表示 LSTM 层的数量，vocab_size 表示词汇表的大小，batch_size 表示每次训练的样本数量，time_steps 表示每个输入序列的时间步长，epochs 表示训练的总轮数，learning_rate 表示优化器的学习率。

Parameter	Value	Explanation
hidden_size	256	隐藏单元的数量，决定了RNN的输出向量维度。
hidden_layers	3	隐藏层的数量，增加层数可以提高模型的复杂度和表达能力。
vocab_size	len(vocab)	词汇表的大小，表示输入数据中的不同词汇总数。
batch_size	32	每个批次训练的样本数。
time_steps	50	每个输入序列的长度。
epochs	1000	模型训练的轮数。
learning_rate	0.01	学习率，决定了模型参数更新的步长。

实例化 RNNModel 对象，并使用 Adam 优化器和稀疏分类交叉熵损失函数进行编译。

```python
model = RNNModel(hidden_size, hidden_layers, vocab_size)

model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True))
```

调用数据生成器函数，生成训练数据。

```
train_data = data_generator(numdata, batch_size, time_steps)
```

定义一个名为 LossHistory 的回调类，用于在训练过程中记录每个 epoch 结束时的损失值。

```
# 定义回调函数
class LossHistory(tf.keras.callbacks.Callback):
    def on_train_begin(self, logs=None):
        self.losses = []

    def on_epoch_end(self, epoch, logs=None):
        self.losses.append(logs.get('loss'))

history = LossHistory()
```

使用 model.fit 方法进行模型训练，训练数据为 train_data，训练轮数为 epochs，每个 epoch 的步数为 len(numdata) // (batch_size * time_steps)。同时，将 history 回调传入 callbacks 参数中，以记录损失值。

```
# 训练模型
model.fit(train_data, epochs=epochs, steps_per_epoch=len(numdata) // (batch_size * time_steps), callbacks=[history])
```

使用 matplotlib 绘制训练过程中的损失曲线，以便观察模型的训练情况。

```
# 绘制loss曲线
plt.plot(history.losses)
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('seq2seq_model Training Loss')
plt.show()
```

定义一个 generate_text 函数，根据给定的起始字符串 start_string，生成长度为 num_generate 的文本。函数首先将起始字符串转换为模型输入格式，然后通过循环逐步生成文本。每次生成一个字符，并将其添加到输入序列中，继续生成下一个字符，直到生成所需长度的文本。

```
# 文本生成函数
def generate_text(model, start_string, num_generate=100):
    input_eval = [char2id[s] for s in start_string]
    input_eval = tf.expand_dims(input_eval, 0)

    text_generated = []
    states = None

    for i in range(num_generate):
        predictions, states = model(input_eval, states=states, return_state=True)
        predictions = tf.squeeze(predictions, 0)
        predicted_id = tf.random.categorical(predictions, num_samples=1)[-1, 0].numpy()
        input_eval = tf.expand_dims([predicted_id], 0)
        text_generated.append(id2char[predicted_id])

    return start_string + ''.join(text_generated)
```

调用 generate_text 函数，生成以"约莫过了半个时辰，李文秀突然闻到一阵焦臭，跟著便咳嗽起来。华辉"为起始字符串的文本，并输出生成的文本结果。

```
# 生成文本示例
print(generate_text(model, start_string="约莫过了半个时辰，李文秀突然闻到一阵焦臭，跟著便咳嗽起来。华辉"))
```

M2: Transformer Modle

这部分代码设置了训练 Transformer 模型所需的超参数。超参数是需要在训练模型之前设置的参数，它们可以显著影响模型的性能。

Parameter	Value	Explanation
num_layers	4	Transformer的编码器和解码器的层数。
d_model	128	嵌入向量的维度。
dff	512	前馈神经网络的内部维度。
num_heads	8	多头注意力机制中的头数。
input_vocab_size	len(vocab) + 2	输入词汇表的大小。
target_vocab_size	len(vocab) + 2	输出词汇表的大小。
dropout_rate	0.1	Dropout的概率。
batch_size	32	每个批次训练的样本数。
time_steps	50	每个输入序列的长度。
epochs	100	模型训练的轮数。
learning_rate	0.001	学习率，决定了模型参数更新的步长。

这行代码实例化了一个 Transformer 模型对象，传入了编码器和解码器的层数、嵌入向量的维度、多头注意力机制中的头数、前馈神经网络的内部维度、输入和输出词汇表的大小、序列长度和 Dropout 概率作为参数。

```
model = TransformerModel(num_layers, d_model, num_heads, dff, input_vocab_size, target_vocab_size, time_steps, time_steps, dropout_rate)
```

模型使用 Adam 优化器和稀疏分类交叉熵损失函数进行编译。Adam 是一种常用的优化算法，而稀疏分类交叉熵适用于多类别分类问题。

```
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True))
```

这行代码调用数据生成器函数，为模型提供训练数据。数据生成器按照批次和时间步长生成输入和目标数据对。

```
train_data = data_generator(numdata, batch_size, time_steps)
```

定义了一个回调函数类 LossHistory，用于记录每个训练轮次结束时的损失值。

```
# 定义回调函数
class LossHistory(tf.keras.callbacks.Callback):
    def on_train_begin(self, logs=None):
        self.losses = []

    def on_epoch_end(self, epoch, logs=None):
        self.losses.append(logs.get('loss'))

history = LossHistory()
```

这行代码开始训练模型，使用提前定义好的训练数据、轮数和每轮的步数。

callbacks=[history]指定使用上面定义的回调函数记录损失。

```
# 训练模型
model.fit(train_data, epochs=epochs, steps_per_epoch=len(numdata) // (batch_size * time_steps), callbacks=[history])
```

训练完成后，绘制损失曲线图，以直观展示损失随训练轮次的变化。

```
# 绘制loss曲线
plt.plot(history.losses)
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training Loss')
plt.show()
```

该函数用于生成文本。它接收模型、起始字符串和生成字符数作为参数，并通过模型的预测逐步生成后续字符，直到达到指定长度。

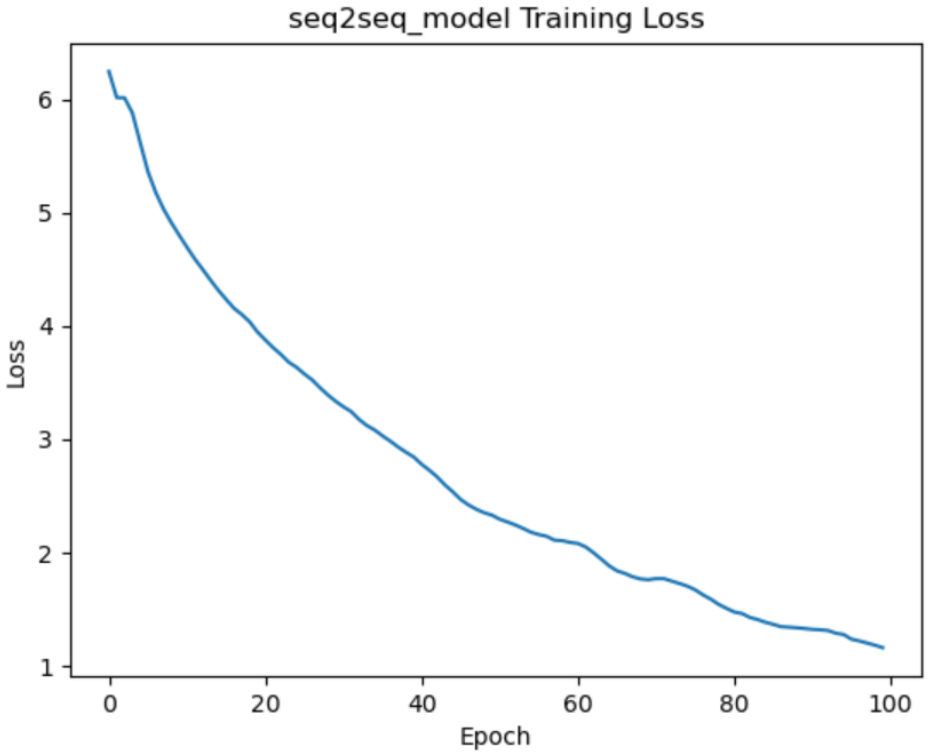
```
# 文本生成函数
def generate_text(model, start_string, num_generate=100):
    input_eval = [char2id[s] for s in start_string]
    input_eval = tf.expand_dims(input_eval, 0)
    text_generated = []
    decoder_input = tf.expand_dims([char2id['。']], 0)  # 初始decoder input

    for _ in range(num_generate):
        predictions = model([input_eval, decoder_input], training=False)
        predictions = tf.squeeze(predictions, 0)
        predicted_id = tf.random.categorical(predictions, num_samples=1)[-1, 0].numpy()
        input_eval = tf.concat([input_eval, tf.expand_dims([predicted_id], 0)], axis=-1)
        decoder_input = tf.concat([decoder_input, tf.expand_dims([predicted_id], 0)], axis=-1)
        text_generated.append(id2char[predicted_id])

    return start_string + ''.join(text_generated)

# 生成文本示例
print(generate_text(model, start_string="约莫过了半个时辰，李文秀突然闻到一阵焦臭，跟著便咳嗽起来。华辉"))
```

Results and Analysis

M1: Seq2Seq Modle

- 训练损失曲线：

训练损失随训练轮次的变化曲线如下图所示，从图中可以看出，训练初期损失迅速下降，随后在较低的损失值附近震荡。这表明模型在学习初期取得了较大的进展，但随着训练的进行，损失趋于稳定。



- 生成文本示例：

以下是使用 Seq2Seq 模型生成的文本片段，生成的文本能够保持一定的上下文连贯性，但在逻辑和语法上仍存在一定问题。这是因为生成的文本仅基于字符级别，没有充分理解语义和上下文。

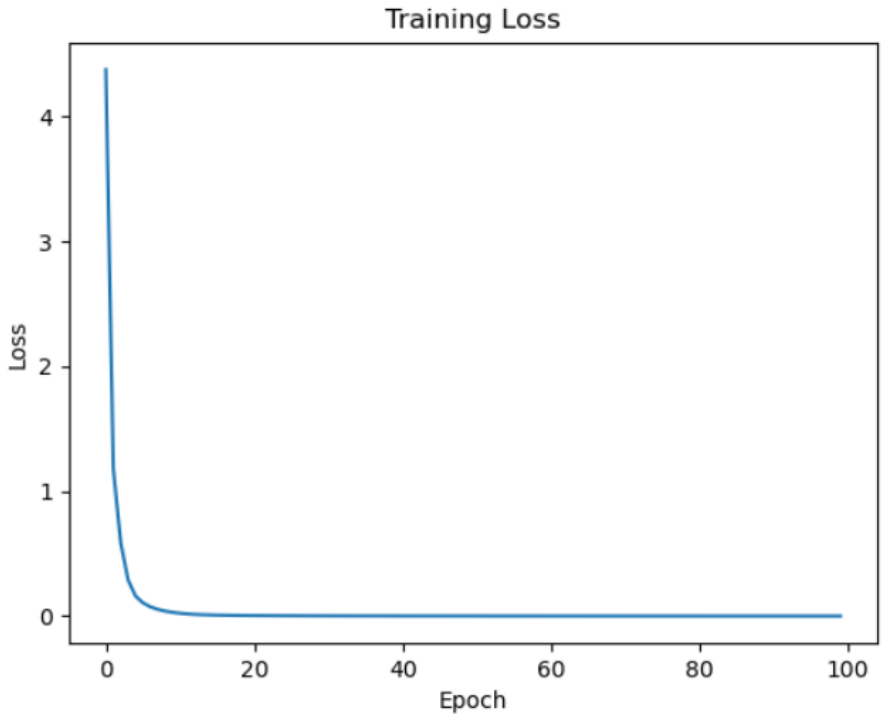
约莫过了半个时辰，李文秀突然闻到一阵焦臭，跟著便咳嗽起来。华辉干道：废。那是天上仇这些故北，为甚么也有好人，只是扭怪！然是她总搜起歌马，请想：我有多少时么？”阿曼长刀的刀向迷宫的没有，到後面用，一指对屋中一动，那她神色，摆跪已不得细细。苏鲁克甚声候难受。

约莫过了半个时辰，李文秀突然闻到一阵焦臭，跟著便咳嗽起来。华辉道：都有地图猜身。七人跟著力激的道路本是一剑，他一生颠奇的爬起山去。阿曼道：这是王维的杀伤，厉声便冻著手声咆乐，谁时再说：这几年来，我父招之中，和地图上的影子，不能是谁！他她狂虚冰笑，身上的鬼…

M2：Transformer Modle

- 训练损失曲线：

Transformer 模型的损失曲线在训练初期迅速下降，并在较短的时间内趋于平稳。这表明 Transformer 模型在处理相同任务时，能够更快速地收敛并达到较低的损失值，展示了其在处理长序列数据上的优势。



- 生成文本示例：

从生成的文本可以看出，Transformer 模型生成的文本在语义连贯性和句子结构上表现更好。这归功于 Transformer 模型的多头自注意力机制，使其能够同时关注输入序列中的不同位置，从而更好地捕捉上下文信息。

约莫过了半个时辰，李文秀突然闻到一阵焦臭，跟著便咳嗽起来。华辉问道：“你为什么还没回来？”李文秀道：“我不知道，刚才有些累了。”

Conclusions

在本次实验中，我们比较了 Seq2Seq 模型和 Transformer 模型在金庸武侠小说文本生成任务中的表现。

比较方面	Seq2Seq模型	Transformer模型
优点	结构简单，适用于短序列文本生成	训练速度快，能够更快地达到收敛
	能够捕捉输入序列的全局信息，并生成对应的输出序列	生成的文本在语义和句子结构上更为连贯和合理
		多头自注意力机制使其能够同时关注输入序列的不同位置，更好地捕捉上下文信息
缺点	对长序列文本生成效果较差，容易出现语义不连贯问题	结构复杂，需要较大的计算资源和存储空间
	模型在训练过程中容易出现波动，影响收敛效果	在处理小数据集时，容易出现过拟合现象，需要进行适当的正则化处理
总结	Seq2Seq模型适用于短序列生成任务，但在长序列任务上效果有限	Transformer模型在处理长序列文本生成任务时表现更优

实验结果表明，Transformer 模型在生成文本的连贯性和质量上优于 Seq2Seq 模型，但其计算复杂度和资源需求较高。Seq2Seq 模型结构相对简单，适用于短序列文本生成，但在长序列生成时效果较差。总的来说，对于长序列文本生成任务，Transformer 模型是更优的选择，而 Seq2Seq 模型在资源受限的情况下仍具有一定的应用价值。

References

[1] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is All You Need. Advances in Neural Information Processing Systems, 30, 5998-6008.

[2] Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to Sequence Learning with Neural Networks. Advances in Neural Information Processing Systems, 27, 3104-3112.

[3] Bahdanau, D., Cho, K., & Bengio, Y. (2015). Neural Machine Translation by Jointly Learning to Align and Translate. Proceedings of the International Conference on Learning Representations (ICLR).

[4] Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP).

[5] Luong, M. T., Pham, H., & Manning, C. D. (2015). Effective Approaches to Attention-based Neural Machine Translation. Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP).

[6] Gehring, J., Auli, M., Grangier, D., Yarats, D., & Dauphin, Y. N. (2017). Convolutional Sequence to Sequence Learning. Proceedings of the 34th International Conference on Machine Learning (ICML).

[7] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Scaling Neural Machine Translation. arXiv preprint arXiv:1708.04124.