上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

学士学位论文

BACHELOR'S THESIS

论文题目 运用隐式注意模型进行 If-Then 程序的合成

学生姓名 陈昕昀

学生学号 5130309066

指导教师 俞勇, Dawn Song

专业 致远计算机科学班

学院（系） 致远学院

运用隐式注意模型进行 If-Then 程序的合成

摘要

从自然语言到程序的自动翻译是一个存在已久且具有挑战性的问题。本课题将考虑一个简单、然而重要的子问题：从文字描述到 If-Then(如果-则) 程序的翻译。当前已有一些网站允许用户通过编写这样的 If-Then 程序来更加便利地规划自己的生活，这样的网站包括 IFTTT、Zapier、Stringify 等。近年来已有一些工作对从自然语言描述到 If-Then 程序的翻译这一问题进行了研究；然而，自然语言庞大的词汇量和复杂的语言结构使得这个问题具有挑战性。在这篇毕业论文中，我设计了一个新的神经网络模型，命名为“隐式注意模型 (Latent Attention)”。在该模型中，一段语言描述里的每个词的权重被分两步进行计算，旨在更好地运用自然语言的结构来分析出一段描述中对于最终程序的合成最关键的子片段。实验结果表明，这一新的模型在 IFTTT 数据集上相对于之前最好的工作错误率降低了 28.57%。除此之外，我还针对这一任务提出了一个新的训练场景，叫“单样例学习 (One-shot Learning)”，并用已有的数据集进行实验来模拟这一场景。对于这一训练数据较少的场景，本毕业论文提出了一种新的模型训练方法，这一训练方法使得模型在这样的场景下与传统方法训练出来的相比效果有非常显著的提高。

关键词： 程序合成，如果-则程序，IFTTT，注意模型，单样例学习

Latent Attention For If-Then Program Synthesis

ABSTRACT

Automatic translation from natural language descriptions into programs is a long-standing challenging problem. In this thesis, I consider a simple yet important sub-problem: translation from textual descriptions to If-Then programs. Recent years have witnessed the emergence of several commercial websites towards this goal, such as IFTTT, Zapier and Stringify. Recent work studied the problem of automatically synthesizing If-Then programs from their descriptions; however, the diversity of vocabulary and sentence structures make this task challenging. In this thesis, I devise a novel neural network architecture for this task, which is called Latent Attention. This model computes multiplicative weights for the words in the description in a two-stage process, with the goal of better leveraging the natural language structures that indicate the relevant parts for predicting program elements. My experimental results demonstrate that on IFTTT dataset, this proposed architecture reduces the error rate by 28.57% compared to prior art. Meanwhile, I propose a one-shot learning scenario of If-Then program synthesis, and simulate it with the existing datasets. For this scenario when the training data is very insufficient, I demonstrate a variation on the training procedure that outperforms the original procedure, significantly closing the gap to the model trained with all data.

KEY WORDS: program synthesis, If-Then program, IFTTT, attention model, one-shot learning

目录

表格

插图

第一章　绪论

在当今这个信息时代，程序无处不在，但并不是每个人都能够熟练进行编程。因此，将编程系统变得更加友好、变得更加容易进行程序编写，是一个亟待发展的方向；尤其是对于尚未掌握编程技能的人来说，更加意义重大。在过去的数十年中，无论是自然语言处理领域，还是编程语言领域，对于这一发展方向，都不断有新的相关工作涌现出来。其中，从自然语言描述到程序的翻译是一个重要而有挑战性的任务，也是一个广受关注的课题。在这篇毕业论文中，我集中研究对于一种形式简单、然而重要的程序的生成，这样的程序叫做 If-Then (如果-则）程序。

If-Then 程序，顾名思义，就是由 If-Then 语句构成的程序。每一个 If-Then 语句由两部分组成。第一部分、也就是语句中的 If 部分中，包含了一个触发函数 (trigger function)；而第二部分、也就是语句中的 Then 部分中，则包含了一个动作函数（action function)。这样一个 If-Then 语句的意思是当触发函数所指定的条件满足时，便执行动作函数中的内容。这样的语句可以方便地让人们根据自己的生活习惯个性化定制一些执行任务，比如进行气象监控、文件整理、日程安排等等。当前已有一些网站允许用户通过编写这样的 If-Then 程序来更加便利地规划自己的生活，这样的网站包括 IFTTT、Zapier、Stringify 等。在这些网站上，用户一般都会为自己编写的 If-Then 程序附上自然语言描述。然而，正如我在前文所说，更加友好的程序编写方式应该是用户只需要提供自然语言描述，相应的 If-Then 程序便可以被自动编写出来。近年来，有不少工作都在研究从自然语言描述到 If-Then 程序的翻译这一课题 [1–6]。已有的工作主要运用三类方法来解决这一问题。第一类方法主要运用了统计学、或者传统机器学习的模型来完成这一任务 [1]。虽然这类方法从一定程度上能够成功地预测出一些较为常见的程序，然而总体来说效果仍有极大的改进空间。近年来，深度神经网络被运用于各种不同的领域，并在很多任务上取得了比传统方法更好的效果。特别地，循环神经网络 [7, 8] 以及基于循环神经网络的序列到序列翻译模型 [9] 在诸多应用上取得引人注目的效果，比如机器翻译 [9]、自动图像说明的生成 [10]、句法解析 [11]、以及智能答疑 [12]。基于这类模型在其它应用上的优秀性能，第二类方法便是运用基于经典序列到序列翻译模型的改良模型进行预测 [4–6]。然

而，I. Beltagy 等人在 [3] 中指出，对于自然语言描述到 If-Then 程序这一任务而言，由于自然语言词汇量的庞大和句法结构的复杂，基于循环神经网络的模型很难对这样的任务进行有效的学习。因此，他们提出了第三类方法，便是同时运用传统机器学习模型与深度神经网络模型进行预测 [3]。具体来说，他们运用一个全连接神经网络和一个逻辑回归的组合模型进行预测，并且这一方法在针对 If-Then 程序的函数预测这一任务上取得了比第二类方法更好的效果。

在这篇论文中，为了完成从自然语言描述到 If-Then 程序的翻译这一任务，我设计了一个新的带有注意机制的深度神经网络模型，名叫“隐式注意模型（Latent Attention）”。在隐式注意模型结构中，对于自然语言描述中的每个单词，模型都会学出一个权重值，来表示这一单词对于触发函数和动作函数预测的相关性。不过，与标准的注意模型不同，在这一模型中，每个单词的权重都分为两步计算，这一过程旨在更好地运用自然语言的结构来分析出一段描述中对于最终程序的合成最关键的子片段。在实验中，我将隐式注意模型运用于双向长短时记忆循环神经网络 (Bi-directional LSTM) [7] 的输出序列上；实验结果表明，我所设计的新的模型在 If-Then 程序的函数预测任务上将已有工作的最高预测准确率从 82.5% [3] 提高到了 87.5%。也就是说，相比其它已有的工作，这一新的模型在准确率上提高了 5 个百分点，在错误率上相对于已有工作的错误率降低了 28.57%。

在这篇毕业论文中，相比于已有的其它工作，我除了提出自己新的模型与方法之外，对于自然语言描述到 If-Then 程序的翻译这一任务本身，我也将更加全面地进行探索。首先，当前已有的其它工作都只研究了 Chris Quirk 等人在 [1] 提出的任务，也就是说，将模型在大规模数据集上进行训练后，再在额外的数据上进行测试。在这篇毕业论文中，除了这一已有的任务以外，我还考虑了一个新的训练场景，叫做“单样例学习 (One-shot Learning)”。之前已有一些工作在图像分类任务上考虑这样一个训练数据较少的场景 [13–15]。在我看来，这样的训练场景在本毕业论文研究的问题中也是一个需要考虑的场景。在我们考虑的 If-Then 程序中，无论是程序中的触发函数还是动作函数，基本上都对应于一些网站提供的服务或者应用程序的名称。而在当今这个时代，每天都会涌现出大量创业公司为用户提供各种各样的服务，所以在这个任务中，函数类别的总个数其实是动态增长的。因此，当 IFTTT 或 Zapier 等网站刚开始对一个新的应用程序提供支持时，包含这一应用程序的调用的 If-Then 程序必然会比较少。由于基于统计学和机器学习的模型通常需要大量的数据进行训练，在这样的场景

下，模型通常预测的准确率较低。基于这样的现状，在这篇毕业论文中，我提出了一个新的模型训练方法，旨在提高模型在训练数据缺乏的情况下的预测准确率。在实验中，我用已有的数据集来模拟这样的单样例学习场景；实验结果表明，运用我所设计的训练方法来训练隐式注意模型之后，该模型比传统方法训练出来的模型在这样的场景下预测准确率能有明显的提升。

除了增加任务类型的广度以外，在这篇毕业论文里，我也在更多的数据集上进行了测试，旨在更加全面地评估模型的通用性，同时对这一课题进行更多地探索。当前已有的从自然语言描述到 If-Then 程序的翻译的工作全部都只在 IFTTT.com 网站所提供的公开数据集上对模型进行训练与测试；然而，正如我在前文所提到的，除了这一网站以外，还有其它网站为用户提供编写并执行 If-Then 程序的服务。在这篇论文中，我除了在 IFTTT 数据集上对模型进行训练与测试以外，也在 Zapier.com 提供的公开数据集上进行了一系列实验。由于这篇毕业论文是目前为止唯一一份在这一数据集上提供实验结果的工作，因此在实验中，我只用我所设计的新的模型、以及我所实现的一些已有的经典模型进行测试。实验结果表明，在该数据集上，我所设计的新的模型也取得了比已有其它模型更好的效果。

这篇毕业论文接下来的文章结构如下：在第二章里，我将对 If-Then 程序合成这一任务进行具体的阐释，并对相关的数据集进行大体上的介绍。在第三章里，我将对已有相关工作进行概括。在第四章里，我将详细介绍我所设计的新的模型的结构、训练方法与实现细节。为了更全面地对比不同的模型，我实现了标准的注意模型作为基准模型进行对比实验，并在第五章里详细介绍了该模型的结构与实现细节。在第六章里，我将展示各个模型在 If-Then 程序合成的标准任务上的实验结果。在第七章里，我将针对从自然语言描述到 If-Then 程序的翻译这一课题，提出一个新的场景，叫做单样例学习 (One-shot learning)，并展示模型在该场景下的实验结果。在第八章里，我将对我所提出的隐式注意模型的实验结果进行进一步的分析。在结语中，我将简要概括这篇毕业论文的成果，并对该课题未来的发展给出自己的思考与展望。

1.1 本章小结

在这一章，我对这篇毕业论文的研究课题与成果做了综述。总而言之，这篇毕业论文完成了以下几项工作：

1. 提出了一个新的深度神经网络模型，名叫隐式注意模型。在 IFTTT 数据集上，这一新的模型将已有工作的最高预测准确率从 82.5% 提高到了 87.5%；也就是说，相比其它已有的工作，这一新的模型在准确率上提高了 5 个百分点，在错误率上相对于已有工作的错误率降低了 28.57%。

2. 针对从自然语言描述到 If-Then 程序的翻译这一课题，提出了一个新的训练场景，名叫单样例学习场景（One-shot Learning）。针对这一训练数据缺乏的场景，我提出了一个新的模型训练方法。运用这一训练方法之后，相比于运用传统方法训练出来的模型，在这一场景下的预测准确率能够取得明显的提升。

3. 针对从自然语言描述到 If-Then 程序的翻译这一课题，在更多的数据集上进行评估，从而对于这一课题进行更加全面的探索。

第二章　If-Then 程序的合成

在这一章里，我将对 If-Then 程序合成的相关任务进行具体的阐释，并对相关的数据集进行大体上的介绍。

2.1　If-Then 程序

在这篇毕业论文中，我集中研究一种形式较为简单、但重要的程序，叫做 If-Then（如果-则）程序。If-Then 程序，顾名思义，就是由 If-Then 语句构成的程序。每一个 If-Then 语句由两部分组成。第一部分、也就是语句中的 If 部分中，包含了一个触发函数（trigger function）；而第二部分、也就是语句中的 Then 部分中，则包含了一个动作函数（action function）。这样一个 If-Then 语句的意思是当触发函数所指定的条件满足时，便执行动作函数中的内容。

If-Then 程序可以方便地让人们根据自己的生活习惯个性化定制一些执行任务，比如进行气象监控、文件整理、日程安排等等。由于 If-Then 程序形式较为简单，因此对于尚未熟练掌握编程技能的人而言，相比于用更加复杂的编程语言来描述自己想要完成的事情，编写 If-Then 程序是一种非常便捷的任务描述方式。因此，当前已有一些网站允许用户通过编写这样的 If-Then 程序来更加便利地规划自己的生活，这些网站广受用户欢迎，并且拥有数以万计的由用户编写的 If-Then 程序，这样的网站包括 IFTTT.com、Zapier.com、Stringify.com，等等。在下一节里，我将对这些网站进行介绍。

2.2　数据集

在这篇毕业论文中，我的实验部分运用了来自 IFTTT.com 和 Zapier.com 的公开数据集。总体来说，这两个网站对于 If-Then 程序的格式定义非常类似。每一个 If-Then 语句由一个触发频道 (trigger channel)、一个触发函数 (trigger function)、一个动作频道 (action channel)、以及一个动作函数 (action function) 组成，其中一些函数可能会带有参数。在 If-Then 程序中，触发频道、触发函数、动作频道和动作函数的值都是从网站预先定义的列表中选取；而对于函数的参数而言，有些参数的值只能从一个预定义的列表中选取，有些参数的值则是允许用户进行个性化定制。这两个网站都支持上百种不同的触发频道和动作频道，这

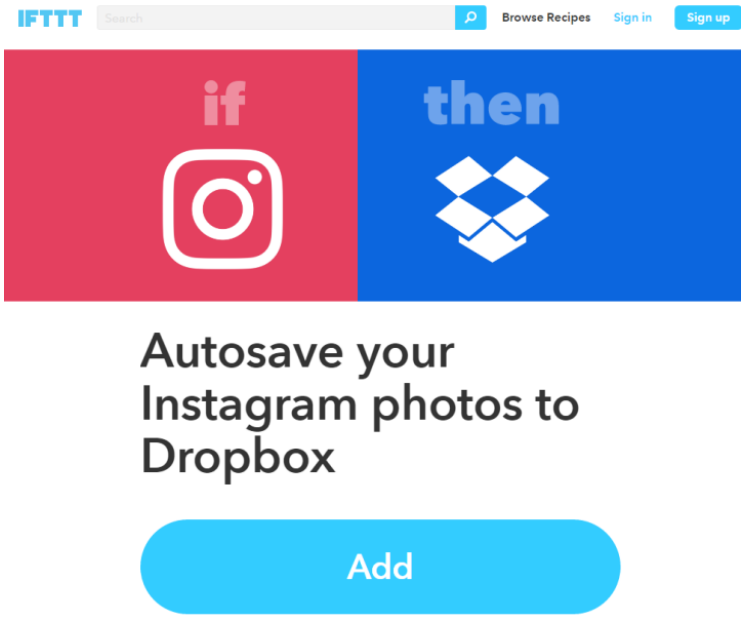

图 2–1 IFTTT.com 的 If-Then 程序样例

些频道基本上都是一些网站提供的服务或者应用程序的名称。在每个频道中还包含了一系列不同的函数，用于更加细化地解释同一种服务或应用所支持的不同功能。在这两个网站里，每一个 If-Then 程序有且仅有包含一个 If-Then 语句。在实际应用中，这样的简化形式足以表达用户的需求，同时也使得程序自动生成这样的任务不会过于复杂。另外，很多用户在分享他们所编写的 If-Then 程序的同时，也为他们所编写的程序附上一条简短的自然语言描述，用以解释这些程序的功能。接下来，我将首先对 IFTTT 和 Zapier 两个网站分别进行详细的介绍，再简要介绍一下 Stringify 网站。

2.2.1 IFTTT 数据集

图 2–1展示了一个 IFTTT 网站的 If-Then 程序样例。这个程序拥有如下自然语言描述

Autosave your Instagram photos to Dropbox

这句话的意思是当用户指定的某一 Instagram 账户中存储的图片有所更改时，便相应地同步修改用户指定的 Dropbox 账户中存储的图片信息。这个 If-Then 程序的触发频道是 Instagram, 触发函数是 Any_new_photo_by_you，动作频道是 Dropbox，动作函数是 Add_file_from_URL。其中，函数 Add_file_from_URL

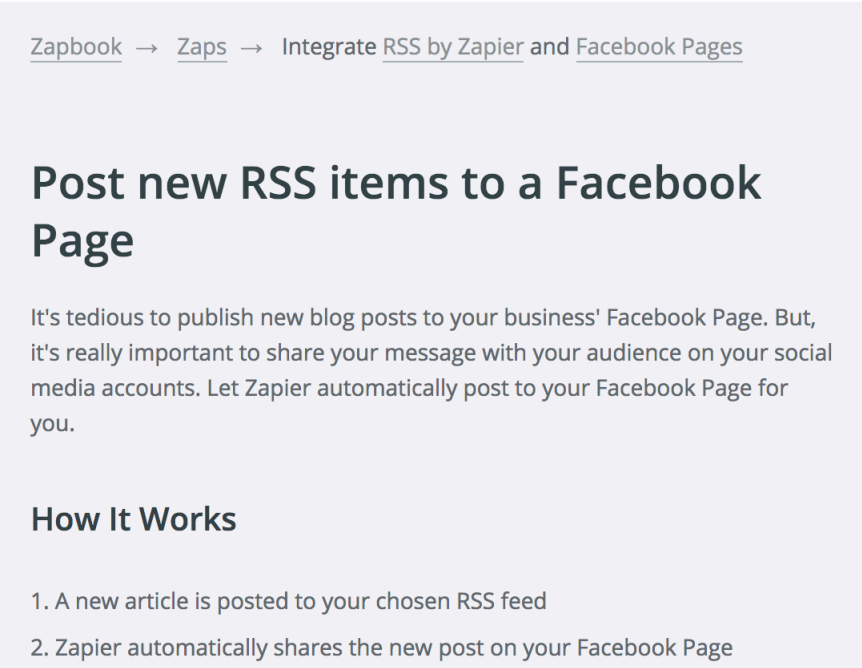

图 2–2 Zapier.com 的 If-Then 程序样例

有三个参数：源文件的 URL 链接、被保存文件的名称、以及目标文件夹的名称。

总体来说，IFTTT 数据集相比于其它用于研究语义解析的数据集而言，是一个挑战性较大的数据集。由于所有的 If-Then 程序及其相应的自然语言描述均是由用户自己编写的，一方面，数据集各个类别的样本个数分布并不均衡，每个类别的样本个数很大程度上取决于该类别所对应的应用程序在用户群中的受欢迎程度；另一方面，用户书写自然语言描述大多比较随意，不仅语法上变化多样，同时句子中会有各种个性化简写和笔误。因此，对于自然语言描述到 If-Then 程序的翻译这一任务而言，这个数据集极具挑战性。

2.2.2 Zapier 数据集

图 2–2展示了一个 Zapier 网站的 If-Then 程序样例。这个程序拥有如下自然语言描述：

Post new RSS items to a Facebook Page

这句话的意思是当用户订阅的 RSS 有新的条目时，便将这些条目同步到用户指定的 Facebook 页面上。这个 If-Then 程序的触发频道是 RSSAPI，触发函数是 new_feed，动作频道是 FacebookV2API，动作函数是 page_stream。

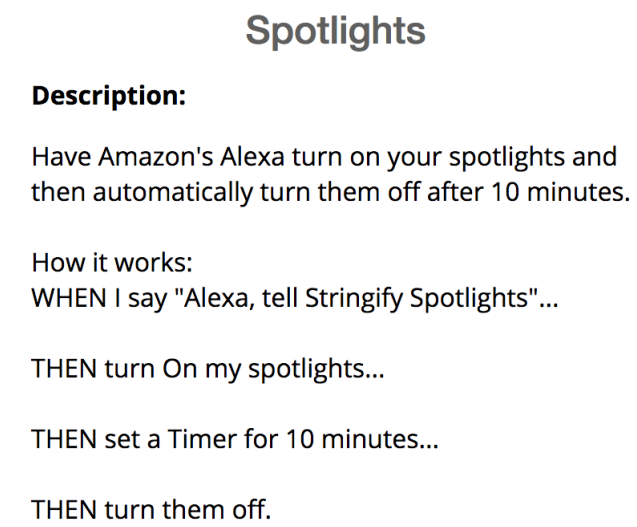

图 2–3 Stringify.com 的 If-Then 程序样例

尽管 Zapier 数据集中的所有 If-Then 程序及其对应的自然语言描述仍然是由用户撰写的，但总体来说这一数据集相比于 IFTTT 数据集而言难度有所降低，因为 Zapier 网站对于公开程序的自然语言描述的语法与用词都有严格的规定，同时每个程序都要经过人工审核才能公开，不像 IFTTT 网站是自动公开的 [1]。因此，IFTTT 数据集和 Zapier 数据集在 If-Then 程序中的关系，可以类比于 ImageNet 数据集 [16] 和 CIFAR-10 数据集 [17] 在图像分类任务中的关系。运用这两个不同的数据集进行实验，能够更加全面地对从自然语言描述到 If-Then 程序的翻译这一任务进行探究。

2.2.3 Stringify 数据集

图 2–3展示了一个 Stringify 网站的 If-Then 程序样例。相比于 IFTTT 和 Zapier 网站，Stringify 的 If-Then 程序复杂很多。首先，在这一网站中，每个 If-Then 程序可能包含多于一个 If-Then 关系；其次，生成的 If-Then 程序中每个子句并不是由一个网站定义好的频道或函数集合中选取，而是由用户个性化定义。由于当前 Stringify 网站并没有大规模的公开数据，因此在这篇毕业论文，我没有运用这一网站的数据进行实验。不过在论文正文的最后我将提到，让模型学会生成这样形式较为复杂的程序，是一个未来工作中值得探索的方向。

[1] Zapier 网站对于公开数据的书写规范参见此处：https://zapier.com/developer/documentation/v2/shared-zaps/#shared-zaps-title

2.3 If-Then 程序的合成

虽然 If-Then 程序相对来说形式已经较为简单，然而正如我在前文所说，更加友好的程序编写方式应该是用户只需要提供自然语言描述，相应的 If-Then 程序便可以被自动编写出来。近年来，有不少工作对从自然语言描述到 If-Then 程序的翻译这一课题进行研究 [1–6]。Chris Quirk 等人的工作 [1] 第一次针对这一课题提出了一系列任务，并且之前已有的所有相关工作探究的均是这份工作提出的任务。

在这篇毕业论文中，首先，我对已有工作提出的任务 [1] 进行了探究。具体来说，在这篇毕业论文中，我主要集中在对于 If-Then 程序的频道与函数的预测上。一个 If-Then 程序中的部分函数可能会带有额外的参数，但是在分析数据时我发现，其实基本上对于所有样本，如果只看自然语言描述本身，是无法完全准确地预测出所有函数参数的，因此之前所有的工作都没法对任何一个样本的所有函数参数进行完全正确的预测。同时，就像 I. Beltagy 等人在 [3] 中所述，对于 If-Then 程序生成而言，这些函数的参数并不是其中的关键部分。因此，我将 If-Then 程序生成这一任务转化为了若干个分类问题，即分别预测触发频道、触发函数、动作频道和动作函数的类别。而对于函数参数的预测，我将使用一个直观的基于频率的统计方法，并且在实验结果中展示这一方法的有效性。

除了已有工作提出的任务以外 [1]，我还针对 If-Then 程序的合成这一课题，提出了一个新的场景，叫作单样例学习（One-shot learning）场景。之前已有一些工作在图像分类任务上考虑这样一个训练数据较少的场景 [13–15]。在我看来，这样的训练场景在本毕业论文研究的问题中也是一个需要考虑的场景。在我们考虑的 If-Then 程序中，无论是程序中的触发函数还是动作函数，基本上都对应于一些网站提供的服务或者应用程序的名称。而在当今这个时代，每天都会涌现出大量创业公司为用户提供各种各样的服务，所以在这个任务中，函数类别的总个数其实是动态增长的。那么，当 IFTTT 或 Zapier 等网站刚开始对一个新的应用程序提供支持时，包含这一应用程序的调用的 If-Then 程序必然会比较少。由于基于统计学和机器学习的模型通常需要大量的数据进行训练，在这样的场景下，模型通常预测的准确率较低。因此，在这篇毕业论文中，我提出了一个新的模型训练方法，旨在提高模型在训练数据缺乏的情况下的预测准确率。

2.4 本章小结

在这一章里，首先，在第一节中，我对这篇毕业论文考虑的 If-Then 程序的形式进行了详细的说明。接下来，在第二节中，我分别介绍了在这篇毕业论文的实验部分运用到的 IFTTT 和 Zapier 数据集，同时对相关的 Stringify 数据集也进行了大致的介绍。在最后一节中，首先，我详细介绍了从自然语言描述到 If-Then 程序这一任务的提出意义与任务描述。接下来，我提出了一个与之相关的一个变种的任务，并介绍了该任务提出的背景、以及提出该任务的目的与期望。

第三章　相关工作

近年来，自动程序生成领域引发了越来越多的关注。一些已有工作对生成应用于特定领域 (domain-specific) 的程序进行了研究，比如生成正则表达式 [18]、用来解析文件的代码 [19]、数据库询问 [20, 21]、控制机器人行为的指令 [22]、运行操作系统的指令 [23]、自动操作智能手机的指令 [24]、分析表格数据的程序 [25]，等等。

近来的一些工作考虑同时运用自然语言描述和对于程序结构的指定 (specification) 这两个信息来生成编程语言代码 [26]。其中，大部分这方面的工作都依赖于语义解析技术的应用 [1, 27–29]。

与这篇毕业论文最相关的，便是一些运用 IFTTT 数据集研究从自然语言描述到 If-Then 程序翻译这一任务的工作 [1–6]。其中，Chris Quirk 等人在 [1] 中第一次引入了将 IFTTT.com 上的自然语言描述转化为 If-Then 程序这一任务。已有的其它工作主要运用三类方法来解决这一问题。第一类方法主要运用了统计学、或者传统机器学习的模型来完成这一任务 [1]。虽然这类方法从一定程度上能够成功地预测出一些较为常见的程序，然而总体来说效果仍有极大的改进空间。近年来，深度神经网络被运用于各种不同的领域，并在很多任务上取得了比传统方法更好的效果。特别地，循环神经网络 [7, 8] 以及基于循环神经网络的序列到序列翻译模型 [9] 在诸多应用上取得了引人注目的效果，比如机器翻译 [9]、自动图像说明的生成 [10]、句法解析 [11]、以及智能答疑 [12]。基于这类模型在其它应用上的优秀性能，第二类方法便是运用基于经典序列到序列翻译模型的改良模型进行预测 [4–6]。然而，I. Beltagy 等人在 [3] 中指出，对于自然语言描述到 If-Then 程序这一任务而言，由于自然语言词汇量的庞大和句法结构的复杂，基于循环神经网络的模型很难对这样的任务进行有效的学习。因此，他们提出了第三类方法，便是同时运用传统机器学习模型与深度神经网络模型进行预测 [3]。具体来说，他们运用一个全连接神经网络和一个逻辑回归的组合模型进行预测，并且这一方法在针对 If-Then 程序的函数预测这一任务上取得了比第二类方法更好的效果。在这篇毕业论文中，我基于循环神经网络结构和传统的注意机制，提出一个新的注意模型，并且在实验中展示了比已有其它工作

更高的预测准确率。

3.1 本章小结

在这一章里，我对这篇毕业论文的相关工作进行了综述，并介绍了已有其它工作与这篇毕业论文的课题的联系。

第四章　隐式注意模型 (Latent Attention)

在这一章里，我将详细介绍我所设计的隐式注意模型。

4.1　模型设计动机

为了将一个自然语言描述正确地翻译为相应的 If-Then 程序，首先，我们需要找出文本中与需要预测的类别，也就是触发频道、触发函数、动作频道或动作函数最相关的部分。就以之前提到过的这句描述为例：

Autosave your Instagram photos to Dropbox

在这个句子中，对于触发频道与函数的预测来说，蓝色文本的内容 “Instagram photos” 是最为相关的；而对于动作频道和动作函数来说，红色文本的内容 “Dropbox” 是最为相关的。

为了学习这样的对应关系，我们可以采用已有的注意机制 [9, 12]。在一个注意机制中，首先，我们需要对于输入文本的每个单词计算一个权重，表示这一单词对于最终预测的相关性与重要性；然后，我们根据这些权重，将每个输入单词对应的 embedding vector 进行加权求和，作为注意机制的结果。然而，事实上对于每个输入单词来说，它的权重值不仅取决于这个单词本身、或者整个句子包含了哪些单词，还取决于整个句子的总体结构。比如说，在下面这个例子，

Post photos in your Dropbox folder to Instagram

我们可以很明显地发现，“Dropbox” 与触发频道和函数的预测有关，而 “Instagram” 与动作频道和函数的预测相关。然而，对比之前的那句描述我们会发现，这两句话包含的单词集几乎一样，然而却给出了几乎完全相反的预测。从而我们会发现，在我们这个任务中，句子中单词出现的顺序对于预测有着重要的影响。再比如说，在下面这个例子：

Autosave your Instagram photos from Dropbox

类似地，这句描述与第一句描述中出现的单词集几乎一样，甚至连所有关键词出现的位置都是一样的，然而同样很明显地，对于这句描述的预测也与第一句描述完全相反。

我用之前已有的基准模型进行实验时，发现已有的模型中，确实有一部分能够准确地抓住文本描述中与预测相关的关键词，比如前面例子中的 “Instagram” 和 “Dropbox”。然而随后我发现，仅仅抓住这些关键词是不够的，因为在这个问题中我们其实要预测 If 和 Then 两个部分，而我们训练出来的模型也许能正确地标注出关键词，然而它们经常会混淆 If-Then 程序中 If 部分和 Then 部分的预测，也就是说将这两者的预测顺序弄反。于是我就思考，当我们人看到这样的一句自然语言描述时，是如何分析因果关系的。比如在上述三个例子中，对于前两个例子，当我们看到中间的 “to” 这个词的时候，就会推断出 “to” 之前的部分是讲述原因，而 “to” 后面的词是讲述结果; 而在第三个例子中，中间的连词相应地变成了 “from”，于是就变成了 “from” 后面的部分讲述原因，而 “from” 之前的部分讲述结果。所以如果一个模型能够学会通过这样的一些介词或连词来辅助进行逻辑分析的话，就能够对一句话有更好的理解。比如对于第一个例子，如果模型能够学会这样进行分析，便能够推断出如果要预测触发频道或触发函数，就应该将 “to” 之前的 “Instagram photos” 视为关键词；反之，如果要预测动作频道或动作函数的话，就应该视 “to” 之后的 “Dropbox” 为关键词。

基于这样的思考，在这篇毕业论文中，我设计了一个新的注意模型，名叫隐式注意模型（Latent Attention）。这个模型的设计目的，便是让模型能够模仿我们人的方式来对这种预测问题进行分析与推断。在这个模型中，首先，我先对每个单词计算一个隐式注意权重 (latent weight)，用以表示这个词对于句子中哪些位置所对应的部分与触发条件的预测有关、哪些部分与动作条件的预测有关的推断的重要程度。这一步的作用是给像 “to”，“from” 这样的介词或指示性的单词赋予更高的权重，因为这样的词对于指示句子的因果关系结构起着比较大的作用。在第二步中，我参考标准的注意机制 [9, 12] 来计算最终的注意权重 (active weight)；但与已有的注意机制不同，在这一步中，上一步计算的隐式注意权重也参与计算这一步的注意权重。比如说对于第一个例子，由于在第一步中，模型会赋予 “to” 较高的隐式注意权重，因此在这一步中，根据 “to” 这个词所出现的位置，当预测 If 部分时，模型会对 “to” 之前的单词赋予较高的注意权重；反之，当预测 Then 部分时，模型会对 “to” 之后的单词赋予较高的权重。这一步的作用是强调那些与最终预测关联性最大的词，比如在第一个例子中，预

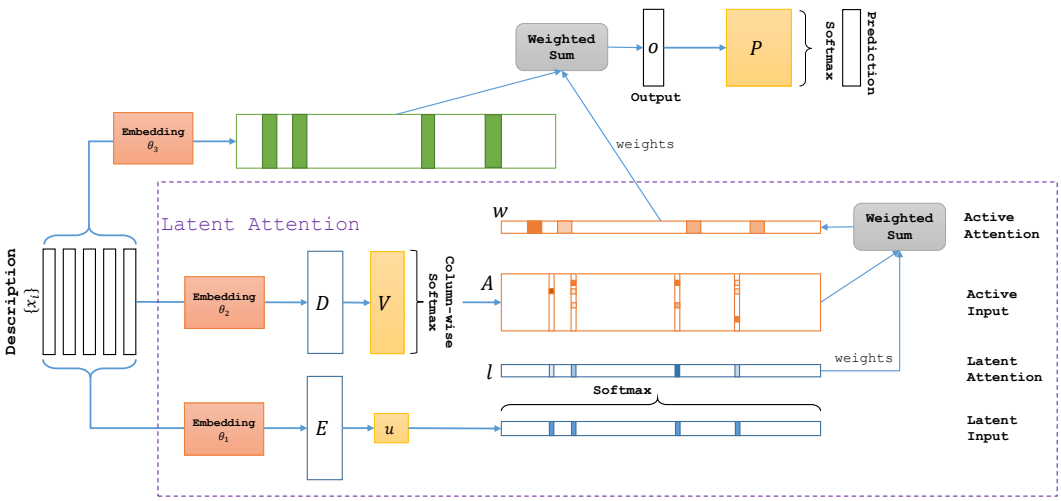

图 4–1 隐式注意模型 (Latent Attention) 的结构

测触发条件时，“Instagram photos” 将会被赋予最高的最终权重；反之，如果要预测动作频道或动作函数的话，“Dropbox” 将被赋予最高的最终权重。最后，模型将对文本中的每个词，根据最终权重的高低来综合分析，并做出预测。

在接下来几节中，我将具体介绍模型的结构、预测方式，以及模型的训练方法。

4.2 模型结构

我所设计的隐式注意模型的结构如图 4–1所示。在接下来的描述中，我会按照符号使用的惯例，运用小写字母来表示列向量，而运用大写字母来表示矩阵。

首先，模型的输入是一个单词序列 $x_1, ..., x_J$，其中每个单词都来自一个包含了 N 个不同单词的词汇表。我将这样的一个输入序列记为 $X = [x_1, ..., x_J]$。这里，J 是每段文本描述的最大长度。接下来，我将对模型的每一层进行介绍。

4.2.1 隐式注意层 (Latent attention layer)

首先，我将输入序列中的每个单词 x_i 编码为一个 N 维的独热向量 (One-hot vector)。具体来说，假如 x_i 是词汇表中的第 n 个词，则它所对应的独热向量的第 n 维的值是 1，其余维度的值为 0。然后，我通过以下公式，将整个输入序列 X 转化为一个 d 维的 embedding 序列：

$$E = \texttt{Embed}_{\theta_1}(X) \tag{4–1}$$

在这个公式中，θ_1 是模型的一组需要进行训练的参数，E 的大小为 $d \times J$。在这一步中，我考虑了不同的 embedding 方法，我将在 4.3这一节对这些方法进行具体的介绍。

隐式注意层的输出是输入序列中每个词的隐式注意权重。对于该权重的计算，我运用了一个标准的 softmax，并将 E 作为 softmax 的输入。具体来说，令 l 是一个 J 维的向量，用以表示隐式注意层的输出；u 是一个 d 维的向量，该向量是模型中需要训练的参数的一部分，则

$$l = \textbf{softmax}(u^T \texttt{Embed}_{\theta_1}(X)) \tag{4–2}$$

4.2.2 最终注意层 (Active attention layer)

在最终注意层的输出中，输入序列中的每个词将被赋予一个最终注意权重，用以表示这个词对于最终预测的相关性。首先，我运用和前一小节同样的方法，将输入序列 X 转化为一个 d 维的 embedding 序列 D；不同的是，在这里，我使用模型中另一组需要进行训练的参数 θ_2, 即

$$D = \texttt{Embed}_{\theta_2}(X) \tag{4–3}$$

与 E 相同，这里 D 的大小也是 $d \times J$。

接下来，对 D 中的每个 d 维向量 D_i，我运用一个 softmax 来计算它所对应的最终注意层的的输入 A_i，具体公式如下:

$$A_i = \textbf{softmax}(VD_i) \tag{4–4}$$

这里, A 的大小是 $J \times J$，A_i 是 A 的第 i 维列向量，V 是模型中一个需要训练的参数矩阵，它的大小为 $J \times d$。注意到 $VD_i = (VD)_i$, 因此，我们可以通过逐列计算 VD 的 softmax 输出值，来对 A 进行计算。

对于最终注意权重的计算，则是运用隐式注意层的输出作为权值，对每个 A_i 进行带权求和，具体公式如下:

$$w = \sum_{i=1}^{J} l_i A_i = Al \tag{4–5}$$

4.2.3 输出表示

在这一部分，首先，我运用与前文所述类似的 embedding 方法，运用第三组需要进行训练的参数 θ_3，将输入序列 X 转化为一个 $d \times J$ 的 embedding 矩阵。对于模型的输出表示 o 的计算，则是运用最终注意层的输出作为权重，对这一 embedding 矩阵进行带权求和，具体公式如下：

$$o = \texttt{Embed}_{\theta_3}(X)w \tag{4–6}$$

4.2.4 最终预测

对于最终生成的 If-Then 程序，模型将分别对以下部分进行预测：（1）触发条件，即触发频道和触发函数；（2）动作条件，即动作频道和动作函数；（3）触发函数和动作函数所对应的参数。接下来，我将分别对这三部分的预测方法进行介绍。

4.2.4.1 对于触发条件和动作条件的预测

对于触发条件和动作条件，在这篇毕业论文中，我考虑了如下两种预测方法：

4.2.4.1.1 独立预测 在独立预测中，我将对于触发条件的预测和对于动作条件的预测看成两个互相独立的任务，运用两个结构相同、但是参数不同的模型分别进行预测。对于每个条件的推断，我将其转化为一个经典的分类问题，并运用一个 softmax 来做出最终预测，具体公式如下：

$$\hat{f} = \textbf{softmax}(Po) \tag{4–7}$$

在这一公式中，P 是模型中一个需要训练的参数矩阵，它的大小为 $d \times M$，这里 M 是不同类别的触发或动作条件的种数。

4.2.4.1.2 联合预测 事实上，在这个任务中，触发条件和动作条件的预测虽然是两个不同的问题，但它们都属于 If-Then 程序的一部分；并且，它们之间其实

是具有相关性的，因为与触发条件的预测相关的文本和与动作条件的预测相关的文本存在一定的互补关系。因此，在联合预测中，我将使用同一个模型，对触发条件和动作条件进行同时预测。

具体来说，对于整个模型从隐式注意层到输出表示这一部分，都与独立预测所使用的模型结构一致。不同的是，在最终预测部分，我将输出表示 o 同时作为两个 softmax 的输入，并将这两个 softmax 的输出分别作为触发条件和动作条件的预测，相关公式如下：

$$\hat{f}_t = \textbf{softmax}(P_t o) \tag{4–8}$$

$$\hat{f}_a = \textbf{softmax}(P_a o) \tag{4–9}$$

在这两个公式中，$\hat{f}_t$ 和 $\hat{f}_a$ 分别对应于触发条件和动作条件的预测结果，P_t 和 P_a 是模型中两个需要训练的参数矩阵，它们的大小分别为 $d \times M_t$ 和 $d \times M_a$，这里 M_t 是不同类别的触发条件的种数，M_a 是不同类别的动作条件的种数。

4.2.4.2 对于函数参数的预测

对于函数参数的预测，我们不能把它直接看作一个分类问题，因为在前文中我提到，有些函数的参数是允许用户进行个性化定制的。因此，已有的其它工作在处理这一任务时，均采用生成模型 [1, 4, 5] 进行预测。然而在分析数据时我发现，其实基本上对于所有样本，如果只看自然语言描述本身，是无法完全准确地预测出所有参数的，因此之前所有的工作都没法对任何一个样本的所有参数进行完全正确的预测。基于这一点，我使用了一个非常直观的方法来进行参数的预测，这是一个基于频率统计的方法。具体来说，首先，对于训练数据集中的每个触发函数或动作函数 f，记 $ARG(f)$ 为该函数所有不同参数类型的集合，我考虑每个参数类型 $arg \in ARG(f)$，并对于该参数在训练数据集中出现过的所有不同参数值 v，统计其出现的频率。我将该频率记为 $\boldsymbol{Pr}(v|f, arg)$。接下来，对于每个文本描述，我先运用前文所述方法预测出相应的 If-Then 程序中的触发函数 f_t 和动作函数 f_a，然后对于这两个函数中的每个参数的值，我们进行如下预测：

$$\text{argmax}_{v_t} \boldsymbol{Pr}(v_t|f_t, arg_t), \forall arg_t \in ARG(f_t) \tag{4–10}$$

$$\text{argmax}_{v_a} Pr(v_a | f_a, arg_a), \forall arg_a \in ARG(f_a) \tag{4–11}$$

注意到在这里，对于不同的样本，只要它们对于函数的预测结果是一样的，那么它们对于该函数对应的参数的预测结果也都是一样的。

在实验部分，我将展示这一方法的测试结果。从实验结果中我们将看到，当把这一方法与隐式注意模型的函数预测结果结合到一起时，这个简单直观的方法已经足以取得比所有已有其它工作都更高的预测准确率。

4.3 实现细节

在这一节里，我将对隐式注意模型的实现细节进行阐释。

4.3.1 Embedding 方法

在这篇毕业论文中，我考虑两种 embedding 方法，从而将输入单词在向量空间中进行表示。在接下来的两小节里，我将分别介绍这两种方法。

4.3.1.1 Dictionary Embedding

我考虑的第一种 embedding 方法是一种直接进行编码的方法，我将其命名为 dictionary embedding。这种 embedding 方法的具体公式如下：

$$\texttt{Embed}_\theta(X) = \theta X \tag{4–12}$$

在这个公式中，θ 是模型中的一个需要训练的参数矩阵，它的大小是 $d \times N$；X 中的每一个行向量 X_i 是输入序列中第 i 个词所对应的独热向量表示，具体细节参照 4.2这一节。

对于 θ 这样的 embedding 矩阵，当前已有一些公开的预训练参数，比如 Word2Vec [30–32] 和 GloVe [33]。然而我发现，这些预训练参数并不适用于这篇毕业论文中所考虑的课题。首先，在这个任务中，大部分重要的词汇都是各种在线服务和应用程序的名称，而这些词汇大多不包含在这些预训练的 embedding 模型中。其次，在这个任务中，数据集的文本描述中拥有大量的用户个性化简写和拼写错误，而由于已有预训练 embedding 模型大多是在规范化数据集上进行训练，因此它们并不能很好地处理这种多样化的拼写。因此，在我的实现中，θ 并没有使用预训练模型，也没有在其它数据集、或者运用额外的目标函

数进行预训练，而是与模型中的其它参数同时进行随机初始化和训练。从实验结果中我发现，当使用隐式注意模型进行预测时，这种简单直接的 embedding 方法已经可以取得比已有其它的大部分工作更好的效果 [1, 4–6]。

4.3.1.2 Bi-directional LSTM Embedding

由于循环神经网络在处理序列数据上具有优秀的性能，这类模型在与自然语言处理相关的应用领域被广泛使用。其中，长短时记忆循环神经网络（Long Short-Term Memory，简写为 LSTM）是经典循环神经网络的一个变种，这种网络相比于经典循环神经网络而言能够更好地学习一个句子中各个部分与和该部分相隔距离较远的部分之间的依赖关系 (long-term dependency)。因此在第二种方法中，我运用了一个双向长短时记忆循环神经网络 (Bi-directional LSTM) [7] 来进行 embedding 操作，我将该方法命名为 Bi-directional LSTM Embedding。概括地说，这种 embedding 方法是在 dictionary embedding 的基础上，首先将 directionary embedding 的输出结果作为一个 Bi-directional LSTM 的输入，这样在每个时刻，Bi-directional LSTM 中的正向与反向 LSTM 均会给出一个输出。于是接下来，对于输入序列的第 i 个单词，我将第 i 时刻的两个 LSTMs 的输出进行连接，作为其 embedding 结果。运用这种 embedding 方法，在每个单词的 embedding 结果中，都会考虑该单词的上下文信息，因此这样的 embedding 结果相比于 dictionary embedding 的结果来说，能够包含更多信息。接下来，我将具体地介绍这一方法的相关实现。

给定输入序列 $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_J \in \mathbb{R}^n$，一个循环神经网络计算

$$\mathbf{h}_t = \tanh(\mathbf{W}_{xh}\mathbf{x}_t + \mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{b}_h) \tag{4–13}$$

其中 $\mathbf{h}_0$ 是一个零向量；$\mathbf{W}_{xh}$ 和 $\mathbf{W}_{hh}$ 是两个需要进行训练的参数矩阵，它们的大小分别是 $m \times n$ 和 $n \times n$；$\mathbf{b}_h \in \mathbf{R}^m$ 是一个偏差向量 (bias).

对于 LSTM 的结构，我运用了 Wojciech Zaremba 等人在 [7] 中提出的版本，具体细节如下：

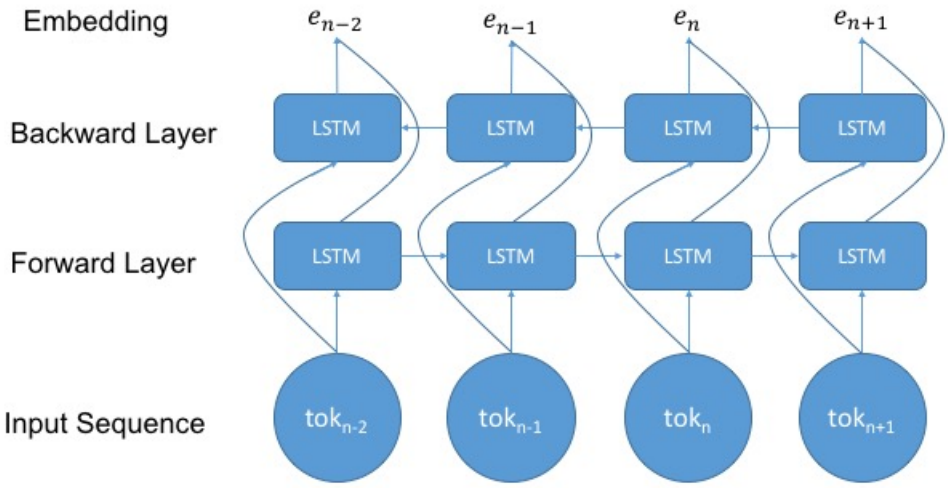

图 4–2 Bi-directional LSTM Embedding 方法

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} \mathbf{T}_{2n,4n} \begin{pmatrix} \mathbf{x}_t \\ \mathbf{h}_{t-1} \end{pmatrix} \tag{4–14}$$

$$\mathbf{c}_t = f \odot \mathbf{c}_{t-1} + i \odot g \tag{4–15}$$

$$\mathbf{h}_t = o \odot \tanh(c_t) \tag{4–16}$$

在上述公式中, σ 表示 sigmoid 函数,$\odot$ 表示逐项相乘。相比于经典循环神经网络而言，LSTM 在保留了经典神经网络的隐状态 (hidden states) 结构以外，还额外引入了记忆细胞 (memory cells) $\mathbf{c}_t$，这些记忆细胞比起隐状态而言能够存储与文本当前部分相隔更远的部分的上下文信息，从而 LSTM 相比于经典循环神经网络而言，能够更好地学习一个句子中各个部分与和该部分相隔距离较远的部分之间的依赖关系 (long-term dependency)。

我所构建的 Bi-directional LSTM 的模型结构如图 4–2所示。具体来说，在我

所构建的 Bi-directional LSTM 中，正向 LSTM 模型将正向的输入序列作为其输入，另一个反向 LSTM 模型则将反向排列的输入序列作为其输入。对于输入序列的第 i 个单词，我将第 i 时刻的两个 LSTMs 的输出进行连接，作为其 embedding 结果。

在实验中我发现，运用这种 embedding 方法，我所设计的隐式注意模型可以取得比已有其它工作更高的预测准确率。

在整个隐式注意模型中，有三组需要训练的 embedding 参数，即 $\theta_1, \theta_2, \theta_3$。对于已有其它相关工作所考虑的任务来说 [1] 来说，我发现将这三组参数的值设为相同之后，对于测试结果而言并无任何影响。然而接下来在我所提出的新的单样例学习场景中，我们会发现，将这三组 embedding 参数分开进行训练，对提高模型在训练数据较少的情况下的测试准确率大有帮助。

4.3.2 归一化最终注意参数

在实验中我发现，在计算输出表示 o 之前，将最终注意参数 w 进行归一化操作可以有效地提高测试准确率。具体来说，我对前文中计算输出表示 o 的公式进行如下修改：

$$o = \texttt{Embed}_{\theta_3}(X)\textbf{normalized}(w) = \texttt{Embed}_{\theta_3}(X)\frac{w}{||w||} \tag{4–17}$$

在这个公式中，$||w||$ 是 w 的 L_2 范数。从实验结果来看，这种归一化操作可以使得预测准确率提高 1 至 2 个百分点。

4.3.3 对于输入序列长度的预处理

我所设计的隐式注意模型要求每个输入序列的长度都是固定的。而在实际数据集中，每个输入序列的长度大多都是不一样的。在实验中，为了处理不同长度的输入序列，我将每个输入序列进行增长或剪切。我的具体做法如下：

1. 如果一个输入序列的长度小于 J，则我在输入序列末尾用一个特殊符号 ⟨NULL⟩ 进行补全，从而使输入序列的长度为 J；

2. 反之，如果一个输入序列的长度大于 J，我则保留输入序列的前 $(J-1)/2$ 和后 $(J+1)/2$ 个单词，将中间其余部分去除。注意到在以往的工作中，大部分剪切输入序列的方法都是将文本末端超过长度限制的部分忽略。在这

里我之所以采用这样的剪切方式，是因为在分析数据时我发现，其实对于 If-Then 程序的自然语言描述来说，大部分关键信息都是集中在文本头部与尾部，因此这样的做法可以最大限度地保留输入文本的核心信息。

在实验评估中，我将 J 的值设为 25。

4.3.4 对于词汇表的预处理

对于每个输入文本，我首先根据文本中的空格和标点符号，比如 .,!?”′ :;)(，对输入文本中的每个单词进行切分 (tokenization)，然后将所有英文字母转化为小写字母。在这一预处理过程中，我将所有的标点符号也作为一个单词保留在模型的输入序列中，因为在之后的实验结果分析中我们会发现，对于这一任务来说，保留这些标点符号是有必要的。接下来，我将训练数据集中所有单词出现的次数进行了统计，并保留其中出现次数最多的 4,000 个单词。对于其余单词，我将它们统一映射为一个特殊符号 ⟨UNK⟩。因此，词汇表的大小 N 为 4,001。

在前文中我介绍到，在 IFTTT 数据集中的文本描述经常会出现拼写错误。因此，之前的一些工作借助了标准的拼写纠正工具来进行修正 [4, 34]。在我的实现中，我没有对拼写错误进行额外的处理，而是希望模型能够在训练过程中，自己学会处理这类情况。

4.4 本章小结

在这一章里，我对我所设计的新的模型进行了全面而详细的描述。首先，在第一节中，我介绍了我设计新的模型的动机。接下来，在第二节中，我详细地介绍了模型各个部分的结构，并阐释了模型如何对 If-Then 程序的各个部分进行预测。在第三节中，我具体地介绍了模型的一些实验细节。

第五章　标准注意模型

为了更全面地对比不同的模型，在这份毕业设计中，除了我所设计的新的模型以外，我还实现了标准的注意模型作为其中一类基准模型，以便之后进行对比实验。在这一章里，我将对标准注意模型的实现进行介绍。

在这一基准模型的实现中，我主要参照了 End-To-End Memory Network [12] 中注意机制的实现。标准注意模型与我所设计的隐式注意模型最大的区别在于，在标准注意模型中只有对于最终注意权重的计算，而没有隐式注意层这一结构。接下来，我将对标准注意模型不同于隐式注意模型的部分进行描述。在以下描述中，除了特殊说明以外，符号的使用和含义与前一章一致。

5.1　注意权重计算层

在标准注意模型中，注意权重 a 用如下公式进行计算：

$$a = \textbf{softmax}(u^T \texttt{Embed}_{\theta_1}(X)) \tag{5–1}$$

在这一公式里，a 的维数为 J，其中每一维的数值体现了输入序列中相应位置的单词与最终预测的相关性；u 是一个 d 维向量，是模型中一个需要进行训练的参数。

5.2　输出表示

在这里，与前一章类似，首先，我运用第二组需要训练的 embedding 参数 θ_2，将输入序列 X 转化为一个 $d \times J$ 的 embedding 矩阵。于是，模型的输出表示 o 则运用之前计算的注意权重，对这一 embedding 矩阵进行带权求和，具体公式如下：

$$o = \texttt{Embed}_{\theta_2}(X)a \tag{5–2}$$

这里，输出表示 o 是一个 d 维的向量。

5.3 最终预测

在最终预测中，对于函数参数的预测方法与前一章所述相同。对于触发条件和动作条件的预测，我采用的也是与前一章类似的方法，就是运用一个 softmax 进行分类，具体公式如下：

$$\hat{f} = \textbf{softmax}(Wo) \tag{5–3}$$

与前一章的类似，在这一公式中，W 是模型中一个需要训练的参数矩阵，它的大小为 $d \times M$，这里 M 是不同类别的触发或动作条件的种数。

5.4 本章小结

在这一章中，我对我在实验中进行对照的标准注意模型的实现进行了介绍，重点阐释了标准注意模型与我所设计的隐式注意模型的不同之处。

第六章　If-Then 程序合成任务的评估

在这一章里，我将展示在标准 If-Then 程序合成任务上的实验结果。其中，对于 IFTTT 数据集，我将展示我所设计的新的模型、我所实现的基准模型的实验结果，并与已有的其它工作进行对比 [1–6]。对于 Zapier 数据集，由于之前没有其它论文在这一数据集上进行实验，因此我将展示我所设计的新的模型和我所实现的基准模型的实验结果。

6.1　实验数据集

在这一节中，我将对 IFTTT 数据集和 Zapier 数据集分别进行介绍。

6.1.1　IFTTT 数据集

Chris Quirk 等人在 [1] 中第一次引入了 IFTTT 数据集，并同时提供了一个爬虫程序，用来从 IFTTT.com 采集数据。在这篇毕业论文中，我运用他们提供的爬虫程序来进行数据集的采集。然而，由于网站中用户编写的 If-Then 程序时时都在更新，因此部分 If-Then 程序已经失效了。于是我采集了所有仍然有效的 If-Then 程序，最终在训练数据集里包含了 68,083 个 If-Then 程序。同时，Chris Quirk 等人在 [1] 中提供了一个包含 5,171 个程序的验证集 (validation set)，以及一个包含 4,294 个程序的测试集 (test set)。直至当前为止，4,220 个来自验证集的程序仍然有效，3,868 个来自测试集的程序仍然有效。在前文中我提到，IFTTT 数据集中自然语言描述形式多样，其中很多自然语言描述即便是对于我们人来说，也很难通过描述本身来推断出对应的触发条件和动作条件。因此，Chris Quirk 等人在 [1] 中定义了一个测试集的子集，叫作 gold test set。为了构建这一测试集，他们雇用了一群来自亚马逊土耳其机器人（Amazon Mechanical Turk）的工作者，让他们根据测试集中的自然语言描述，来推断相应的 If-Then 程序。对于每一句自然语言描述，Chris Quirk 等人会请 5 个不同的人来进行推断。如果其中三个人都进行了正确的预测，说明这个自然语言描述的表意足够清晰，于是他们便将该文本描述加入到 gold test set 里；反之则说明这段描述不够清晰，则不加入到 gold test set 里。运用这个方法，他们构造的 gold test set 里包含了 758 个 If-Then 程序。直至当前为止，其中 584 个程序仍然有效。对于 IFTTT 数据集

表 6–1 IFTTT 数据集的统计信息

	训练集	测试集 (Gold)
触发频道的种数	112	59
触发函数的种数	443	136
动作频道的种数	87	41
动作函数的种数	161	56
If-Then 程序的个数	68,083	584

表 6–2 Zapier 数据集的统计信息

	训练集	测试集
触发频道的种数	326	162
触发函数的种数	570	232
动作频道的种数	292	190
动作函数的种数	392	221
If-Then 程序的个数	21,138	3,963

详细的统计信息如表 6–1所示。

6.1.2 Zapier 数据集

Zapier.com 也提供了一个公开的 If-Then 程序数据集，该数据集可以通过运行爬虫程序获得[1]。在 Zapier 数据集中，总共有 26,423 个 If-Then 程序。我将其中大致 80% 的程序作为训练集，5% 的程序作为验证集，15% 的程序作为测试集，并且保证验证集和测试集中预测的触发条件与动作条件均在训练集中出现过。Zapier 数据集详细的统计信息如表 6–2所示。

6.2 评估模型

首先，在我的实现中，对于输入序列的 embedding，我实现了前文所介绍的 dictionary embedding 和 Bi-directional LSTM embedding 两种方法。对于注意模型的选取，我实现了无注意模型、标准注意模型、以及隐式注意模型这三种不同的情况，其中标准注意模型以及隐式注意模型均已在前文介绍。因此，我总共实现了 6 种不同的模型结构。接下来，我将对无注意模型的情况进行讨论。

当模型中使用 dictionary embedding 的方法进行 embedding、同时没有使用

[1]一个爬虫程序的实现: https://github.com/miguelcb84/ewe-scrapers/blob/master/ewescrapers/spiders/zapier_spiders.py。

注意机制时，对于每个输入序列，我将其中的每个词的 embedding 向量求和，随后将其通过一个 softmax 来进行预测。对于输入序列的处理，我还尝试了对每个词的 embedding 向量求平均来作为 softmax 输入的方法。从实验情况来看，求和与求平均两种方法对实验结果没有影响，因此在接下来的实验结果中，我展示的是运用求和的方法计算出来的结果。

当模型中使用 Bi-directional LSTM embedding 的方法进行 embedding、同时没有使用注意机制时，对于每个输入序列，我把正向 LSTM 和反向 LSTM 的最终隐状态 (final hidden states) 进行连接后，将其作为 softmax 的输入进行预测。

此外，对于这 6 种模型结构，我分别尝试了将触发条件和动作条件进行独立预测和联合预测，从实验情况来看，这两种预测方法对实验结果没有影响，因此在接下来的实验结果中，我展示的是将两个部分分别进行预测的结果。

6.3 模型训练的细节

当训练没有使用注意机制的模型时，它们训练的初始学习速率 (learning rate) 是 0.01，并且每隔 1,000 个时刻会衰减为当前学习速率的 0.9。当模型参数的梯度的 L_2 范数的值大于 5.0 时，它们将被按比例缩小，直至 L_2 范数的值等于 5.0。当训练使用了标准注意机制或隐式注意机制的模型时，它们训练的初始学习速率是 0.001，并且在训练过程中这个学习速率不会衰减。当模型参数的 L_2 范数的值大于 40.0 时，它们将被按比例缩小，直至 L_2 范数的值等于 40.0。所有的模型均用 Adam 优化器进行训练 [35]。模型的所有参数均是在 $[-0.1, 0.1]$ 这一区间进行随机初始化。每一轮训练时，我先将整个训练数据集随机打乱后，再将数据输入到模型中进行训练。每个小批量数据 (mini-batch) 的大小是 32，并且 embedding 向量的维度 d 是 50。

6.4 实验结果

6.4.1 IFTTT 数据集

6.4.1.1 触发条件和动作条件的预测

频道和函数预测的准确率分别如图 6–1和 6–2所示。其中，Dict 表示 dictionary embedding，BDLSTM 表示 Bi-directional embedding，A 表示标准注意模型，LA 表示隐式注意模型，在 Dict 或 BDLSTM 之后没有加任何文字则说明没有运用注意模型。关于准确率的计算，对于频道的预测来说，只有当 If-Then 程序中

的触发频道和动作频道的预测均正确时，我们才说该程序的频道预测是正确的。对于函数的预测来说，由于每个函数都归属于一个频道，因此只有当 If-Then 程序中触发频道、动作频道、触发函数、动作函数均预测正确时，我们才说该程序的函数预测是正确的。在这两幅图中，我也展示了已有的其余 5 份工作的实验结果。其中，Chris Quirk 等人的工作是第一份引入了 If-Then 程序合成这一任务的工作 [1]。I. Beltagy 等人的工作运用一个全连接神经网络和一个逻辑回归的组合模型进行预测 [3]。其余已有的工作则是运用基于经典序列到序列翻译模型的改良模型进行预测 [4–6]。为了使结果显示更加清晰，在这篇毕业论文的正文中我只展示这些图表，具体数值参见附录 A。

对于我所实现的 6 种模型结构中的每一种结构，我都随机初始化了 10 个参数不同的模型，将这些模型分别进行训练后，同时用它们的输出结果进行最终的预测。对于每种结构，当 ensemble k 个模型的时候，首先，我选取 10 个模型中在验证集上预测准确率最高的 k 个模型，然后把它们的 softmax 输出取平均，并将其作为 ensemble 模型的最终输出。对于 5 份已有其它工作 [1, 3–6], 我选取这些论文中的最好的结果展示在这篇毕业论文中。

从实验结果中，我们可以观察到：

- 无论是用 dictionary embedding 的方法，还是用 Bi-directional LSTM embedding 的方法，隐式注意模型总是能取得比标准注意模型和没有运用注意模型的结构更好的预测效果。

- 对于我所实现的 6 种模型结构，运用多于 1 个模型进行 ensemble 的实验效果和运用单一模型相比，总是有显著的提高。

- 当运用多于 1 个模型进行 ensemble 时，Bi-directional LSTM embedding 方法能够比 dictionary embedding 更好的效果。在我看来，原因是对于每个单词，Bi-directional LSTM embedding 的结果会考虑该单词的上下文信息，比如说该单词所在的词组，因此这样的 embedding 结果相比于 dictionary embedding 来说更加有效。

- 对于图 6–1所示的频道预测任务来说，除了没有带注意机制的 dictionary embedding 以外 (图 6–1中 Dict 所对应的结果)，其它模型结构均能取得比 Chris Quirk 等人在 [1] 中展示的更好的效果。当运用带有隐式注意机制或

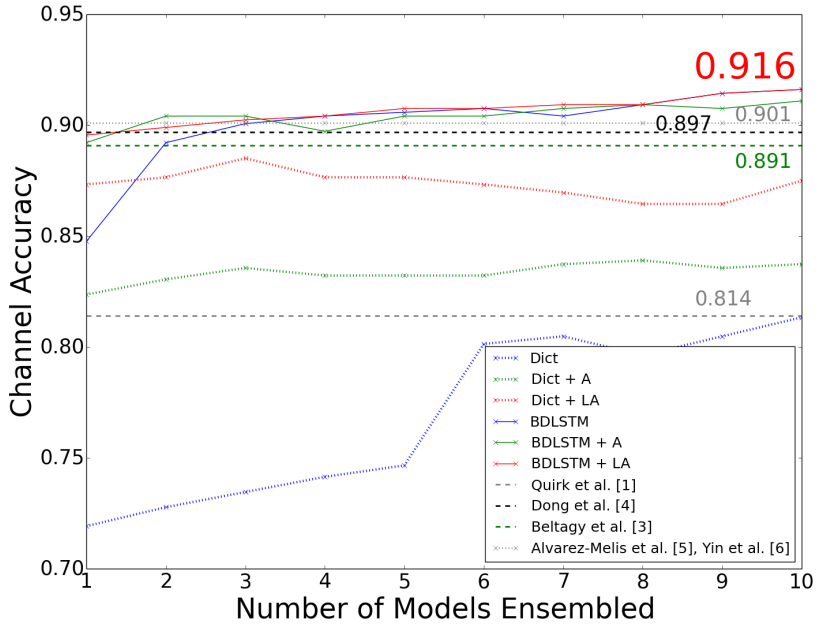

图 6–1 IFTTT 数据集上频道预测的准确率。其中，Dict 表示 dictionary embedding，BDLSTM 表示 Bi-directional LSTM embedding，A 表示标准注意模型，LA 表示隐式注意模型，在 Dict 或 BDLSTM 之后没有加任何文字则说明没有运用注意模型。

标准注意机制的 Bi-directional LSTM 时，只需要 ensemble 3 个不同的模型，我便能够取得比所有已有其它工作更高的预测准确率。 [1, 3–6]。当 ensemble 10 个带有隐式注意机制的 LSTM 时，我可以比已有其它工作取得的最高准确率提高 1.5 个百分点 [5, 6]。

- 对于图 6–2所示的函数预测任务来说，我所实现的 6 种模型结构均能取得比 Chris Quirk 等人在 [1] 中展示的更好的效果。除此之外，ensemble 9 个带有隐式注意机制的 LSTM 可以比之前最好的结果提高 5 个百分点 [3]。也就是说，我所设计的新的模型在错误率上相对于已有工作的最低错误率降低了 28.57%。

6.4.1.2 函数参数的预测

正如我在前文所述，对于这个数据集来说，基本上对于所有样本，如果只看自然语言描述本身，是无法完全准确地预测出所有参数的，因此当前所有的工作都没法对任何一个样本的所有参数进行完全正确的预测。基于这点，Chris

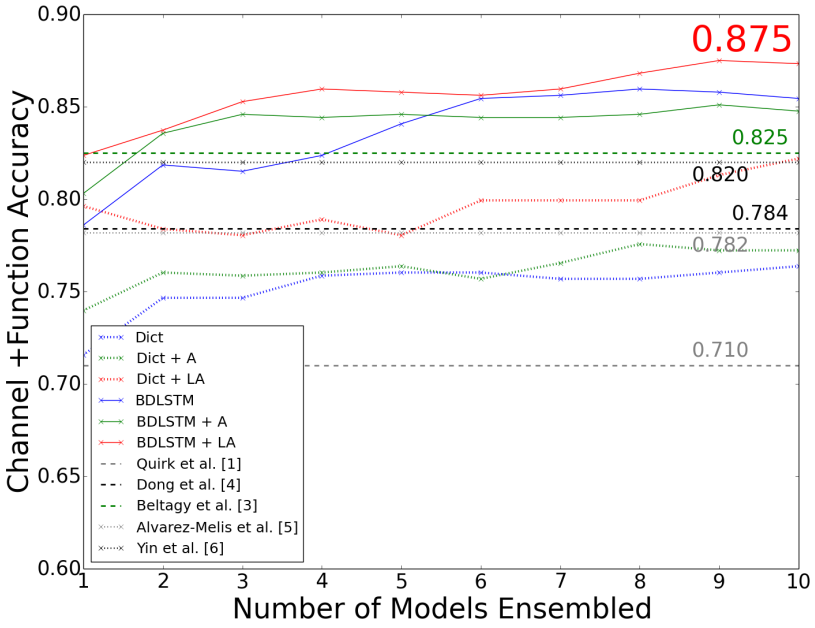

图 6–2 IFTTT 数据集上函数预测的准确率。其中，Dict 表示 dictionary embedding，BDLSTM 表示 Bi-directional LSTM embedding，A 表示标准注意模型，LA 表示隐式注意模型，在 Dict 或 BDLSTM 之后没有加任何文字则说明没有运用注意模型。

Quirk 等人在 [1] 中提出了一种评估函数参数预测效果的指标，叫作 F_1 值。具体来说，我们可以把一段自然语言描述所对应的 If-Then 程序用一个抽象语法树 (abstract syntax tree) 来表示。图 6–3展示了一个 If-Then 程序对应的抽象语法树的例子。于是，我们如果将抽象语法树中的每条边看成是语法里的一条生成规则 (production rule) 的话，我们便可以比较正确的抽象语法树与预测出来的抽象语法树的 F_1 值。

我发现对于函数参数来说，用户并不一定会对一个函数的所有参数都进行制定。对于函数参数值未被制定的情况，我给这些函数参数值赋予一个特殊的值，叫作 $\langle MISSING \rangle$；注意到，这个值与 “” 是不同的，“” 的含义是函数参数值存在、但长度为 0。

函数参数预测的实验结果如图 6–4所示。由于部分已有工作没有展示函数参数预测的结果，因此在该图中没有包含相应工作的结果。从实验结果表明，虽然我用来预测函数参数的方法本身与其它的相比较为简单直观，但仍然能取得比其它工作更好的效果。当然，一个重要的原因是我所设计的模型在频道和函

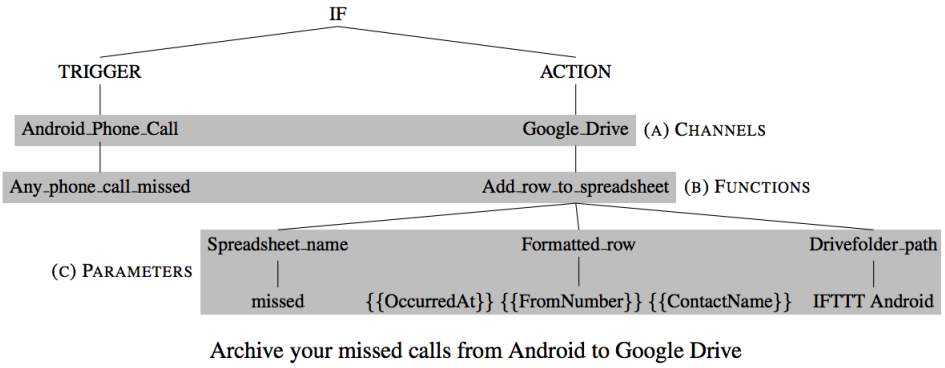

图 6–3 一个 If-Then 程序对应的抽象语法树的例子。这张图截取自 [1]。

表 6–3 Zapier 数据集上的实验结果

	频道	频道 + 函数
BDLSTM+A/LA	**97.6%**	**93.1%**
BDLSTM	97.0%	92.8%
Dict+A/LA	96.6%	92.6%
Dict	89.4%	84.3%

数上的预测准确率比其余工作都有明显的提高。

6.4.2 Zapier 数据集

由于 Zapier 数据集中，If-Then 程序的函数参数不够完善，因此对于这一数据集，我只进行触发条件和动作条件的预测。由于之前没有其它论文在这一数据集上进行实验，因此我只展示我所设计的新的模型和我所实现的基准模型的实验结果。

Zapier 数据集上的预测结果如表 6–3所示。由于 Zapier 数据集与 IFTTT 数据集难度有所降低，并且在实验中我发现，ensemble 多个模型对于预测准确率的提高并没有明显帮助，因此对于每种模型结构，表中展示的是训练一个模型的结果。与 IFTTT 数据集的结果类似，在 Zapier 数据集上，Bi-directional LSTM embedding 比 dictionary embedding 的效果更好；而对于同样的 embedding 方法，带有注意机制的模型比不带有注意机制的模型测试效果好。不过由于这个数据集难度较低，因此即便是基准模型也能取得较好的实验效果，并且隐式注意机制与标准注意机制在测试时取得了相似的准确率。

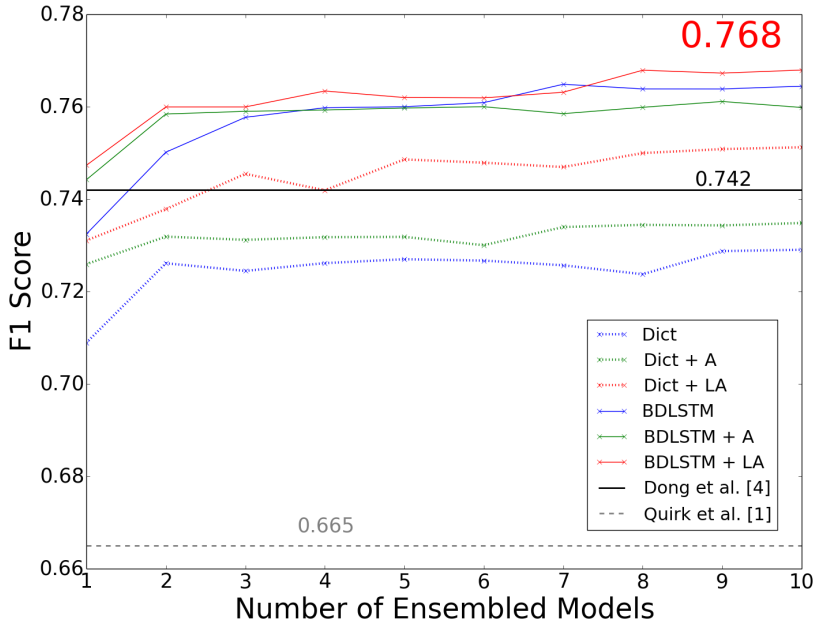

图 6–4 IFTTT 数据集上函数参数预测的 F_1 值。其中，Dict 表示 dictionary embedding，BDLSTM 表示 Bi-directional embedding，A 表示标准注意模型，LA 表示隐式注意模型，在 Dict 或 BDLSTM 之后没有加任何文字则说明没有运用注意模型。

6.5 本章小结

在这一章里，我首先对实验中所用的 IFTTT 和 Zapier 数据集的信息分别进行了介绍。接下来，我展示了各个模型在这两个数据集上的结果，并对实验结果做出了详细地分析。

第七章　单样例学习 (One-shot Learning)

在本章里，我将针对从自然语言描述到 If-Then 程序的翻译这一课题，提出一个新的场景，叫做单样例学习 (One-shot learning)，并运用 IFTTT 和 Zapier 两个数据集来展示模型在该场景下的实验结果。

单样例学习，顾名思义，就是在每个类别的样本只出现 1 次、或者非常少的次数的情况下进行训练。之前已有一些工作在图像分类任务上考虑这样一个训练数据较少的场景 [13–15]。而在我看来，这样的训练场景在本毕业论文研究的问题中也是一个需要考虑的场景。因为在我们考虑的 If-Then 程序中，无论是程序中的触发函数还是动作函数，基本上都对应于一些网站提供的服务或者应用程序的名称。而在当今这个时代，每天都会涌现出大量创业公司为用户提供各种各样的服务，所以在这个任务中。函数类别的总个数其实是动态增长的。因此，那么当 IFTTT 或 Zapier 等网站刚开始对一个新的应用程序提供支持时，包含这一应用程序的调用的 If-Then 程序必然会比较少。由于基于统计学和机器学习的模型通常需要大量的数据进行训练，在这样的场景下，模型通常预测的准确率较低。因此，在这篇毕业论文中，我用已有的数据集对这种训练场景进行模拟，并提出了一个新的模型训练方法，旨在提高模型在训练数据缺乏的情况下的预测准确率。

7.1　Zapier 数据集

对于 Zapier 数据集，我构造了一系列单样例学习的训练集，即对于 Zapier 数据集中的每个函数类别，我只在训练集中随机选取对应的 1 个、2 个、4 个、或 8 个 If-Then 程序。我将这 4 个训练集分别命名为 Train-1、Train-2、Train-4 和 Train-8。测试时，我运用与前一章相同的测试集对模型进行评估。

在单样例学习场景下，模型在 Zapier 数据集上的频道与函数预测准确率分别如表 7–1和表 7–2所示。作为比较，在前一章里，表 6–3展示了模型在全部训练集上训练后的测试结果。从实验结果我们会发现，在训练样本很少时，参数比较多的模型的预测效果反而不如参数比较少的模型，当然这里的前提是这个参数比较少的模型也具有一定的表达能力。当每个类别的训练样本数量达到 8 的时候，对于函数的预测准确率就已经能达到 80% 了，因为 Zapier 网站对于每

表 7–1 在单样例学习场景下，Zapier 数据集上的频道预测准确率

	Train-1	Train-2	Train-4	Train-8
BDLSTM+A/LA	25.4%	70.3%	85.2%	**90.9%**
BDLSTM	44.9%	83.7%	**87.7%**	**90.9%**
Dict+A/LA	**69.2%**	**84.5%**	86.0%	87.4%

表 7–2 在单样例学习场景下，Zapier 数据集上的函数预测准确率

	Train-1	Train-2	Train-4	Train-8
BDLSTM+A/LA	6.26%	24.8%	51.9%	73.2%
BDLSTM	10.1%	53.7%	71.4%	78.2%
Dict+A/LA	**26.3%**	**56.8%**	**72.6%**	**80.6%**

个 If-Then 程序对应的自然语言描述的语法是有严格规定的，所以只需要用比较少的样本就可以训练出一个具有合理泛化能力的模型。

7.2 IFTTT 数据集

对于 IFTTT 数据集而言，如果每个类别只取少量几个样本进行训练的话，实验结果会受到严重影响，因为这个数据集相比于 Zapier 数据集来说难度加大了很多，每个 If-Then 程序的自然语言描述中不仅语法随意，而且还会有很多简写和错别字。因此对于 IFTTT 数据集的单样例学习的实验，我做出了特别的设计，并提出了一个新的模型训练方法，来提高模型在该场景下的预测准确率。

7.2.1 用于模拟单样例学习场景的训练集设计

为了运用我已有的 IFTTT 数据集来模拟单样例学习场景，我设计了两个单样例学习训练集，训练集的具体构造方式如下。首先，我将所有的触发函数按照它们在训练集中出现的次数从大到小进行排序。接下来，我将它们分成两个集合，分别叫作 top100 和 non-top100，其中 top100 集合包含了出现次数最频繁的 100 个触发函数，而 non-top100 集合包含了其余的所有触发函数。

给定一个触发函数的集合 S，我们可以用如下方式构造这样一个不平衡的训练数据集：对于任意一个包含在 S 里的触发函数，将所有原始训练数据集中运用到它的 If-Then 程序，都加入到新构建的不平衡的训练数据集中；对于其余不包含在 S 里的触发函数，我在原始训练数据集中随机选取 10 个运用到该函数的 If-Then 程序，并将它们加入到新的不平衡的训练数据集中。我将这种基于 S 集

表 7–3 IFTTT 数据集上单样例学习场景的训练数据集的统计信息

	SkewTop100	SkewNonTop100
If-Then 程序的总数	61,341	10,707
S 集合中的 If-Then 程序的总数	58,376	9,707
不在 S 集合中的 If-Then 程序的总数	2,965	1,000

合构建的不平衡的训练数据集记为 $(S, \overline{S})$，并且将包含在 S 中的函数称为多数函数 (majority function)，将包含在 $\overline{S}$ 中的函数称为少数函数 (minority function)。在我的实验中，我将 S 分别设置为 top100 集合和 non-top100 集合，从而构建了两个不平衡的训练数据集。我将这两个训练数据集分别称为 SkewTop100 集合与 SkewNonTop100 集合。

我构建这样两个训练数据集的动机是为了模拟两种不同的场景。一方面，SkewTop100 模拟了这样的一个场景：当一个类似于 IFTTT 和 Zapier 这样提供在线服务的创业公司刚刚成立时，通常来说那些比较受欢迎的应用程序所对应的 If-Then 程序会比较多，而那些相对小众的应用程序所对应的 If-Then 程序则需要过一段时间之后个数才会有所增加。另一方面，SkewNonTop100 则模拟了一个与 SkewTop100 相反的场景。

IFTTT 数据集上单样例学习场景的训练数据集的统计信息如表 7–3所示。可以看出，尽管总体来说，SkewTop100 训练数据集所模拟的场景在现实生活中更加常见，但是运用 SkewNonTop100 训练数据集进行实验是一个更加具有挑战性的任务，因为这个训练数据集中包含的 If-Then 程序只占了原始训练数据集的 15.73%。

7.2.2 模型训练方法

在实验中，我评估了以下三种不同的模型训练方法，其中最后一种方法是专门针对注意机制而设计的。

1. **标准训练方法 (Standard training).** 即运用原始的训练方法进行模型训练。

2. **简化版两步训练法 (Naïve two-step training).** 首先，我们使用标准训练方法来对模型进行训练。由于单样例学习场景下，训练数据集的标签分布非常不均衡，在这种情况下，通常来说，模型在少数函数 (minority functions) 上的预测准确率会较低。因此，对于一个单样例训练数据集 $(S, \overline{S})$，我用

如下方法重新构建一个平衡的数据集 (rebalanced dataset)：对于每个包含在 S 集合里的函数，我随机取 10 个包含该函数的 If-Then 程序加入到重新构建的平衡的训练集中；对于包含在 $\overline{S}$ 中的函数，我将所有包含该函数的 If-Then 程序都加入到重新构建的平衡的训练数据集中。经过这样一个过程，在重新构建的数据集中，每个函数所对应的 If-Then 程序的个数基本均衡。于是，我以之前经过标准训练过程的模型作为初始模型，继续用这个重新构建的平衡的训练数据集对模型进行训练。

3. **两步训练法 (Two-step training).** 与简化版两步训练法 (naïve two-step training) 类似，在两步训练法中，我仍然首先对模型用标准训练方法进行训练，然后重新构建一个平衡的训练数据集。但是，与简化版两步训练法不同的是，在第二步中，简化版两步训练法是把整个模型都用重新构建的平衡的训练数据集进行训练，而在两步训练法中，我将用于计算注意权重的模型参数值都固定，只训练模型其它部分的参数。以图 4–1展示的隐式注意模型结构为例，在第二步中，我将 θ_1, θ_2, u 和 V 的参数值都保持不变, 只用重新构建的平衡训练数据集训练 θ_3 和 P 这两组参数。这样做的主要好处是，因为第二步的训练数据量远小于第一步，这种方法可以极大地减少第二步需要训练的参数量，从而使训练过程更加有效。

7.2.3 实验结果

我用我实现的模型结构来对比前文所介绍的三种模型训练方法。对于每种训练方法，我分别运用了 SkewTop100 或者 SkewNonTop100 数据集进行训练。在这组实验中我没有评估没有带注意机制的模型，因为这些模型在标准 If-Then 程序合成任务上的性能不如带有注意机制的模型好，并且这些模型没法运用两步训练法进行训练。此外，在这组实验中，我没有使用 ensemble 技术。

在 IFTTT 数据集上进行单样例学习的实验结果如图 7–1和图 7–2所示，其中对应的具体数值信息见附录 A。作为参考，单个 BDLSTM+LA 模型在触发函数的预测上能够取得 89.38% 的准确率，其中在 top100 的函数所对应的 If-Then 程序集合上的准确率是 91.11%，在 non-top100 的函数所对应的 If-Then 程序集合上的准确率是 85.12%。根据实验结果，我们可以观察到以下现象：

- 运用两步训练法的方法之后，无论是总的预测准确率，还是在少数函数上的预测准确率，相比于标准训练方法和简化两步训练法的实验结果而言，总体都有提升。

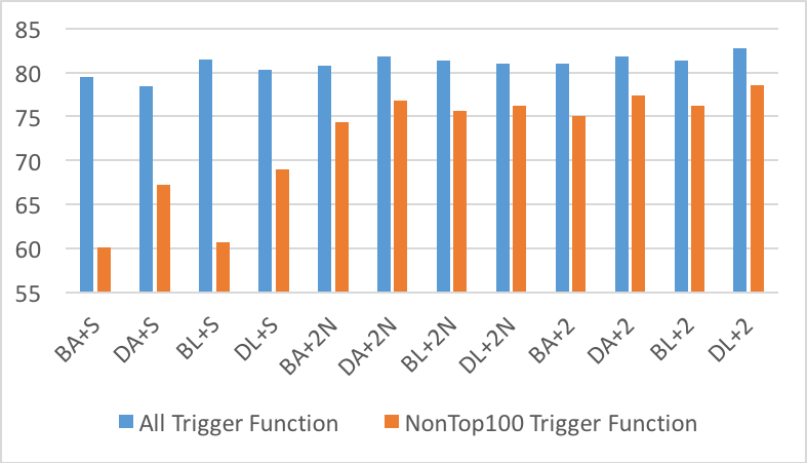

图 7–1 运用 SkewTop100 数据集进行训练的触发函数预测准确率。对于每一列 XY+Z, $X \in \{B, D\}$, 表示 embedding 的方法用的是 Bi-directional LSTM embedding 还是 dictionary embedding；$Y \in \{A, L\}$, 表示注意机制用的是标准注意机制还是隐式注意机制；$Z \in \{S, 2N, 2\}$, 表示训练方法用的是标准训练方法、简化版两步训练法、还是两步训练法。

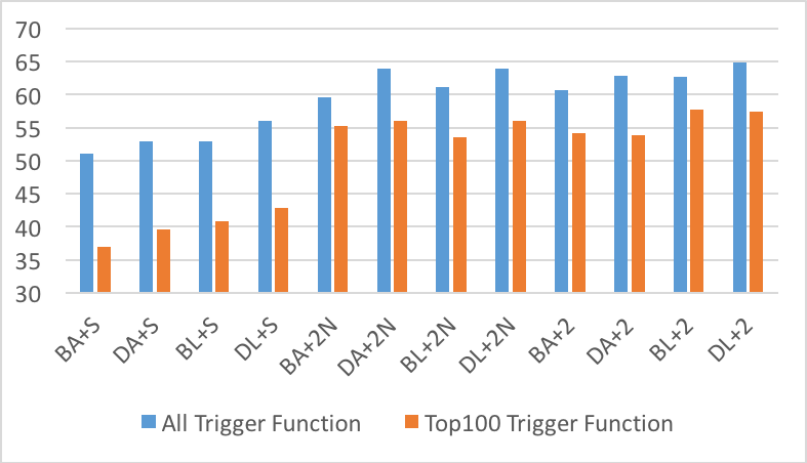

图 7–2 运用 SkewNonTop100 数据集进行训练的触发函数预测准确率。对于每一列 XY+Z, 其含义同图 7–1。

- 当使用同样的训练方法时，隐式注意模型的实验效果比标准注意模型的要好。

- 在单样例学习场景中，运用两步训练法训练出来的 Dict+LA 模型能够取得最佳的预测准确率。其中，运用 SkewTop100 数据集进行训练后，该模型能在测试集上取得 82.71% 的触发函数预测准确率；运用 SkewNonTop100 数据集进行训练后，该模型能在测试集上取得 64.84% 的触发函数预测准

确率。作为参考，当运用整个训练数据集进行训练时，Dict+LA 能够取得 89.38% 的触发函数预测准确率。注意到 SkewNonTop100 数据集只占了整个原始训练数据集的 15.73%，因此运用 SkewNonTop100 数据集对模型进行训练很具有挑战性。

- 尽管在 SkewTop100 训练数据集中，non-top100 函数所对应的 If-Then 程序只占了整个原始训练数据集的 30.54%，运用 SkewTop100 训练数据集进行训练后，Dict+LA 模型仍能够在少数函数上取得 78.57% 的预测准确率。作为参考，当运用整个原始训练数据集进行训练时，Dict+LA 模型能够在少数函数上取得 85.12% 的预测准确率。

7.3 本章小结

在这一章里，我针对从自然语言描述到 If-Then 程序的翻译这一课题，提出一个新的场景，叫做单样例学习 (One-shot learning)。对于这个场景，我分别运用已有的 IFTTT 和 Zapier 数据集设计了相关的实验，并展示了模型在该场景下的实验结果。此外，我还提出了一个新的模型训练方法，使得模型在这样一个训练数据缺乏的情况下的预测准确率得到了显著的提升。

第八章　隐式注意模型实验结果分析

在这一章里，我将对隐式注意模型的实验结果进行进一步的分析。

在图 8–1中，我展示了一些模型预测正确和错误的样例，以及模型对这些样例所计算的注意权重。这里，我展示的是 Dict+LA 模型的结果。我之所以展示 Dict+LA 的结果、而不是展示 BDLSTM+LA 的结果，是因为运用 Bi-directional LSTM embedding 的方法时，对于每个输入单词的 embedding 并不完全对应于这个单词本身，这些 embedding 还包含了这个词在文本中的上下文信息。因此，相比而言，Dict+LA 的结果可以比较方便地用于分析与阐释模型的预测行为。

从图中我们可以发现，在预测正确的样本中，那些对决定句子中哪些位置与触发条件的预测有关、哪些位置与动作条件的预测有关的单词被赋予了较高的隐式注意权重，比如介词等。一个比较有趣的例子是句子 (b)，在这个句子中，“,” 这个字符被赋予了最高的隐式注意权重。这说明隐式注意模型推测 “,” 这个字符与诸如 “to” 这样的英文单词在句子中起到了类似的作用。另外一个类似的现象是，在句子 (c) 中，“>” 这个字符被赋予了最高的隐式注意权重。这里就解释了我之前对句子进行预处理时，没有将标点符号从输入中去除掉的原因，因为人们可能会用这些符号来作为一些英文单词的简写。

在一些被错误预测的例子中，我发现一些注意权重没有被正确地赋予。在句子 (e) 中，虽然文本里没有明显地说明触发频道是 “Facebook”，但是文本中的 “photo of me” 这段文字暗示了 “me” 应该被 “tagged in the photo”。基于这点，一个人便可以推断出触发函数应该是出自 “Facebook” 这个频道，该函数的名称是 “You_are_tagged_in_a_photo”，然而 Dict+LA 模型没有从训练数据中学会这样较为复杂的分析方式。在这个例子中，我们希望模型给 “of me” 这段文字赋予较高的注意权重，然而事实却并非如此，从实验结果来看，“of” 和 “me” 这两个词的注意权重分别是 0.01 和 0.007。这说明 Dict+LA 模型没有将这两个词和 “You_are_tagged_in_a_photo” 这个函数建立起对应关系。另一方面，由于在 BDLSTM+LA 的 embedding 中，对于每个词的 embedding 会考虑其上下文信息，因此对于这句话，BDLSTM+LA 模型能够学会同时考虑 “of” 和 “me” 这两个词，因此能够对这个句子做出正确的预测。

Correct Predictions

		(a)	Post	your	Instagram	photos	to	Tumblr	(b)	Spreadsheet	with	the	daily	weather	,	triggered	at	sunrise.
weights		latent					0.75	0.14						0.57				
		trigger			0.8								0.15		0.21		0.47
		action						0.76		0.33					0.54		
label		trigger	Instagram.Any_new_photo_by_you							Weather.Sunrise							
		action	Tumblr.Create_a_photo_post							Google_Drive.Add_row_to_spreadsheet							

		(c)	Instagram	>	flickr	(d)	If	send	IFTTT	a	text	tagged	#todo,	from	cell	phone	then	quick	add	event	to	google	calendar.
weights		latent	0.12	0.81										0.16			0.42			0.17			
		trigger	0.67	0.2	0.1						0.15	0.29	0.23					0.12					
		action	0.16	0.13	0.7										0.1			0.18	0.14				0.23
label		trigger	Instagram.Any_new_photo_by_you				SMS.Send_IFTTT_an_SMS_tagged																
		action	Flickr.Upload_public_photo_from_URL				Google_Calendar.Quick_add_event																

Misclassified Examples

		(e)	Download	any	photos	of	me	to	dropbox	Truth (Trigger)
weights		latent	0.11					0.83		Facebook.You_are_tagged_in_a_photo
		trigger	0.44	0.19	0.24					Prediction
		action	0.34		0.18				0.39	Android_Photos.Any_new_photo
		(f)	Instagram			to		Wordpress		Truth (Action)
weights		latent				0.92				WordPress.Create_a_post
		trigger		0.85						Prediction
		action						0.8		WordPress.Create_a_photo_post

图 8–1 Dict+LA 模型的注意权重的例子。其中，`latent`、`trigger` 和 `action` 分别表示隐式注意权重、用于预测触发条件的最终权重、以及用于预测动作条件的最终权重。在图中，我省略了对于小于 0.1 的数值的展示。

在句子 (f) 中，Dict+LA 将触发函数预测成了 “Create_a_post”，因为模型并没有认识到这样一个事实，就是 Instagram 只与图片有关。因此，在预测触发函数时，模型在 “Instagram” 这个词上赋予的注意权重较低。不过，同句子 (e) 一样，BDLSTM+LA 仍能对这个句子做出正确预测。

8.1 本章小结

在这一章里，我对隐式注意模型的实验结果进行了进一步的分析，展示了模型的正确与错误预测案例，并分析了模型分配注意权重的特点。

第九章　结论

在这篇毕业论文中，我考虑这样一个问题：从文字描述到 If-Then(如果-则)程序的翻译。为了完成这一任务，我设计了一个新的神经网络模型，命名为“隐式注意模型 (Latent Attention)”。实验结果表明，这一新的模型在 IFTTT 数据集上相对于之前最好的工作错误率降低了 28.57%。之前研究这一任务的工作均只在 IFTTT 数据集上进行评估，而在这篇毕业论文中，我引入了一个 Zapier 数据集，并且展示了模型在该数据集上的成果。除此之外，我还针对这一任务提出了一个新的训练场景，叫单样例学习 (One-shot Learning)，并用已有的数据集进行实验来模拟这一场景。对于这一训练数据较少的场景，这篇毕业论文提出了一种新的模型训练方法，这一训练方法使得模型在这样的场景下相比传统方法训练出来的效果有非常显著的提高。

尽管在这篇毕业论文中，我在从自然语言描述到 If-Then 程序这个任务上取得了比其它所有工作更好的预测效果，但对于这一课题，仍然有不少方向值得探索。接下来，我将提出几点未来这方面工作继续发展的思路。

1. **迁移学习。** 当前，我所设计的新的模型虽然在 IFTTT 和 Zapier 两个数据集上都取得了很好的效果，但需要在大量相应训练数据集上进行训练。由于 IFTTT 和 Zapier 数据集中，样本的结构较为相似，因此一个值得探索的方向是如何在不使用、或只使用少量 IFTTT (Zapier) 训练数据集进行训练的前提下，充分运用另一个数据集来使得训练出来的模型能够在两个数据集上同时取得良好的预测效果。

2. **构建对话系统。**在前文我提到，由于用户提供的自然语言描述本身大多没有包含足够多的信息，从而导致没有办法完全正确地生成一个完整的、带有函数参数的 If-Then 程序。因此，在下一步的工作中，我们可以构建一个对话系统，当模型觉得对于 If-Then 程序的某些部分没法进行有把握的预测时，便能够自主地向用户更加深入地咨询其编写程序的意图。

3. **生成更加复杂的 If-Then 程序。** 在这篇毕业论文中，我均假定模型生成的每个 If-Then 程序均只包含一个 If-Then 语句。因此，在之后的研究中，一

个可以探索的方向是生成包含多个 If-Then 语句，以及包含多重 If-Then 关系嵌套的语句，比如类似于 Stringify.com 所支持的 If-Then 程序的形式。

附录 A　实验结果的具体数值信息

在该附录中，我展示了 IFTTT 数据集上实验结果的具体数值。

A.1　标准 If-Then 程序生成任务

标准 If-Then 程序生成任务的实验结果如表格 A–1、A–2和 A–3所示。其中，表格 A–1的数值对应于图 6–1，表格 A–2的数值对应于图 6–2，表格 A–3的数值对应于图 6–4。

A.2　单样例学习

对应于图 7–1和图 7–2的数值结果分别如表格 A–4和 A–5所示。

表 A–1 IFTTT 数据集上的频道预测准确率（对应于图 6–1）

Ensemble	1	2	3	4	5	6	7	8	9	10
Dict	71.9	72.8	73.5	74.1	74.7	80.1	80.5	79.6	80.5	81.3
Dict+A	82.4	83.0	83.6	83.2	83.2	83.2	83.7	83.9	83.6	83.7
Dict+LA	87.3	87.7	88.5	87.7	87.7	87.3	87.0	86.4	86.4	87.5
BDLSTM	84.8	89.2	90.1	90.4	90.6	90.8	90.4	90.9	91.4	91.6
BDLSTM+A	89.2	90.4	90.4	89.7	90.4	90.4	90.8	90.9	90.8	91.1
BDLSTM+LA	89.6	89.9	90.2	90.4	90.8	90.8	90.9	90.9	91.4	91.6
Yin et al. [6]					90.1					
Alvarez-Melis et al. [5]					90.1					
Dong et al. [4]					89.7					
Beltagy et al. [3]					89.1					
Quirk et al. [1]					81.4					

表 A–2 IFTTT 数据集上的函数预测准确率（对应于图 6–2）

Ensemble	1	2	3	4	5	6	7	8	9	10
Dict	71.6	74.7	74.7	75.9	76.0	76.0	75.7	75.7	76.0	76.4
Dict+A	74.0	76.0	75.9	76.0	76.4	75.7	76.5	77.6	77.2	77.2
Dict+LA	79.6	78.4	78.0	78.9	78.0	79.9	79.9	79.9	81.3	82.2
BDLSTM	78.6	81.8	81.5	82.4	84.1	85.4	85.6	86.0	85.8	85.4
BDLSTM+A	80.3	83.6	84.6	84.4	84.6	84.4	84.4	84.6	85.1	84.8
BDLSTM+LA	82.4	83.7	85.3	86.0	85.8	85.6	86.0	86.8	87.5	87.3
Beltagy et al. [3]					82.5					
Yin et al. [6]					82.0					
Dong et al. [4]					78.4					
Alvarez-Melis et al. [5]					78.2					
Quirk et al. [1]					71.0					

表 A–3 IFTTT 数据集上函数参数预测的 F_1 值（对应于图 6–4）

Ensemble	1	2	3	4	5	6	7	8	9	10
Dict	70.9	72.6	72.4	72.6	72.7	72.7	72.6	72.4	72.9	72.9
Dict+A	72.6	73.2	73.1	73.2	73.2	73.0	73.4	73.4	73.4	73.5
Dict+LA	73.1	73.8	74.5	74.2	74.9	74.8	74.7	75.0	75.1	75.1
BDLSTM	73.2	75.0	75.8	76.0	76.0	76.1	76.5	76.4	76.4	76.4
BDLSTM+A	74.4	75.8	75.9	75.9	76.0	76.0	75.8	76.0	76.1	76.0
BDLSTM+LA	74.7	76.0	76.0	76.3	76.2	76.2	76.3	76.8	76.7	76.8
Dong et al. [4]					74.2					
Quirk et al. [1]					66.5					

表 A–4 运用 SkewTop100 数据集进行训练的触发函数预测准确率。对于每一列 XY+Z, $X \in \{B, D\}$, 表示 embedding 的方法用的是 Bi-directional LSTM embedding 还是 dictionary embedding；$Y \in \{A, L\}$, 表示注意机制用的是标准注意机制还是隐式注意机制；$Z \in \{S, 2N, 2\}$, 表示训练方法用的是标准训练方法、简化版两步训练法、还是两步训练法。该结果对应于图 7–1。

	B+S	BA+S	DA+S	BL+S	DL+S	BA+2N	DA+2N	BL+2N	DL+2N	BA+2	DA+2	BL+2	DL+2
All	77.91	79.50	78.40	81.51	80.30	80.82	81.85	81.34	80.99	80.99	81.90	81.30	82.70
NonTop100	57.74	60.10	67.30	60.71	69.10	74.40	76.79	75.60	76.19	75.00	77.40	76.20	78.60

表 A–5 运用 SkewNonTop100 数据集进行训练的触发函数预测准确率。对于每一列 XY+Z, 含义同表格 A–4。该结果对应于图 7–2。

	B+S	BA+S	DA+S	BL+S	DL+S	BA+2N	DA+2N	BL+2N	DL+2N	BA+2	DA+2	BL+2	DL+2
All	47.09	51.00	52.90	52.91	56.00	59.59	63.87	61.13	63.87	60.62	62.80	62.70	64.80
Top100	31.01	37.00	39.70	40.87	42.80	55.29	56.01	53.61	56.01	54.09	53.90	57.70	57.50

参考文献

[1] Quirk C, Mooney R, Galley M. Language to code: Learning semantic parsers for if-this-then-that recipes[C]//ACL. .[S.l.]: [s.n.] , 2015.

[2] Chen X, Liu C, Shin E C, et al. Latent Attention For If-Then Program Synthesis[C]//NIPS. .[S.l.]: [s.n.] , 2016:4574–4582.

[3] Beltagy I, Quirk C. Improved Semantic Parsers For If-Then Statements[C]//ACL. .[S.l.]: [s.n.] , 2016.

[4] Dong L, Lapata M. Language to logical form with neural attention[C]//ACL. .[S.l.]: [s.n.] , 2016.

[5] Alvarez-Melis D, Jaakkola T S. Tree-structured decoding with doubly-recurrent neural networks[C]//ICLR. .[S.l.]: [s.n.] , 2017.

[6] Yin P, Neubig G. A Syntactic Neural Model for General-Purpose Code Generation[C]//ACL. .[S.l.]: [s.n.] , 2017.

[7] Zaremba W, Sutskever I, Vinyals O. Recurrent neural network regularization[J]. arXiv preprint arXiv:1409.2329, 2014.

[8] Chung J, Gulcehre C, Cho K, et al. Empirical evaluation of gated recurrent neural networks on sequence modeling[J]. arXiv preprint arXiv:1412.3555, 2014.

[9] Bahdanau D, Cho K, Bengio Y. Neural machine translation by jointly learning to align and translate[J]. arXiv preprint arXiv:1409.0473, 2014.

[10] Xu K, Ba J, Kiros R, et al. Show, attend and tell: Neural image caption generation with visual attention[J]. arXiv preprint arXiv:1502.03044, 2015.

[11] Vinyals O, Kaiser Ł, Koo T, et al. Grammar as a foreign language[C]//NIPS. .[S.l.]: [s.n.] , 2015.

[12] Sukhbaatar S, Weston J, Fergus R, et al. End-to-end memory networks[C]//NIPS. .[S.l.]: [s.n.] , 2015.

[13] Koch G, EDU T, Zemel R, et al. Siamese Neural Networks for One-shot Image Recognition[C]//ICML. .[S.l.]: [s.n.] , 2015.

[14] Fe-Fei L, et al. A Bayesian approach to unsupervised one-shot learning of object categories[C]//Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on. .[S.l.]: [s.n.] , 2003:1134–1141.

[15] Lake B M, Salakhutdinov R, Gross J, et al. One shot learning of simple visual concepts.[C]//Proceedings of the 34th Annual Conference of the Cognitive Science Society. .[S.l.]: [s.n.] , 2012:659–664.

[16] Russakovsky O, Deng J, Su H, et al. Imagenet large scale visual recognition challenge[J]. International Journal of Computer Vision, 2015, 115(3):211–252.

[17] Krizhevsky A. Learning multiple layers of features from tiny images[J]. 2009.

[18] Kushman N, Barzilay R. Using semantic unification to generate regular expressions from natural language[C]//NAACL. .[S.l.]: [s.n.] , 2013.

[19] Lei T, Long F, Barzilay R, et al. From natural language specifications to program input parsers[C]//ACL. .[S.l.]: [s.n.] , 2013.

[20] Zelle J M. Learning to parse database queries using inductive logic programming[C]//AAAI. .[S.l.]: [s.n.] , 1996.

[21] Berant J, Chou A, Frostig R, et al. Semantic Parsing on Freebase from Question-Answer Pairs.[C]//EMNLP. .[S.l.]: [s.n.] , 2013.

[22] Kate R J, Wong Y W, Mooney R J. Learning to transform natural to formal languages[C]//AAAI. .[S.l.]: [s.n.] , 2005.

[23] Branavan S R, Chen H, Zettlemoyer L S, et al. Reinforcement learning for mapping instructions to actions[C]//ACL. .[S.l.]: [s.n.] , 2009.

[24] Le V, Gulwani S, Su Z. Smartsynth: Synthesizing smartphone automation scripts from natural language[C]//MobiSys. .[S.l.]: [s.n.] , 2013.

[25] Gulwani S, Marron M. Nlyze: Interactive programming by natural language for spreadsheet data analysis and manipulation[C]//SIGMOD. .[S.l.]: [s.n.] , 2014.

[26] Ling W, Grefenstette E, Hermann K M, et al. Latent Predictor Networks for Code Generation[J]. CoRR, 2016. http://arxiv.org/abs/1603.06744.

[27] Wong Y W, Mooney R J. Learning for semantic parsing with statistical machine translation[C]//NAACL. .[S.l.]: [s.n.] , 2006.

[28] Jones B K, Johnson M, Goldwater S. Semantic parsing with bayesian tree transducers[C]//ACL. .[S.l.]: [s.n.] , 2012.

[29] Artzi Y. Broad-coverage ccg semantic parsing with amr[C]//EMNLP. .[S.l.]: [s.n.] , 2015.

[30] Mikolov T, Chen K, Corrado G, et al. Efficient estimation of word representations in vector space[C]//Proceedings of Workshop at ICLR. .[S.l.]: [s.n.] , 2013.

[31] Mikolov T, Sutskever I, Chen K, et al. Distributed representations of words and phrases and their compositionality[C]//Advances in neural information processing systems. .[S.l.]: [s.n.] , 2013:3111–3119.

[32] Mikolov T, Yih W t, Zweig G. Linguistic Regularities in Continuous Space Word Representations.[C]//NAACL. .[S.l.]: [s.n.] , 2013.

[33] Pennington J, Socher R, Manning C D. Glove: Global Vectors for Word Representation.[C]//. .[S.l.]: [s.n.] .

[34] Bird S, Klein E, Loper E. Natural language processing with Python: analyzing text with the natural language toolkit[M].[S.l.]: " O'Reilly Media, Inc.", 2009.

[35] Kingma D, Ba J. Adam: A method for stochastic optimization[J]. arXiv preprint arXiv:1412.6980, 2014.

致谢

这份工作是我在加州大学伯克利分校实习期间以第一作者发表的 NIPS 2016 论文的延伸；在这段实习期间，我的导师是 Professor Dawn Song。非常感谢 Professor Dawn Song，她的实验室里的两位学生 Chang Liu, Richard Shin, 以及目前在 Google 工作的 Mingcheng Chen 学长对我这项工作的支持与帮助！

非常感谢指导教师俞勇教授对于我这项工作的支持与建议！

非常感谢卢宏涛教授与我对于这份工作的交流与讨论！

非常感谢在每次阶段工作汇报过程中，在现场认真倾听的俞勇教授、卢宏涛教授和邓小铁教授；非常感谢在最后的毕业论文答辩中，在现场认真倾听、并与我交流讨论的俞勇教授、卢宏涛教授、邓小铁教授、张丽清教授、赵海教授和朱其立教授！

LATENT ATTENTION FOR IF-THEN PROGRAM SYNTHESIS

Programming is ubiquitous nowadays, but programmers are not. In the future, programming systems will be much more friendly for users who may not have received professional programming training. Recent years have witnessed the emergence of several commercial websites towards this goal, among which IFTTT.com is an example that allows users to simply specify a trigger and an action to construct an If-Then program, which means that the action will be taken when the trigger condition is met. Such an If-Then program enables users to easily customize their digital lives by performing tasks such as weather monitoring, organization, and scheduling, without having to code in a more full-fledged programming language.

Despite its simplicity, the If-Then program representation may still not be explanatory to users. On IFTTT.com, an If-Then program creator usually provides an optional natural language description to explain the functionality of the created If-Then program. In my view, however, a more desirable way for end-users to create an If-Then program is to describe only its functionality in text, which will be automatically translated into the corresponding If-Then program. Inspired by this idea, in this thesis, I studied the problem of translating from natural language descriptions to If-Then programs.

Automatically synthesizing If-Then programs based on their descriptions is an instance of the challenging semantic parsing problem, which has baffled researchers for decades. Recent work [1–6] studied the problem of automatically synthesizing If-Then programs from their descriptions. In particular, LSTM-based sequence-to-sequence approaches [4–6] and an approach of ensembling a neural network and logistic regression [3] were proposed to deal with this problem. In [3], however, the authors claim that the diversity of vocabulary and sentence structures makes it difficult for an RNN to learn useful representations, and their ensemble approach indeed shows better performance than the LSTM-based approaches [4–6] on the function prediction task.

In my investigation, I found that neural networks can be used to generate the correct If-Then programs in most cases, but sometimes they misplace the triggers with actions. To overcome this difficulty, in this thesis, I design a new attention architecture, called Latent Attention. With Latent Attention, a weight is learned on each token to determine its importance for prediction of the trigger or the action. Unlike standard attention methods, Latent Attention computes the token weights in a two-step process, which aims to better capture the sentence structure. By employing Latent Attention over outputs of a Bi-directional LSTM, I show that on IFTTT dataset: (1) when predicting the trigger and action functions together, the new Latent Attention model can improve over the best prior result [3] by 5 percentage points from 82.5% to 87.5%, reducing the error rate of [3] by 28.57%; (2) When predicting the trigger and action channels together, the new Latent Attention model can improve over the best prior result [5, 6] by 1.5 percentage points from 90.1% to 91.6%; (3) When predicting the entire abstract syntax trees of If-Then programs, the new Latent Attention model can improve over the best prior result [4] by 2.6 percentage points from 74.2% to 76.8%.

Meanwhile, for this natural language descriptions to If-Then program synthesis task, besides IFTTT.com, there exist other websites supporting If-Then program synthesis as well, such as Zapier.com. However, all previous works only focus on IFTTT dataset. To give a more comprehensive study of this If-Then program synthesis task, I introduce Zapier dataset, and provide experimental results using this dataset for the first time.

Besides the If-Then program synthesis task proposed by [1], I am also interested in a new scenario. When websites such as IFTTT.com release new channels and functions, in such a scenario, for a period of time, there will be very few recipes using the newly available channels and fucntions. However, we would still like to enable synthesizing If-Then programs using these new functions. The rarity of such recipes in the training set creates a challenge similar to the one-shot learning setting. In this scenario, we want to leverage the large amount of recipes for existing functions, and the goal is to achieve a good prediction accuracy for the new functions without significantly compromising the overall accuracy. To achieve this goal, I propose a novel two-step method for training the model. For the evaluation, I simulate the one-shot learning scenario with the existing datasets, and my experimental results show that the Latent Attention

model on top of dictionary embedding combining with this new training algorithm can achieve a reasonably good performance for the one-shot learning task.

Although in this thesis, my experimental results outperform the state-of-the-art results on IFTTT dataset, still, this thesis is only a first attempt towards the ultimate goal to make programming universally accessible to all computer users, and it is by no means the final destination. In the conclusion of this thesis, I provide some thoughts of the future work, and I hope the community can further explore such kinds of description-to-language translation tasks in the future.