



Natural Language Processing

Language Translation

学 院 智能工程学院

专 业 智能科学与技术

组 员 1 黄楚丹 21312024

组 员 2 沈鹏飞 21312188

组 员 3 刘书睿 21312295

指导老师 沈颖

2024 年 4 月 30 日

目 录

第 1 章 介绍	1
1.1 实验背景	1
1.2 实验目的	1
第 2 章 实验原理	1
2.1 文本预处理	1
2.1.1 本质	1
2.1.2 基本方法	2
2.2 Keras	3
2.2.1 模型构建	3
2.2.2 编译模型	3
2.2.3 训练与评估	4
2.2.4 预测与应用	4
2.3 LSTM 网络	4
2.3.1 基础结构	4
2.3.2 网络架构	5
第 3 章 实验环境	5
第 4 章 数据集	6
4.1 来源	6
4.1.1 Tatoeba	6
4.1.2 ManyThings	6
4.2 结构和内容	6
第 5 章 实验过程	7
5.1 Seq2Seq 模型	7
5.1.1 实验总体设计	7
5.1.2 数据及环境准备	8
5.1.3 安装相关包	8
5.1.4 导入实验环境	8
5.1.5 配置参数	8
5.1.6 读取数据并处理	9
5.1.7 句子向量化	10
5.1.8 创建模型	11

5.1.9 编译模型	11
5.1.10 模型训练	13
5.1.11 损失曲线绘制	13
5.1.12 构建测试函数	14
5.1.13 测试	15
第 6 章 模型优化与评估	16
6.1 seq2seq	16
6.2 Attention 机制	18
第 7 章 思考题	19
第 8 章 开放题	20
第 9 章 心得体会	23
9.1 理解代码结构	23
9.2 参数调整与优化	23
9.3 实验结果的评估与分析	23
9.4 论文阅读	24
第 10 章 Labor Contribution	24

图目录

图 1-1 机器翻译发展呢阶段	1
图 2-1 文本预处理结构	2
图 2-2 分词方法	2
图 2-3 前向传播公式	5
图 4-1 cmt_zhsim.txt	7
图 4-2 cmt.txt	7
图 5-1 实验总体设计示意图	7
图 5-2 create_model() 函数	12
图 5-3 训练模型的结构	12
图 5-4 编码器推理模型的结构	13
图 5-5 解码器推理模型的结构	13
图 5-6 损失函数曲线 (batch=32, epoch=500)	14
图 5-7 Seq2Seq 模型结果	16
图 6-1 Loss.	16
图 6-2 Accuracy.	16

图 6-3	Performance Metrics	16
图 6-4	简单翻译实例	17
图 6-5	复杂翻译实例	17
图 6-6	复杂翻译实例	17
图 6-7	复杂翻译实例	18
图 6-8	信息丢失	18
图 6-9	Loss.	19
图 6-10	Accuracy.	19
图 6-11	Attention 机制翻译实例	19
图 6-12	Relation map.	19
图 8-1	This model reads an input sentence “ABC” and produces “WXYZ” as the output sentence.	21
图 8-2	The graphical illustration of the proposed model trying to generate the t-th target word yt given a source sentence.	21

表目录

表 2-1	网络参数	5
表 3-1	实验环境	6
表 5-1	软件包版本	8
表 6-1	超参设置	16
表 6-2	超参数设置	18
表 10-1	Labor Contribution.	24

第 1 章 介绍

1.1 实验背景

Machine Translation 是使计算机能够将一种语言转化为另一种语言的技术领域。机器翻译主要分为统计机器翻译时期、神经网络机器翻译时期以及前沿的研究，如图1-1所示。随着深度学习技术的发展，特别是 RNN 的广泛使用，我们已经能够处理

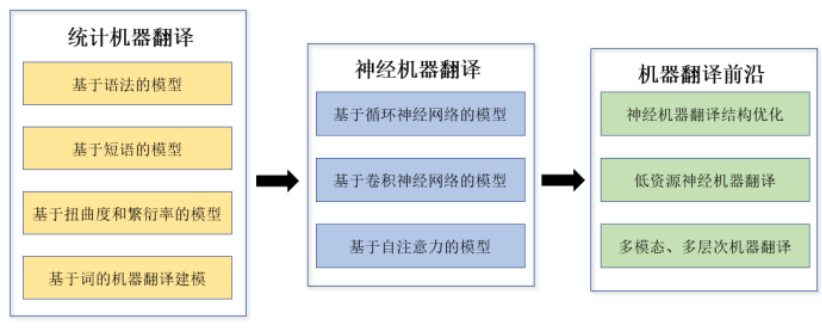


图 1-1 机器翻译发展呢阶段

和生成长度相同的序列数据。然而，在机器翻译的具体应用中，输入和输出序列的长度往往是不相等的，这就需要更为复杂的模型来处理这种类型的数据转换。Seq2Seq 模型，首次在 2014 年提出，正是为了解决这一问题。该模型设计了一个能够接收可变长输入序列并输出另一个可变长序列的框架，极大地推动了机器翻译技术的发展。

1.2 实验目的

通过中英翻译实验，深入了解神经网络在 NLP 机器翻译领域的应用。通过案例代码的学习，重点理解 seq2seq 解码器-编码器框架的思想。通过包含注意力机制的 Seq2Seq 模型的实验，深入学习如何通过加入注意力机制来提升机器翻译的性能。

第 2 章 实验原理

2.1 文本预处理

2.1.1 本质

文本预处理是将原始文本数据转换为符合模型输入要求的格式的过程。在自然语言处理中，文本预处理是一个基本且关键的步骤，因为它直接影响到模型的质量和性能。文本预处理涉及多个环节，主要包括数据清洗、文本标准化、分词、文本向量化等，旨在将原始、无结构化的文本数据转换为结构化的、数值化的形式，以便机器学习模型能够理解和处理，如图2-1所示。

文本预处理能将原始、无结构化的文本数据清洗、转换并标准化为适应机器学习模型输入的格式，从而提升模型性能并降低处理难度。

2.1.2 基本方法



图 2-1 文本预处理结构

文本处理的基本方法包括分词、词性标注和命名实体识别。下面分别介绍。

1. 分词:

分词是将连续的字序列按照特定的规则或算法重新组合成词序列的过程，有基于词典和字的两种分词方法，如图2-2所示。

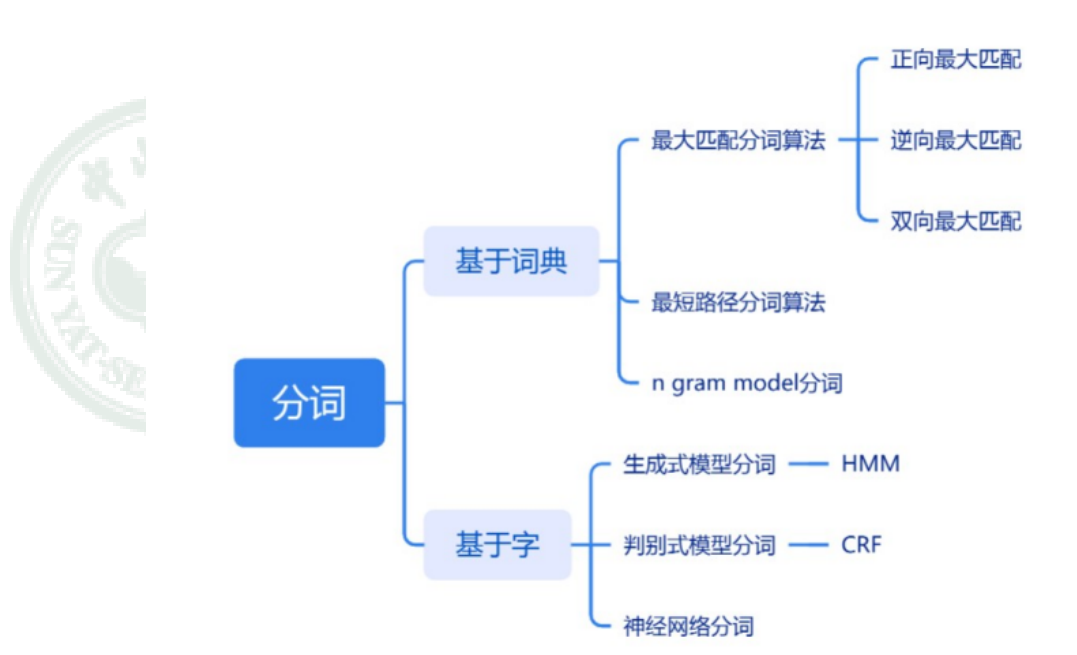


图 2-2 分词方法

2. 词性标注

词性标注旨在为文本中的每个词分配其对应的词性标签。主要分为两类：

- 基于统计模型的词性标注方法，其基本思想是将词性标注看作序列标注问题，利用统计模型确定给定词序列中每个词的最可能词性。常用模型有隐马尔可夫模型（HMM）、条件随机场（CRF）等，训练依赖于有标记数据的大型语料库，其中每个词都已正确标注词性。
- 基于深度学习的词性标注方法，一般的处理方法是词性标注视为序列标注任务，常用模型有 LSTM+CRF、BiLSTM+CRF 等。深度学习模型能够自动学习文本中的复杂特征和模式，无需手动设计规则或特征。

3. 命名实体识别

命名实体识别旨在从文本中识别出具有特定意义的实体，如人名、地名、组织名等，主要包含以下四种学习方法。

- 有监督的学习方法，依赖大规模的已标注语料库进行模型训练，常用模型有隐马尔可夫模型、语言模型、最大熵模型、支持向量机、决策树和条件随机场等。基于条件随机场的方法是命名实体识别中最成功的方法之一。
- 半监督的学习方法，其特点是利用少量标注的数据集（种子数据）进行自主学习。在标注资源有限的情况下，这种方法能够有效地利用未标注数据进行模型训练。
- 无监督的学习方法不依赖标注数据，而是利用词汇资源（如 WordNet）等进行上下文聚类。由于缺乏明确的标注信息，无监督方法通常需要更复杂的算法和更多的计算资源来识别实体。
- 基于深度学习的方法常用模型有 LSTM+CRF、BiLSTM+CRF 等，将命名实体识别视为序列标注任务，利用深度学习模型自动学习文本中的复杂特征和模式。深度学习模型具有强大的表征学习能力，能够捕捉文本中的长期依赖关系和复杂模式，从而提高命名实体识别的性能。

2.2 Keras

在 NLP 领域中，使用 Keras 进行模型构建和实验是非常普遍的，因为 Keras 提供了一个简洁而强大的接口来定义和训练各种类型的深度学习模型。下面是一些使用 Keras 在自然语言处理领域中需要掌握的基本操作。

2.2.1 模型构建

1. 模型定义: 使用 Sequential 或 Functional API 来构建模型。
2. 层的选择:
 - Embedding Layer: 将单词索引转换为密集向量。
 - MaxPooling1D: 用于下采样信息。
 - Dense: 用于输出预测。
 - Dropout: 用于防止过拟合。
3. 注意力机制: 增强模型对特定部分的聚焦能力。

2.2.2 编译模型

1. 优化器: 选择适当的优化器，如 Adam, SGD 等。
2. 损失函数: 根据问题的性质选择合适的损失函数，如 categorical_crossentropy, binary_crossentropy 等。
3. 评价指标: 如 accuracy, AUC 等。

2.2.3 训练与评估

1. 模型训练：使用 `fit` 方法训练模型。
2. 模型评估：使用 `evaluate` 方法在测试集上评估模型的性能。
3. 模型保存与加载：保存训练好的模型。

2.2.4 预测与应用

1. 使用模型进行预测：使用 `predict` 方法进行预测。
2. 细化输出：根据需求处理模型输出，如将输出转换为文本标签。

2.3 LSTM 网络

LSTM 是一种特殊类型的循环神经网络 (RNN)，专为解决标准循环网络在处理数据时遇到的长期依赖问题而设计。由 Sepp Hochreiter 和 Jürgen Schmidhuber 在 1997 年首次提出。

2.3.1 基础结构

1. 门控机制 s

在数字电路中，gate 为一个二值变量 (0,1)，0 代表关闭状态、不许任何信息通过，1 代表开放状态，允许所有信息通过。gate 在 LSTM 网络中实际上就是一层全连接层，输入是一个向量，输出是一个 0 到 1 之间的实数向量。表示以一定的比例允许信息通过。

2. 激活函数

- sigmoid 函数：Sigmoid 函数的输出映射在 (0,1) 之间，单调连续，输出范围有限。
- tanh 函数：tanh 函数的输出映射在 (-1,1) 之间，在一定范围的內，比 Sigmoid 函数收敛速度更快。

3. 网络参数

网络参数如表格5-1所示。

符号	含义
x_t	当前时刻的输入
C_t	信息候选状态
h_{t-1}	上一时刻外部状态
f_t	遗忘门控制信号
i_t	输入门控制信号
C_{t-1}	输出门控制信号
C_t	上一时刻记忆单元
o_t	当前时刻记忆单元
h_t	当前时刻外部状态

表 2-1 网络参数

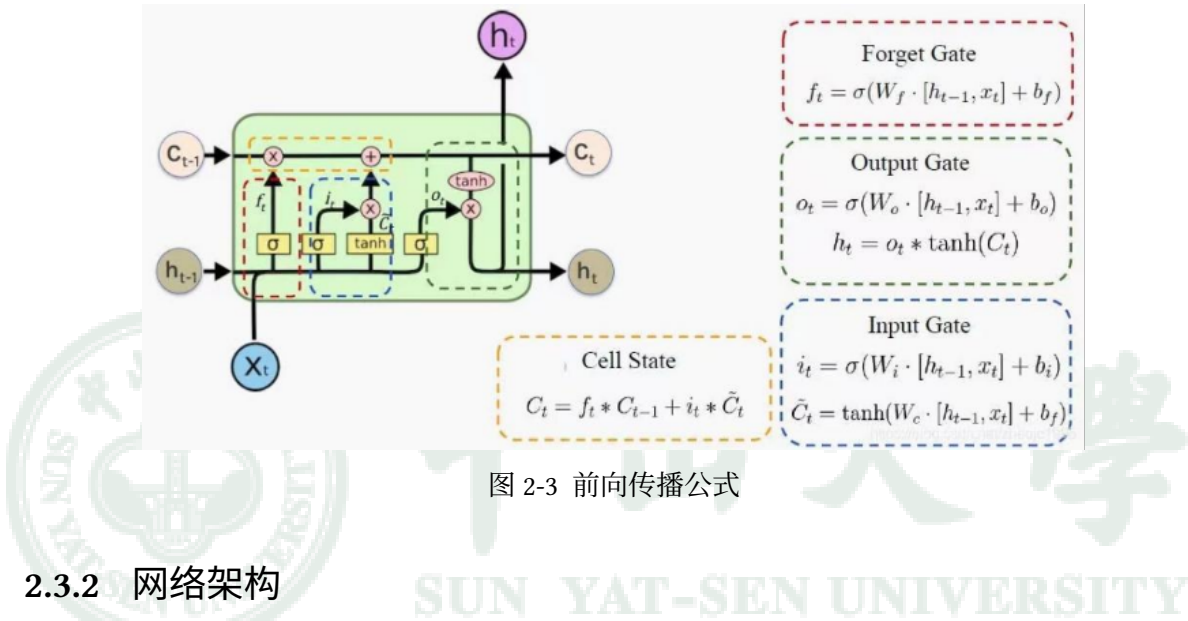


图 2-3 前向传播公式

2.3.2 网络架构

前向传播共六个公式如图2-3所示。LSTM 三个门的作用如下所述。

- 1. 遗忘门 f : 决定上一个时刻的记忆单元状态需要遗忘多少信息，保留多少信息到当前记忆单元状态。
- 2. 输入门 i : 控制当前时刻输入信息候选状态有多少信息需要保存到当前记忆单元状态。
- 3. 输出门 o : 控制当前时刻的记忆单元状态有多少信息需要输出给外部状态。

第 3 章 实验环境

实验环境如表格3-1所示。

操作系统	Ubuntu 20.04.4 LTS
编程语言	Python 3.8.10
深度学习框架	Keras 2.9.0/Tensorflow2.9.0
NLP 库	jieba 0.42.1
开发工具	Jupyter Notebook 6.4.12
版本控制	Git2.25.1

表 3-1 实验环境

第 4 章 数据集

4.1 来源

数据集来源于tatoeba.org的数据。

4.1.1 Tatoeba

Tatoeba 是一个保存海量句子和译文的数据库。因为有成千上万名社区成员共同志愿贡献，Tatoeba 的内容一直在不断成长。

Tatoeba 为用户提供例句工具，帮助用户理解单词如何在句子上下文中应用。用户输入感兴趣的单词以后，Tatoeba 就会返回包含该单词的句子，并且带有目标语言的译句。Tatoeba 这个名字（日语意为举例来说）完全契合网站理念。其数据按照多种 Creative Commons 许可证的标准公开。

4.1.2 ManyThings

此网站使用数据主要包含下面部分，

- 以单词及其词族为重点的英语句子
- 每日听力和重复发音练习
- 英语句子（带音频）
- 双语句子对
- 日英平行语料库（日英パラールコーパ）
- 搜索语料库
- 日语阅读与翻译练习
- 创作者试图过滤掉那些可能具有冒犯性且不适合所有年龄段的句子

4.2 结构和内容

本次实验共使用了两个数据集。

1. cmn.txt:

数据集共 20403 条数据，如图所示，存储在 cmn.txt 文件中，每行包含一个数据点，每个数据点由一个英文句子和一个对应的中文翻译组成。这些句子之间通过制表符。例如，文件中的第一行为'Hi。'，其中'Hi' 是英文句子，'嗨。' 是它的中文翻译。如图4-2所示。

2. cmn_zhsim:

数据集共 20133 条数据，存储在 cmn_zhsim.txt 文件中。它是基于 cmn.txt 修改而来。其继承了先前数据集的结构和内容，但针对中文字符进行了优化，将所有的繁体字转换为了简体字以适应简体中文读者或简体中文自然语言处理任务。如图4-1所示。

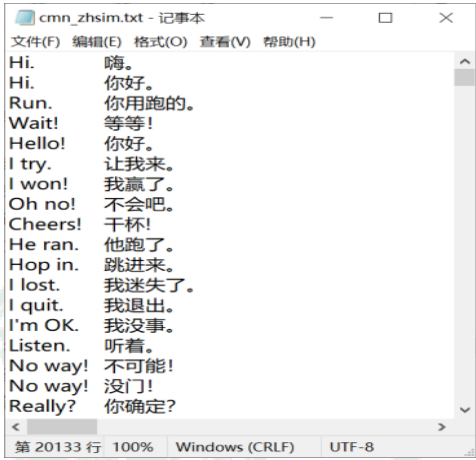


图 4-1 cmn_zhsim.txt

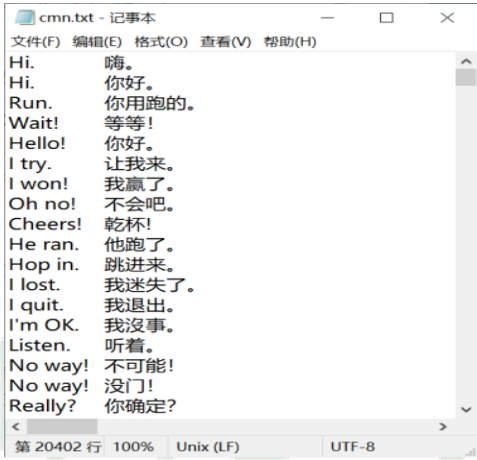


图 4-2 cmn.txt

第 5 章 实验过程

5.1 Seq2Seq 模型

5.1.1 实验总体设计

实验总体设计如图5-1所示。由两个主要由编码器和解码器组成。编码器负责处理输入序列，比如一个句子中的单词或字符，通过神经网络来捕捉序列中的时序信息。它的最终隐藏状态被认为是整个输入序列的上下文或“思想”的压缩表示。这个上下文状态然后传递给解码器，解码器逐步构建输出序列，即目标语言的翻译。

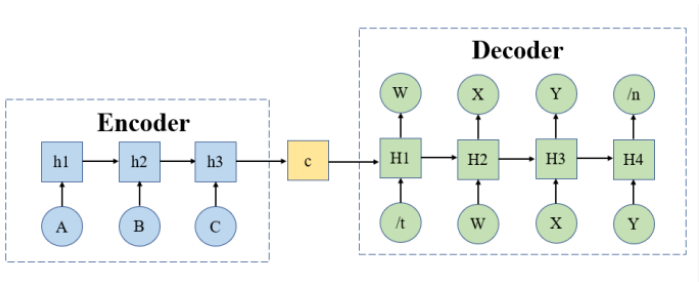


图 5-1 实验总体设计示意图

5.1.2 数据及环境准备

我们需要的数据均放在 `cmn_zhism.txt` 文件中，共 20403 条英中对应数据。

5.1.3 安装相关包

我们需要安装 `numpy`、`pandas` 及 `TensorFlow`。相关包版本如下

Package	Version
<code>numpy</code>	1.23.1
<code>pandas</code>	2.0.3
<code>tensorflow</code>	2.9.0

表 5-1 软件包版本

5.1.4 导入实验环境

我们从 `tensorflow.keras` 模块导入一些特定的函数和类, 以及导入 `pandas` 和 `numpy` 库。其中:

1. TensorFlow Keras Layers

- `Input`: 用于实例化一个用作模型的输入的 Keras 张量。
- `LSTM`: 长短期记忆网络 (LSTM) 层。
- `Dense`: 全连接层, 每个输入节点都与每个输出节点相连。

2. TensorFlow Keras Models

- `Model`: 这是构建所有神经网络模型的基类。
- `load_model`: 这个函数用于加载一个保存的 Keras 模型。

3. TensorFlow Keras Utilities

- `plot_model`: 这个函数用于生成一个模型的结构图, 可视化神经网络的层次结构和层之间的连接。

5.1.5 配置参数

我们定义 LSTM 层中隐藏单元的数量、每次训练过程中网络看到的样本数量、迭代次数、训练过程中用到的样本总数以及存储训练数据的文件路径。代码如下:

```

1 N_UNITS = 256      #LSTM单元中隐藏层节点数
2 BATCH_SIZE = 64    #网络训练时每个批次的大小
3 EPOCH = 30         #训练的epoch数
4 NUM_SAMPLES = 10000 #训练数据数量, 即样本条数
5 data_path = './cmn_zhsim.txt' #训练数据路径(根据的路径设置)

```

5.1.6 读取数据并处理

1. 数据加载

使用 `read_table` 函数从指定路径加载数据, 数据没有表头, 所以使用 `header=None`。取前 `NUM_SAMPLES` 条数据, 然后给数据框的两列分别命名为 'inputs' 和 'targets', 分别表示英文输入句子和中文目标句子。代码如下:

```
1 df=pd.read_table(data_path,header=None).iloc[:NUM_SAMPLES,:]  
2 df.columns=['inputs','targets']  
3
```

2. 添加起始和终止标志

对目标句子(中文)添加特殊字符“”和”, 分别表示句子的开始和结束。并将输入和目标列转换为列表, 以便后续处理。代码如下:

```
1 df['targets'] = df['targets'].apply(lambda x: '\t'+x+'\n')  
2 #英文句子列表  
3 input_texts = df.inputs.values.tolist()  
4 #中文句子列表  
5 target_texts = df.targets.values.tolist()  
6
```

3. 确定字符集

使用 `.unique().sum()` 将所有唯一的句子合并为一个长句子, 再使用 `set()` 提取不重复的字符, 最后排序。这样可以得到输入(英文)和目标(中文)的字符集。代码如下:

```
1 input_characters = sorted(list(set(df.inputs.unique().sum())))  
2 target_characters = sorted(list(set(df.targets.unique().sum())))  
3
```

4. 确定维度计算输入和目标文本中最长的句子长度, 以便确定网络的时间步长。同时计算输入和输出字符的数量, 以定义模型的输入和输出层的维度。代码如下:

```
1 #输入数据的时刻t的长度, 这里为最长的英文句子长度  
2 INPUT_LENGTH = max([len(i) for i in input_texts])  
3
```

```

4 #输出数据的时刻t的长度，这里为最长的中文句子长度
5 OUTPUT_LENGTH = max([len(i) for i in target_texts])
6
7 #每个时刻进入encoder的lstm单元的数据xt的维度，这里为英文中出现的字符数
8 INPUT_FEATURE_LENGTH = len(input_characters)
9
10 #每个时刻进入decoder的lstm单元的数据txt的维度，这里为中文中出现的字符数
11 OUTPUT_FEATURE_LENGTH = len(target_characters)
12

```

5.1.7 句子向量化

1. 初始化输入输出张量

- `encoder_input`: 为编码器输入预留空间，其形状由样本数、输入句子的最大长度和输入字符集的大小决定。
- `decoder_input` 和 `decoder_output`: 为解码器输入和输出预留空间，形状由样本数、输出句子的最大长度和输出字符集的大小决定。

代码如下：

```

1 encoder_input =
2 np.zeros((NUM_SAMPLES, INPUT_LENGTH, INPUT_FEATURE_LENGTH))
3 decoder_input =
4     np.zeros((NUM_SAMPLES, OUTPUT_LENGTH, OUTPUT_FEATURE_LENGTH))
5 decoder_output =
6     np.zeros((NUM_SAMPLES, OUTPUT_LENGTH, OUTPUT_FEATURE_LENGTH))
7

```

- #### 2. 构建字符到索引的映射字典
- 这些字典用于将字符转换为相应的索引和反向查找。`input_dict` 和 `target_dict` 将字符映射到一个唯一的索引，`input_dict_reverse` 和 `target_dict_reverse` 用于从索引回到字符。代码如下：

```

1 #构建英文字符集的字典
2 input_dict = {char:index for index,char in enumerate(input_characters)}
3 #键值调换
4 input_dict_reverse = {index:char for index,char in
5     enumerate(input_characters)}
6 #构建中文字符集的字典
7 target_dict = {char:index for index,char in enumerate(target_characters)}
8

```

```

7 #键值调换
8 target_dict_reverse = {index:char for index,char in
    enumerate(target_characters)}
9

```

3. 字符级 one-hot 编码对输入和输出数据进行 one-hot 编码是将字符转换为二进制向量，其中只有对应该字符索引的位置为 1，其余为 0。代码如下：

```

1     #encoder的输入向量one-hot
2 for seq_index,seq in enumerate(input_texts):
3     for char_index, char in enumerate(seq):
4         encoder_input[seq_index,char_index,input_dict[char]] = 1
5 #decoder的输入输出向量one-hot,
    训练模型时decoder的输入要比输出晚一个时间步，这样才能对输出监督
6 for seq_index,seq in enumerate(target_texts):
7     for char_index,char in enumerate(seq):
8         decoder_input[seq_index,char_index,target_dict[char]] = 1.0
9         if char_index > 0:
10             decoder_output[seq_index,char_index-1,target_dict[char]] = 1.0
11

```

5.1.8 创建模型

我们定义一个序列到序列 (seq2seq) 模型 `create_model()`，如图5-2所示。这个模型是在 Keras 框架下实现的，下图详细地定义了训练阶段和推断阶段的不同网络结构。

5.1.9 编译模型

我们为模型设置 Adam 优化器和分类交叉熵损失函数。接着查看模型结构。代码如下：

```

1 model_train.compile(optimizer='adam',loss='categorical_crossentropy')

```

1. 训练的模型

代码如下：

```

1 model_train.summary()

```




图 5-2 create_model() 函数

模型结构如图5-3所示，其中英文字符（含数字和符号）共有 73 个，中文中字符（含数字和符号）共有 2128 个，LSTM 中隐藏层节点数为 256 个，全连接层有 2128 个单元。

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, None, 73)]	0	[]
input_2 (InputLayer)	[(None, None, 2128)]	0	[]
lstm (LSTM)	[(None, 256), (None, 256), (None, 256)]	337920	['input_1[0][0]']
lstm_1 (LSTM)	[(None, None, 256), (None, 256), (None, 256)]	2442240	['input_2[0][0]', 'lstm[0][1]', 'lstm[0][2]']
dense (Dense)	(None, None, 2128)	546896	['lstm_1[0][0]']

=====
Total params: 3,327,056
Trainable params: 3,327,056
Non-trainable params: 0

图 5-3 训练模型的结构

2. 编码器推理模型

代码如下：

```
1 encoder_infer.summary()
```


模型结构如图5-4所示，其中 LSTM 中影藏层节点数为 256 个。

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, None, 73)]	0
lstm (LSTM)	[(None, 256), (None, 256), (None, 256)]	337920
Total params: 337,920 Trainable params: 337,920 Non-trainable params: 0		

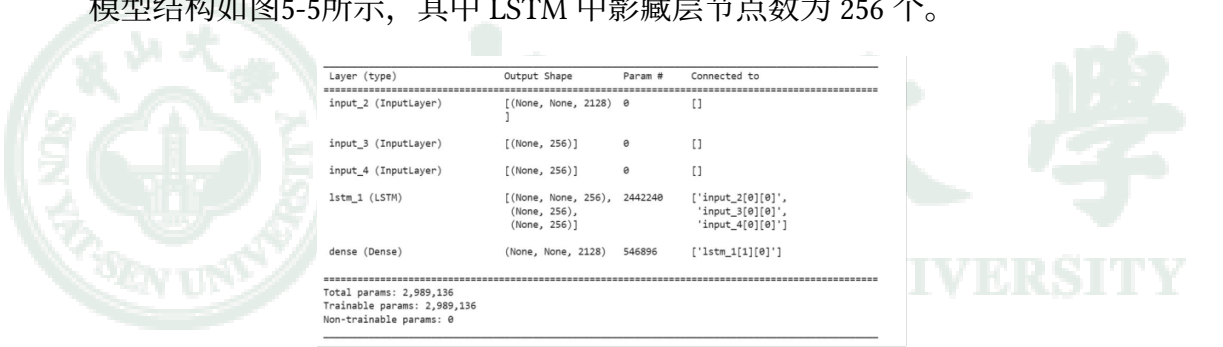
图 5-4 编码器推理模型的结构

3. 解码器推理模型

代码如下：

```
1 decoder_infer.summary()
```

模型结构如图5-5所示，其中 LSTM 中影藏层节点数为 256 个。



Layer (type)	Output Shape	Param #	Connected to
input_2 (InputLayer)	[(None, None, 2128)]	0	[]
input_3 (InputLayer)	[(None, 256)]	0	[]
input_4 (InputLayer)	[(None, 256)]	0	[]
lstm_1 (LSTM)	[(None, None, 256), (None, 256), (None, 256)]	2442240	['input_2[0][0]', 'input_3[0][0]', 'input_4[0][0]']
dense (Dense)	(None, None, 2128)	546896	['lstm_1[1][0]']
Total params: 2,989,136 Trainable params: 2,989,136 Non-trainable params: 0			

图 5-5 解码器推理模型的结构

5.1.10 模型训练

在每次训练轮次结束时，模型将 20% 的训练数据留出作为验证集。代码如下：

```
1 history = model_train.fit([encoder_input,decoder_input],decoder_output,  
2 batch_size=BATCH_SIZE, epochs=EPOCH,validation_split=0.2)
```

5.1.11 损失曲线绘制

代码如下：

```
1 plt.plot(epochs, train_loss , 'b', label='Training loss')  
2 plt.plot(epochs, val_loss , 'r', label='Validation loss')
```

```

3 plt.title('Training and Validation Loss')
4 plt.xlabel('Epochs')
5 plt.ylabel('Loss')
6 plt.legend()
7 # 保存图片文件
8 plt.savefig('training_and_validation_loss_plot.png')
9 plt.show()

```

图5-6展示了当 batch=32, epoch=500 时的损失函数曲线。



图 5-6 损失函数曲线 (batch=32, epoch=500)

5.1.12 构建测试函数

定义 `predict_chinese()` 函数，每次解码步骤只预测一个字符，并使用解码器上一个时间步的输出和状态来预测下一个字符，直至序列完成或遇到结束标记。代码如下：

```

1  def predict_chinese
2  (source, encoder_inference, decoder_inference, n_steps, features):
3      #先通过推理encoder获得预测输入序列的隐状态
4      state = encoder_inference.predict(source)
5      #第一个字符'\t',为起始标志
6      predict_seq = np.zeros((1,1, features))
7      predict_seq[0,0, target_dict['\t']] = 1
8
9      output = ''
10     #开始对encoder获得的隐状态进行推理

```

```

11 #每次循环用上次预测的字符作为输入来预测下一次的字符，直到预测出了终止符
12 for i in range(n_steps): #n_steps为句子最大长度
13     #给decoder输入上一个时刻的h,c隐状态，以及上一次的预测字符
14     yhat,h,c = decoder_inference.predict([predict_seq]+state)
15     #注意，这里的yhat为Dense之后输出的结果，因此与h不同
16     #每次预测都是最后一个中文字符
17     char_index = np.argmax(yhat[0,-1,:])
18     char = target_dict_reverse[char_index]
19     output += char
20     #更新state，本次状态做为下一次的初始状态继续传递
21     state = [h,c]
22     #前一状态的输出结果会作为下一状态的输入信息
23     predict_seq = np.zeros((1,1, features))
24     predict_seq[0,0, char_index] = 1
25     if char == '\n':#预测到了终止符则停下来
26         break
27 return output

```

5.1.13 测试

代码如下：

```

1 for i in range(100,200):
2     test = encoder_input[i:i+1,,:]) #i:i+1保持数组是三维
3     out = predict_chinese(
4 test , encoder_infer , decoder_infer , OUTPUT_LENGTH,OUTPUT_FEATURE_LENGTH)
5     print(input_texts[i])
6     print(out)

```

结果如图5-7所示。

```
1/1 [=====] - 0s 364ms/step
1/1 [=====] - 0s 330ms/step
1/1 [=====] - 0s 27ms/step
1/1 [=====] - 0s 30ms/step
1/1 [=====] - 0s 26ms/step
1/1 [=====] - 0s 28ms/step
Try some.
试试吧。

1/1 [=====] - 0s 25ms/step
1/1 [=====] - 0s 27ms/step
1/1 [=====] - 0s 28ms/step
1/1 [=====] - 0s 26ms/step
1/1 [=====] - 0s 26ms/step
1/1 [=====] - 0s 26ms/step
Who died?
谁死了?

1/1 [=====] - 0s 25ms/step
1/1 [=====] - 0s 29ms/step
1/1 [=====] - 0s 30ms/step
1/1 [=====] - 0s 28ms/step
1/1 [=====] - 0s 27ms/step
1/1 [=====] - 0s 27ms/step
1/1 [=====] - 0s 26ms/step
Birds fly.
鸟类飞行。
```

图 5-7 Seq2Seq 模型结果

第 6 章 模型优化与评估

6.1 seq2seq

在基本的 Seq2Seq 模型中，我们调试所得到的最优训练超参数如下：

N_UNITS	256
BATCH_SIZE	64
EPOCH	300

表 6-1 超参设置

在该参数下的模型训练集与验证集的 loss 曲线如图6-1所示，accuracy 曲线如图6-2所示。观察以上曲线我们可以看到在约第 130 个 epoch 的迭代后，模型的性能在验证

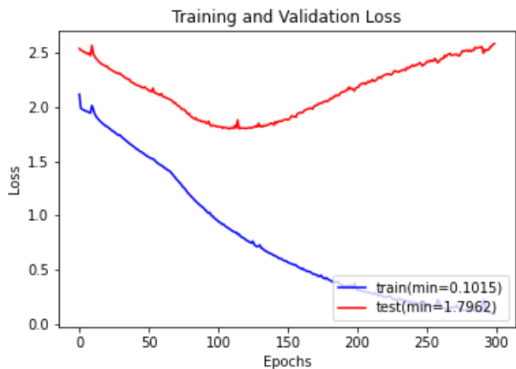


图 6-1 Loss.

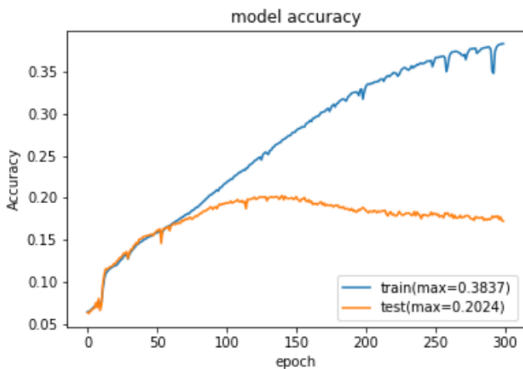


图 6-2 Accuracy.

图 6-3 Performance Metrics

集上达到最优的性能，在测试集上达到的最小 loss 为 1.7962，最大 accuracy 为 20.24%。
通过训练中的性能指标可以看出，尽管在训练集上已经快要过拟合的情况下，模型的性能表现却依然不甚出色，尤其是在对于长难句的翻译上面。我们可以通过对于

数据集中具体的翻译实例看出这一特点。此外，我们发现当我们增大样本条数时，也

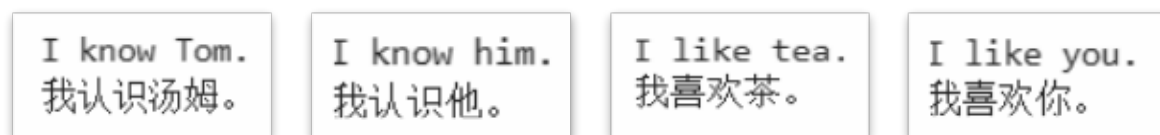


图 6-4 简单翻译实例

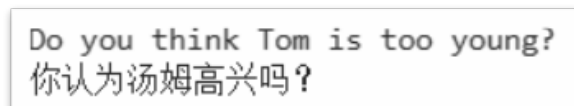


图 6-5 复杂翻译实例

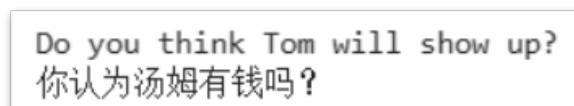


图 6-6 复杂翻译实例

会导致训练出来的模型翻译性能下降，与惯性思维下数据量越大模型性能越好的认知相悖。这是因为在 cmn_zhsim 数据集中，整体上越靠后的句子就会越长越复杂，而 Seq2Seq 模型对长难句的编码-解码过程中存在信息丢失，导致对于数据集的训练效果十分不好。这还会导致最终的模型在简单句上也出现翻译不恰当的情况，如图6-8所示。

以上原因，均是由于在 Seq2Seq 模型中，只用固定大小的状态向量来连接编码模块和解码模块，要求编码器将整个输入序列的信息压缩到状态向量中。在这个过程中是一个有损压缩，序列越长，编码损失就越大，当句子过长时会导致编码器无法记录足够详细的信息，进而导致解码器翻译失败。此外，随着输入序列长度的增加，也会导致编码和解码过程中出现的梯度消失和梯度爆炸问题愈发严重。这是导致 Seq2Seq 模型在长难句上翻译效果急剧下降的主要原因。

为了解决这一问题，我们还尝试了在 seq2seq 模型的基础上添加了 attention 机制的模型，期望添加了 attention 机制下的 seq2seq 模型能够取得更加优秀的表现。下面是我们加入了 attention 机制后的模型评估，我们将在其中感受到 attention 机制在长难句的翻译表现上的巨大提升。

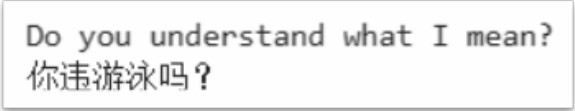


图 6-7 复杂翻译实例



图 6-8 信息丢失

6.2 Attention 机制

在加入 attention 机制的 Seq2Seq 模型中，我们调试所得到的最优训练超参数如6-2所示。

N_UNITS	256
BATCH_SIZE	64
EPOCH	300

表 6-2 超参数设置

在该过程中，为了更加突出地表现模型在加入 attention 机制前后性能上的差别以及运行过程中实时监测在每个 batch 中的模型训练情况，我们在实验材料中代码上做了细微的改动与优化

- 1. 修改了 loss_function
- 2. 添加了性能评估函数 acc_function
- 3. 添加了验证集前向传播函数，用于计算模型在加入了 attention 机制后在验证集上的 loss 和 accuracy
- 4. 添加了测试集与验证集训练过程中训练时间与 loss 和 accuracy 参数的实时显示

在对模型进行调参与优化后，模型训练集与验证集的 loss 曲线如图6-9所示， accuracy 曲线如图6-10所示。对比基本的 Seq2Seq 模型我们可以发现，加入了 attention 机制以后，该模型的训练效果得到了显著的提升。在加入了 attention 机制后，模型很快就开始收敛，并且基本不存在过拟合的现象。在大约第 8 个 epoch 时模型就已达达到最优性能，此时最小的验证集 loss 为 0.1038，最高 accuracy 为 93.88%。这已经是机器翻译中非常好的一个水平了。

利用训练好的模型进行对实例的机器翻译我们可以发现，加入 attention 机制后，模型不仅仅对于简单句可以实现准确翻译，对于长难句，甚至在包含复句的情况下，也有了更好的翻译效果。以下是一些加入 attention 机制后的 Seq2Seq 模型翻译实例如图6-11所示。

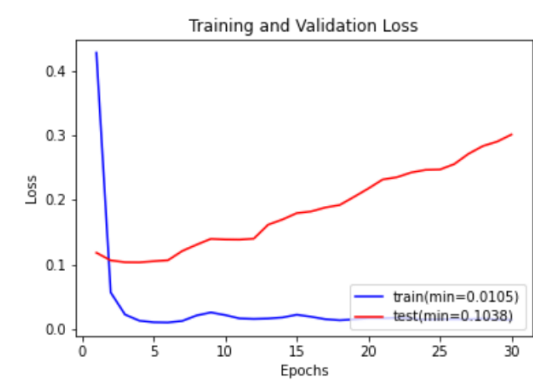


图 6-9 Loss.

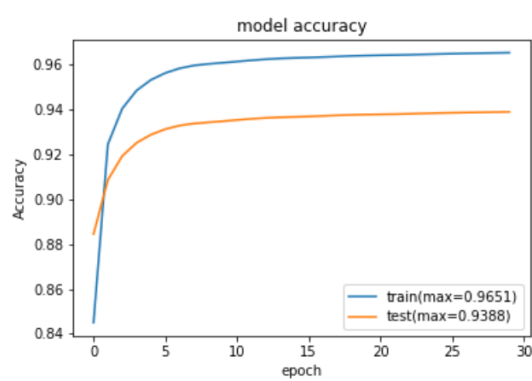


图 6-10 Accuracy.

输入: <start> 汤姆 试过 还 回 泳衣 来 换成 更大 一号 的 , 但是 员工 告诉 他 那 是 不 被 允 许 的 . <end>
翻译结果: tom tried to return the swimsuit for a larger size , but the clerk told him that wasn t allowed . <end>

图 6-11 Attention 机制翻译实例

我们可以发现，尽管句子中的单词数已经达到十个之多，模型仍然具有非常优秀的翻译效果。

利用 attention 机制，模型能够学习到一个注意力分布，即模型在生成每个目标词时，对源句子中的每个词的关注程度。此时模型能够更好地处理长句子和源目标语言之间的复杂关系，特别是当句子长度超过训练语料中的句子时。除此以外，加入注意力机制后，模型还能够生成与人类直觉相符的语言上合理的对齐。这就是加入了 attention 机制后，模型在机器翻译上性能得到明显提升的主要原因。

为了方便地观察在 attention 机制中地注意力分布情况，我们将注意力分布绘制成图表进行显示。以上翻译实例的注意力分布如图6-12所示。在注意力机制下，模型在

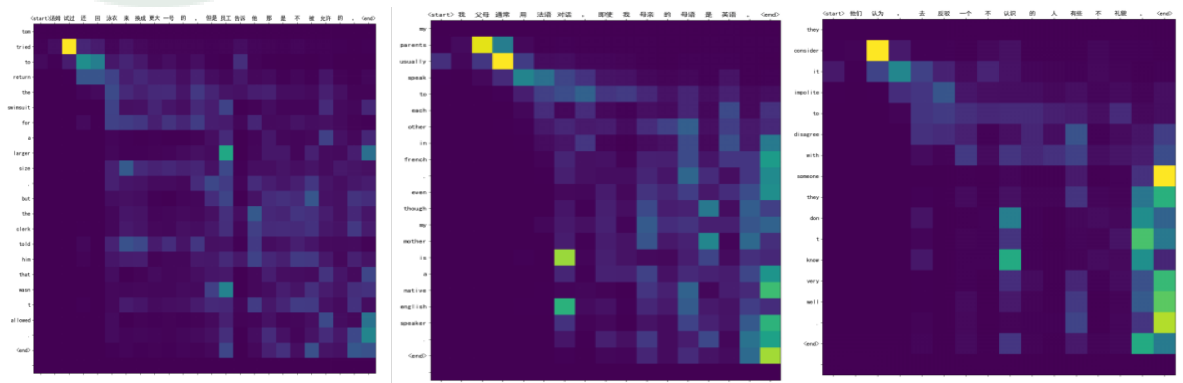


图 6-12 Relation map.

生成目标句子的每个词时，会软搜索源句子中与预测目标词最相关的部分。观察上图，颜色越明亮的区域说明这些词横纵坐标对应的中英文在句子中的对应关系越紧密。一般来说，这些词例如 [“试过”，“try”]、[“父母”，“parents”]、[“通常”，“usually”] 和 [“认识”，“consider”] 等。它们在注意力机制中的分布相关性极高，在模型生成翻译句的时候会重点关注这些相关性高的词，从而提升在长难句中的翻译效果。

第 7 章 思考题

结合 Seq2Seq 框架及内部使用单元思考, Seq2Seq 框架在编码-解码过程中是否存在信息丢失? 具体表现在哪些方面?

先给出结论, 严格来讲, 如果这里的“信息丢失”指的是在源语句中的信息是否没有被利用, 那么则不存在“信息丢失”。但是针对每个源语句中的单词, 则每个单词都有不同程度的信息丢失。

因为 seq2seq 模型的 encoder 将整个序列压缩到特定大小的向量 c 中, 因此序列中的所有单词信息都在 c 中有所体现, 从这个角度来讲, seq2seq 模型不存在“信息丢失”的问题。

但是因为该模型有以下特征, 所以对序列中的所有信息都存在不同程度的丢失。

1. 只用固定大小的状态向量 c 来连接 Encoder 和 Decoder, 这就要求 Encoder 将整个输入序列的信息有损压缩到状态向量中, 序列越长, 信息量越大, 编码的损失就越大。
2. 在输入较长序列的时候, 会出现梯度消失和梯度爆炸的问题。
 - 梯度爆炸: 在反向传播过程中, 梯度需要通过序列的每个时间步进行传播。如果每个时间步的权重矩阵的梯度乘积大于 1, 那么随着时间步的增加, 梯度将呈指数增长。
 - 梯度消失: 在 RNN 中, 每个时间步的权重矩阵都会与前一个时间步的梯度相乘。如果权重矩阵的值接近于 0 或 1, 那么随着时间步的增加, 梯度可能会迅速减小。

由于模型本身结构能力的限制, 导致必然会出现一定程度的信息丢失。为了缓解这种瓶颈, 至少在两个方面可以努力。

1. 使用更优秀的、带有记忆功能的编码器, 比如 LSTM, 但是 LSTM 也只能实现短期记忆, 对长序列处理能力仍旧不足。
2. 使用注意力机制, 利用注意力机制可以更好地提取变量间的局部注意力关系, 生成更好的状态向量。

第 8 章 开放题

在介绍这篇文章之前, 我们有必要先简答了解一下 Machine Translation 这个任务的主流方向。该任务主要有两种解决的策略, 如下所述。

1. SMT: Statistical Machine Translation, 这是一种基于统计学的机器翻译方法, 它依赖于大量的平行语料库。主要通过平行语料库学习源词和短语与目标词和短语的 alignment。利用这些 alignment 构建统计模型。
2. NMT: Neural Machine Learning, 利用 DL 端到端地学习 machine translation, 不需要依赖于既定的 rules 和 phrase, 通常使用各类 RNN 来进行训练。这是一种 Data-driven 的机器翻译方法。

目前比较有影响力的方向是 NMT, 在 2014 年, 有一篇文章《Sequence to Sequence Learning with Neural Networks》提出了一种端到端的 Machine Translation 方法。该方法首先使用一个 LSTM 将输入序列映射到一个固定维度的向量, 然后使用另一个深层 LSTM 从该向量解码目标序列。模型的 pipeline 如图8-1所示。

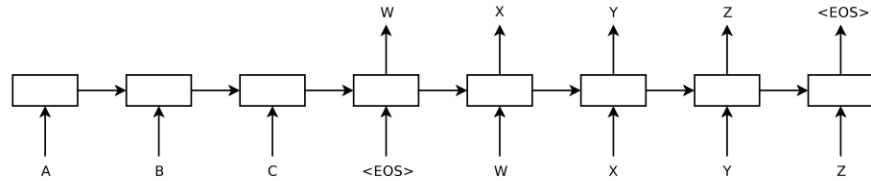


图 8-1 This model reads an input sentence “ABC” and produces “WXYZ” as the output sentence.

在 WMT’ 14 英语到法语的翻译任务上, 该模型达到了 34.8 的 BLEU 分数, 并且在处理长句子时没有困难。与此相比, 基于短语的统计机器翻译在同一数据集上的 BLEU 分数为 33.3。课件相比于是 SMT, 该方法有相对的优势。

在该 seq2seq 工作的基础上, 2015 年的《NEURAL MACHINE TRANSLATION BY JOINTLY LEARNING TO ALIGN AND TRANSLATE》工作在 seq2seq 的基础上添加了 Attention 机制, 这是 Machine Translation 领域首次引入 Attention Mechanism。

这篇工作指出了指出了传统的基于 encoder-decoder 的 NMT 方法中, 将源句子编码成固定长度向量可能成为提升性能的瓶颈。该文章提出了一种新的 NMT 模型, 该模型能够联合学习 alignment 和 translation。在生成翻译词的同时, 模型会(软)搜索源句子中与预测目标词最相关的部分, 而不必显式地将这些部分形成硬编码的段落。下面我们将具体介绍该工作的细节, 从而准确理解什么是“(软)搜索源句子中与预测目标词最相关的部分”。

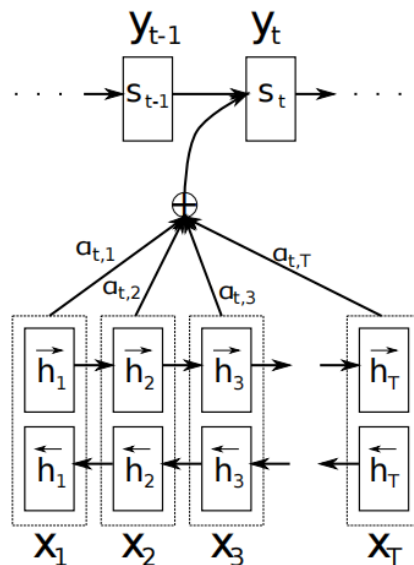


图 8-2 The graphical illustration of the proposed model trying to generate the t-th target word y_t given a source sentence.

如图8-2所示。在该模型中，将条件概率定义为8-1。

$$p(y_i | y_1, \dots, y_{i-1}, x) = g(y_{i-1}, s_i, c_i) \quad (8-1)$$

在 encoder 中， $h_j = [\vec{h}_j^\top; \overleftarrow{h}_j^\top]^\top$ 。其中的 \vec{h}_j^\top 和 \overleftarrow{h}_j^\top 都是由双向循环网络产生的。每一个输入 X_j 就会产生一个 h_j ， h_j 即为一种 annotation，再将各个 annotation 按照下面所述的公式进行加权得到 c_j ， c_j 即为源语句中与目标搜索语句最相关的部分的特征。值得注意的是， $a_{i,j}$ 是按照下面公式计算的，其中的 $e_{i,j}$ 是 trainable 的前馈神经网络。通过神经网络对对齐和翻译的联系训练，最终可以将当前 $t - th$ 的最相关的部分编码成特征，并从特征中学习目标语句，从而将 attention 机制引入 SMT 中。

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}$$

(8-2)

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

(8-3)

可以做如下总结，该模型就是通过将 soft-alignment 和可变长度特征引入，从而实现了 attention 机制，主要体现在下面几个方面

1. Soft Alignment: 模型在生成目标句子的每个词时，会 soft-search 源句子中与预测目标词最相关的部分。这种软对齐方式允许模型在生成翻译时动态地聚焦于源句子的不同位置，而不是简单地将整个源句子编码成一个固定长度的向量。
2. Context Vector: 对于目标词的每个生成步骤，模型会计算一个上下文向量，该向量是基于源句子中所有位置的 annotations 的加权和。这些注释是由编码器网络生成的，包含了源句子的上下文信息。
3. Weight Computation: 上下文向量的每个注释的权重是通过一个 alignment model 计算的，该模型使用前一个目标词的隐藏状态和源句子中每个位置的注释来评估匹配程度。权重反映了源句子中每个位置对于生成当前目标词的重要性。
4. Attention Distribution: 通过这种方式，模型能够学习到一个注意力分布，即模型在生成每个目标词时，对源句子中的每个词的关注程度。这种分布是 soft 的，意味着它可以将注意力分散在多个源句子的词上，而不仅仅是一个固定的对齐点。

通过这种 attention 机制，模型能够更好地处理长句子和源目标语言之间的复杂关系，特别是当句子长度超过训练语料中的句子时，传统的编码器-解码器模型可能会

遇到困难。论文中的实验结果表明，提出的模型在长句子翻译任务上的性能显著优于基本的编码器-解码器模型，并且能够生成与人类直觉相符的语言上合理的对齐。

第9章 心得体会

实验前，我们首先开展了小组会议。经过深入讨论和考量，最终决定选择机器翻译作为我们自然语言处理课程的课题。一个好的课题不仅要有足够的深度和广度，还要能激发我们的研究兴趣。在讨论过程中，我们认真评估了每个课题的可行性、创新性和实用性，最终选择了机器翻译。我们一致认为这个课题具有广泛的应用价值、技术挑战性、研究热点与前沿性、丰富的实验资源、跨学科融合、商业价值和产业需求以及技术迭代与创新等多方面的优势。

在选题过程中，我们充分发挥了团队合作的优势。每个成员都积极发表自己的意见和看法，对各个课题进行了深入的分析和比较。我们相互启发，共同思考，也明确了分工。

实验中，我们主要探索了两个模块：seq2seq 模型和加入了 Attention 机制的 seq2seq 模型。这个过程让我们对这两个模型有了更深入的理解。由于实验手册已经给出了实现 seq2seq 模型和加入 Attention 机制的模型的代码，所以我们实验的重点更多地放在理解代码、调整参数、进行实验和结果分析上。以下是我们遇到的难点及解决方案的详细描述。

9.1 理解代码结构

当面对一个现成的代码库时，我们需要理解其整体结构和各个模块的功能。通过阅读代码中的注释和相关的文档，我们得以了解代码的目的、设计思路和关键部分的实现。然后通过逐步运行代码的不同部分，并观察输出结果，来深入理解代码的执行流程。

9.2 参数调整与优化

即使代码已经给出，但参数的设置和调整仍然是影响模型性能的关键因素。我们首先了解代码中各个参数的含义和作用，这有助于我们更有目的地调整参数。同时通过尝试不同的参数组合，观察其对模型性能的影响，并找到最优的参数设置。

9.3 实验结果的评估与分析

如何评估和分析实验结果，以判断模型是否达到预期的性能，也是一个需要关注的难点。这里面代码没有给出评估指标，所以需要我们自己探索。我们通过绘制损失曲线、准确率曲线等图表，更直观地观察模型的训练过程和性能变化。此外，我们还分析模型在测试集上的错误，了解模型在哪些情况下容易出错，从而指导后续的改进方向。

实验结束后，我们对本次实验进行了复盘。我们感受到实验的顺利进行离不开成员之间的沟通和交流，无论是线上还是线下，我们都会及时分享工作进展、遇到的问题

题和解决方案。同时也感受到了彼此所展现出的优秀专业素养。这种专业素养不仅体现在每个人的专业技能上，更体现在大家对待工作的态度、解决问题的能力 and 团队协作的精神上。

9.4 论文阅读

通过本次实验，我认为 paper reading 主要包含两个重要的部分，分别是 read inside 和 read outside。read inside 是通过阅读文章本身了解主要的技术路线和创新点。read outside 是站在文章之外，读懂相关领域的发展 timeline，后一种技术路线为何替代前一种技术路线，从而能够了解相关领域的瓶颈和发展方向。

第 10 章 Labor Contribution

姓名	学号	contribution
黄楚丹	21312024	介绍、实验原理、实验过程
沈鹏飞	21312188	模型训练与测试、模型优化、模型性能评估与分析
刘书睿	21312295	开放题、思考题、typeset、心得体会-论文阅读

表 10-1 Labor Contribution.



中山大學
SUN YAT-SEN UNIVERSITY