第 5 章　知识图谱

5.1　知识图谱概述

5.1.1　什么是知识图谱

知识图谱概念

在维基百科的官方词条中：知识图谱是 Google 用于增强其搜索引擎功能的知识库。本质上，知识图谱旨在描述真实世界中存在的各种实体或概念及其关系，其构成一张巨大的语义网络图，节点表示实体或概念，边则由属性或关系构成。现在的知识图谱已被用来泛指各种大规模的知识库。《知识图谱发展报告 (2018) 》中写到，知识图谱 (Knowledge Graph) 以结构化的形式描述客观世界中概念、实体及其关系，将互联网的信息表达成更接近人类认知世界的形式，提供了一种更好地组织、管理和理解互联网海量信息的能力。

总之，知识图谱是一张由知识点相互连接而成的语义网络。两个节点，一条线，是知识图谱的基本单位，我们把这两个点叫做“实体”（entity），这条线叫做“关系”（relation）。像上述这样有方向的一段关系中，关系出发的那个节点叫做“主体”(subject)，中间的属性叫做“谓语”（predicate），指向的节点叫“宾语”(object)。形成如下的 SPO 三元组结构：

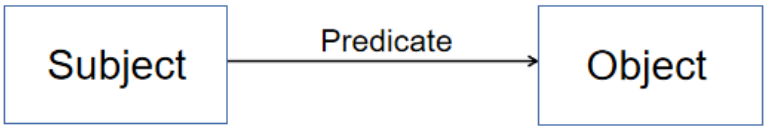


图 5.1

无数这样的三元组构成了一个巨大的网络，就是我们的知识图谱。所以我们知道了如下概念：1、知识图谱是由无数知识相互连接形成的巨大网络。2、知识图谱的基本单位是一个三元组，一个三元组可以表示两个实体之间的关系。所以我们可以说：知识图谱是表示实体之间关系的巨大网络。而这些实体的关系，是一个小“知识”，所以叫“知识”图谱。

知识图谱对搜索的改变

对于传统搜索：从前人们上网查一个东西，输入关键词后出来很多相关链接，比如搜索“周杰伦”，如下图：

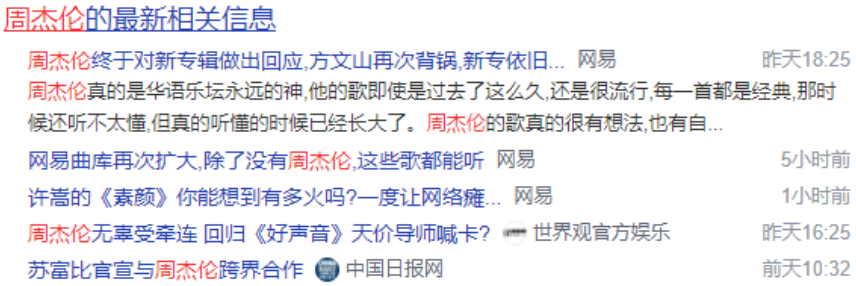


图 5.2

于是人们需要一个个点开这些链接去寻找想要了解的信息，如果想知道他的身世，可能要直接搜“周杰伦身世”，或者想知道他的歌曲，就要搜“周杰伦的歌曲”。搜索引擎只是单纯地从各个网页中提取用户输入的关键词进行匹配，然后把结果按相关度排序，再按排序返回搜索出的条目。也就是说，计算机没有去“理解”人们输入的词，而只是把这些词当做一个个字段去匹配而已。

对于知识图谱搜索：1、搜索一个关键词，它返回关于这个关键词的各方面信息，如下图：

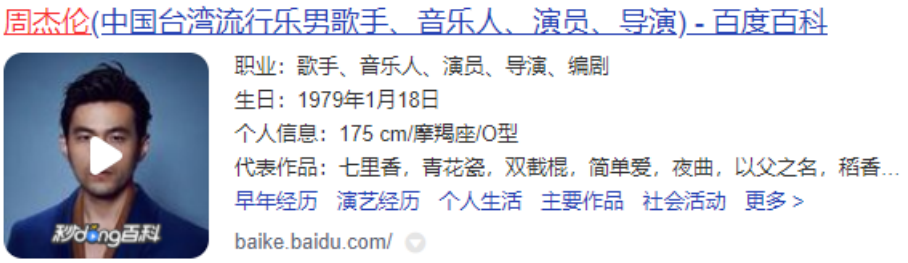


图 5.3

包括他的经历，生活，主要音乐作品，相关电影，相关艺人等。比传统搜索更全面和系统。百度百科其实就是一个大的中文知识图谱，它把中文中各种各样的“知识”连接起来，可以理解为一个大的知识网络。

2、在知识图谱搜索中，电脑“理解”了你输入的内容，从而返回了更为精确的结果，比如搜索“周杰伦身高”：

图 5.4

图中红框框出的为知识图谱搜索的结果，其他内容为普通搜索结果。

知识图谱的分类

常见的知识图谱可以分为以下几类:

图 5.5

WordNet 是最著名的词典知识库，主要用于词义消歧。WordNet 是一个在 20 世纪 80 年代由 Princeton 大学的著名认知心理学家 GeorgeMiller 团队构建的一个大型的英文词汇数据库。名词、动词、形容词和副词以同义词集合 (synsets) 的形式存储在这个数据库中。每一个 synset 代表了一个同义词集合。各个 synsets 之间通过语义关系和词性关系等边连接。WordNet 中单词之间的主要关系是同义词关系，比如 shut 和 close 是同义词，car 和 automobile 是同义词。这些同义词（表示相同概念并且在许多情况下是可以互换的单词）被分为无序的同义词集合 (synsets)。

wordnet 中一共有 11.7 万个这样的同义词集合。WordNet 主要定义了名词、动词、形容词和副词之间的语义关系。例如名词之间的上下位关系（如：“猫科动物” 是 “猫” 的上位词），动词之间的蕴含关系（如：“打鼾” 蕴含着 “睡眠”）等。

Cyc 是在 1984 年由 Douglas Lenat 开始创建。最初的目标是要建立人类最大的常识知识库。典型的常识知识如”Every tree is a plant ” 和”Planats die entally” 等。Cyc 知识库主要由术语 Terms 和断言 Assertions 组成。Terms 包含概念、关系和实体的定义。Assertions 用来建立 Terms 之间的关系，这既包括事实 Fact 描述，也包含规则 Rule 的描述。最新的 Cyc 知识库包含有 50 万条 Terms 和 700 万条 Assertions。Cyc 的主要特点是基于形式化的知识表示方法来刻画知识。形式化的优势是可以支持复杂的推理。但过于形式化也导致知识库的扩展性和应用的灵活性不够。Cyc 提供开放版本 OpenCyc。

Freebase 依靠 Wikipedia+ 领域知识 + 群体智能方法，是百科知识图谱。它包含 5813 万实体、32 亿个实体关系三元组，2000 多个概念类型和近 4 万个属性，是公开可获取的规模最大的知识图谱之一。Freebase 的目标就是收集世界上的所有信息，并且整理出信息之间的关系。比如电影数据库 FMDB，在里面可以查到每一部电影、每一个导演、每一个演员。目前 freebase 已经停止更新。

YAGO 是由德国马普研究所研制的链接数据库。YAGO 主要集成了 Wikipedia、WordNet 和 GeoNames 三个来源的数据。YAGO 将 WordNet 的词汇定义与 Wikipedia 的分类体系进行了融合集成，使得 YAGO 具有更加丰富的实体分类体系。YAGO 还考虑了时间和空间知识，为很多知识条目增加了时间和空间维度的属性描述。目前，YAGO 包含 1.2 亿条三元组知识。YAGO 是 IBM Wason 的后端知识库之一。

BabelNet 是类似于 WordNet 的多语言词典知识库。它的目标是解决 WordNet 在非英语语种中数据缺乏的问题。BabeNet 采用的方法是将 WorNet 词典与 Wikipedia 百科集成。首先建立 WordNet 中的词与 Wikipedia 的页面标题的映射，然后利用 Wikipedia 中的多语言链接，再辅以机器翻译技术，来给 WordNet 增加多种语言的词汇。BabelNet3.7 包含了 271 种语言，1400 万同义词组，36.4 万词语关系和 3.8 亿从 Wikipedia 中抽取的链接关系，总计超过 19 亿 RDF 三元组。BabelNet 集成了 WordNet 在词语关系上的优势和 Wikipedia 在多语言语料方面的优势，构建成了目前最大规模的多语言词典知识库。

ConceptNet 是常识知识库。最早源于 MIT 媒体实验室的 Open Mind Common Sense (OMCS) 项目。OMCS 项目是由著名人工智能专家 Marvin Minsky 于 1999 年创立。ConceptNet 主要依靠互联网众包、专家创建和游戏三种方法来构建。新版本导入大量开放的结构化数据，如 DBPedia、Wikinary、Wordnet 等。ConceptNet 知识库以三元组形式的关系型知识构成。ConceptNet5 版本已经包含有 2800 万关系描述。与 Cyc 相比，ConceptNet 采用了非形式化、更加接近自然语言的描述，而不是像 Cyc 那样采用形式化的谓词逻辑。与链接数据和谷歌知识图谱相比，ConceptNet 比较侧重于词与词之间的关系。从这个角度看，ConceptNet 更加接近于 WordNet, 但是又比 WordNet 包含的关系类型多。ConceptNet 完全免费开放，并支持多种语言。

DBPedia 是早期的语义网项目。DBPecia 意指数据库版本的 Wikipedia，它是从 Wikipedia 抽取出来的链接数据集。DBPedia 采用了一个较为严格的本体，包含人、地点、音乐、电影、组织机构、物种、疾病等定义。此外，DBPedia 还与 Freebase、OpenCYC、Bio2RDF 等多个数据集建立了数据链接。DBPedia 采用 RDF 语义数据模型，总共包含 30 亿 RDF 三元组。

NELL(Never-Ending Language Learner) 是卡内基梅隆大学开发的知识库。NELL 主要采用互联网挖掘的方法从 Web 自动抽取三元组知识。NELL 的基本理念是：给定一个初始的本体（少量类和关系的定义）和少量样本，让机器能够通过自学习的方式不断的从 Web 学习和抽取新的知识。目前 NELL 已经抽取了 400 多万条高置信度三元组知识。

cnSchema.org 是一个基于社区维护的开放的中文知识图谱。并且 cnSchema 分类 (classes)、数据类型 (datatype) 的词汇集包括了上千种概念、属性 (propertities) 和关系 (relations) 等常用概念定义，以支持知识图谱数据的通用性、复用性和流动性。结合中文的特点，设计者复用、连接并扩展了 Schema.org、Wikidata、Wikipedia 等已有的知识图谱 Schema 标准，为中文领域的开放知识图谱、聊天机器人、搜索引擎优化等提供可供参考和扩展的数据描述和接口定义标准。

5.1.2 知识图谱的技术流程

技术流程

1、知识获取

知识获取的目标是从海量的文本数据中通过信息抽取的方式获得知识。

2、知识表示与建模

语义网的核心就是让计算机能够理解文档中的数据，以及数据和数据之间的语义关联关系，从而使得计算机可以更加自动化、智能化处理这些信息，常用的表示方式是用资源描述框架 RDF 和本体语言 OWL。

3、知识存储

知识存储就是研究采用何种方式将已有的知识图谱进行存储。基于图的数据结构，存储方式主要有两种形式：RDF–Apache Jena 和图数据库（Graph Database)–Neo4j。

4、知识抽取

知识抽取包括实体识别和关系抽取两个部分，实体识别主要是识别出待处理文本中七类（人名、机构名、地名、时间、日期、货币和百分比）命名实体。关系抽取就是自动识别由一对概念和联系这对概念的关系构成的相关三元组。

5、知识融合

知识融合是对不同来源、不同语言或不同结构的知识进行融合，从而对已有知识图谱进行补充、更新和去重。YAGO 就是 WordNet 和 Wikipedia 融合而成；BabelNet 是融合不同语言的知识图谱，实现了跨语言的知识关联和共享。

6、知识推理

我们需要通过知识推理进行知识补全，通过知识建模、知识获取和知识融合，就可以构建一个可用的知识图谱。但是知识图谱中存在知识缺失（实体缺失、关系缺失），此时需要用推理的手段发现已有知识中隐含的知识。目前知识推理的研究主要集中在针对知识图谱中缺失关系的补足。所采用的方法是基于传统逻辑规则的方法进行推理和基于表示学习的推理。

知识推理的一个重要应用就是形成自动问答系统，关键问题在于如何将问题映射到知识图谱所支撑的结构表示中，在此基础上才能利用知识图谱中的上下文语义约束以及已有的推理规则，并结合常识等相关知识，得到正确答案。

5.1.3 知识图谱的应用

知识图谱的价值

知识图谱技术是人工智能技术的重要组成部分，以结构化的方式描述客观世界中的概念、实体及其之间的关系。知识图谱技术提供了一种更好的组织、管理和理解互联网海量信息的能力，将互联网的信息表达成更接近于人类认知世界的形式。因此，建立一个具有语义处理能力和开放互联能力的知识库，可以在智能搜索、智能问答、个性化推荐等智能信息服务中产生应用价值。

知识图谱的典型应用领域

1、信息检索：搜索引擎中对实体信息的精准聚合和匹配、对关键词的理解以及对搜索意图的语义分析等；

2、自然语言理解：知识图谱中的知识作为理解自然语言中实体和关系的背景信息；

3、问答系统：匹配问答模式和知识图谱中知识子图之间的映射；

4、推荐系统：将知识图谱作为一种辅助信息集成到推荐系统中以提供更加精准的推荐选项；

5、电子商务：构建商品知识图谱来精准地匹配用户的购买意愿和商品候选集合；

6、金融风控：利用实体之间的关系来分析金融活动的风险以提供在风险触发后的补救措施(如联系人等)；

7、公安刑侦：分析实体和实体之间的关系以获得线索等；

8、司法辅助：法律条文的结构化表示和查询来辅助案件的判决等；

9、教育医疗：提供可视化的知识表示，用于药物分析、疾病诊断等；

知识图谱的应用

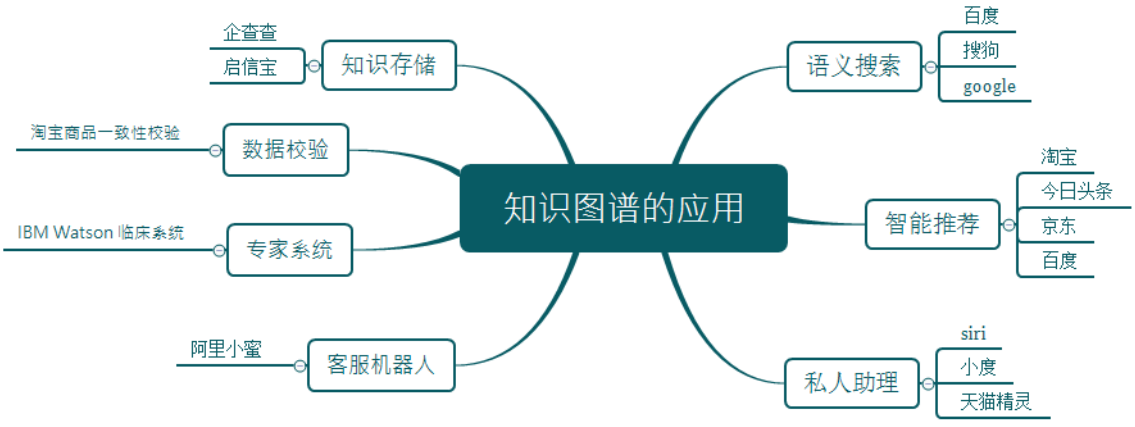


图 5.6

5.2 知识存储

5.2.1 基于关系型数据库的存储方法

面向 RDBMS 的方法——三元组方法

三元组表是将知识图谱存储到关系数据库的最简单、最直接的办法，它是在关系型数据库中建立一张具有 3 列的表，该表的模式为：

表 5.1

主语	谓语	宾语
马云	born	1964
马云	founder	阿里巴巴
任正非	born	1944
任正非	founder	华为

三元组表存储方案虽然简单明了，但三元组表的行数与知识图谱的边数一样，其最大问题在于是将知识图谱查询翻译为 SQL 查询后的三元组表自连接，当三元组表规模较大时，多个自连接操作会使 SQL 查询性能低下。

面向 RDBMS 的方法——水平表

水平表存储方案同样非常简单，与三元组表不同，它的每行记录存储一个知识图谱中一个主语的所有谓语和宾语。实际上，水平表就相当于知识图谱的邻接表。水平表的列数是知识图谱中不同谓语的数量，行数是知识图谱中不同主语的数量。

表 5.2

主语	born	founder	...	employees	headquarters
马云	1964	阿里巴巴	...		
任正非	1944	华为	...		
HarmonyOS					
阿里巴巴			...	254702	浙江省杭州市
华为			...	194000	广东省深圳市

水平表的优点在于：与三元组表相比，水平表的查询大为简化，仅需单表查询即可完成该任务，不用进行连接操作。

水平表的缺点在于：

1、所需列的数目等于知识图谱中不同谓语数量，在真实知识图谱数据集中，不同谓语数量可能为几千个到上万个，很可能超出关系数据库允许的表中列数目的上限；

2、对于一行来说，仅在极少数列上具有值，表中存在大量空值，空值过多会影响表的存储、索引和查询性能；

3、在知识图谱中，同一主语和谓语可能具有多个不同宾语，即一对多联系或多值属性，而水平表的一行一列上只能存储一个值，无法应对这种情况（可以将多个值用分隔符连接存储为一个值，但这违反关系型数据库设计的第一范式）；

4、知识图谱的更新往往会引起谓语的增加、修改或删除，即水平表中列的增加、修改或删除，这是对于表结构的改变，成本很高。

面向 RDBMS 的方法——属性表

属性表 (Property Table) 存储方案是对水平表的细化，将同类主语分到一个表中，不同类主语分到不同表中。这样就解决了表中列的数目过多的问题。把一个水平表分为了 person（人）、os（操作系统）和 company（公司）三个表。

表 5.3

主语	born	founder	home
马云	1964	阿里巴巴	浙江省杭州市
任正非	1944	华为	贵州省安顺市

表 5.4

主语	developer	version	kernel	preceded
HarmonyOS	华为	2.0	Linux	1.0

表 5.5

主语	product	employees	headquarters
阿里巴巴	淘宝、闲鱼、支付宝	254702	浙江省杭州市
华为	智能手机、电脑	194000	广东省深圳市

属性表的优点：属性表既克服了三元组表的自连接问题，又解决了水平表中列数目过多的问题。

属性表仍有缺点：

1、对于规模稍大的真实知识图谱数据，主语的类别可能有几千个到上万个，按照属性表方案，需要建立几千个到上万个表，这往往超过了关系型数据库的限制；

2、对于知识图谱上稍复杂的查询，属性表方案仍然会进行多个表之间的连接操作，从而影响查询效率；

3、即使在同一类型中，不同主语具有的谓语集合也可能存在较大差异，这样会造成与水平表中类似的空值问题；

4、水平表方案中存在的一对多联系或多值属性存储问题仍然存在。

面向 RDBMS 的方法——垂直表

垂直划分 (Vertical Partitioning) 存储方案是由美国麻省理工学院的 Abadi 等人在 2007 年提出的 RDF 数据存储方法。该方法以三元组的谓语作为划分维度，将 RDF 知识图谱划分为若干张只包含（主语，宾语）两列的表，表的总数量即知识图谱中不同谓语的数量；也就是说，为每种谓语建立一张表，表中存放知识图谱中由该谓语连接的主语和宾语值。从上面的案例可得出，10 种谓语对应着 10 张表，每张表都只有主语和宾语列。

born

主语	宾语
马云	1964
任正非	1944

version

主语	宾语
HarmonyOS	2.0

product

主语	宾语
阿里巴巴	淘宝
阿里巴巴	闲鱼
阿里巴巴	支付宝
华为	智能手机
华为	电脑

developer

主语	宾语
HarmonyOS	华为

kernel

主语	宾语
HarmonyOS	Linux

proceded

主语	宾语
HarmonyOS	1.0

employees

主语	宾语
阿里巴巴	254702
华为	194000

home

主语	宾语
马云	浙江省杭州市
任正非	贵州省安顺市

founder

主语	宾语
马云	阿里巴巴
任正非	华为

headquarters

主语	宾语
阿里巴巴	浙江省杭州市
华为	广东省深圳市

图 5.7

5.2.2 面向 RDF 的存储方法

面向 RDF 三元组的数据库

RDF 是 W3C 指定的在语义万维网上表示和交换机器可理解信息的标准数据模型。RDF(Resource Description Framework)，即资源描述框架，其本质是一个数据模型 (Data Model)。它提供了一个统一的标准，用于描述实体或资源。简单来说，就是表示事物的一种方法和手段。RDF 形式上表示 SPO 三元组，有时候也称为一条语句 (Statement)，知识图谱中我们也称其为一条知识。RDF 三元组集合中：每个 Web 资源具有一个 HTTP URI 作为唯一的 id；一个 RDF 图定义为三元组 (s,p,o) 的有限集合；每个三元组代表一个陈述句；(s,p,o) 表示资源 s 与资源 o 之间具有联系 p，或者表示资源 s 具有属性 p 且取值为 o。SPARQL 是 W3C 指定的 RDF 图数据的标准查询语言。SPARQL 借鉴了 SQL，同属于声明式查询语言。由于 RDF 是 W3C 推荐的表示语

义网上关联数据 (Linked Data) 的标准格式，RDF 也是表示和发布 Web 上知识图谱的最主要数据格式之一。面向 RDF 的三元组数据库是专门为存储大规模 RDF 数据而开发的知识图谱数据库，其支持 RDF 的标准查询语言 SPARQL。主要包括开源和商业 RDF 三元组数据库。

开源 RDF 三元组数据库包括：Apache 旗下的 Jena，Eclipse 旗下的 RDF4J，RDF-3X，gStore 商业 RDF 三元组数据库包括：Virtuoso，AllegroGraph，GraphDB，BlazeGraph。

开源数据库–Apache Jena

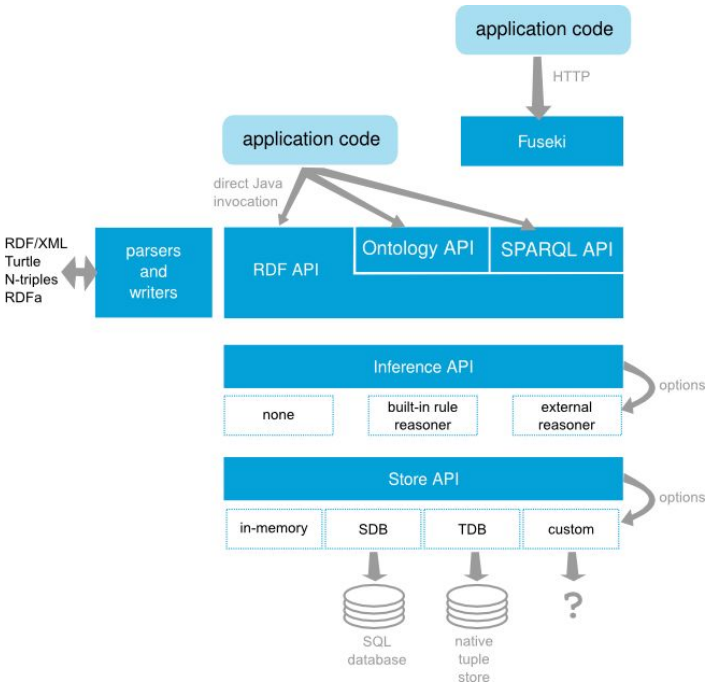


图 5.8

Jena 是 HP 实验室语义网研究项目组的开源工具，是一个基于 Java 的语义网应用框架。它是一个免费开源的支持构建语义网络和数据链接应用的 Java 框架，支持内存和永久存储。

最底层是数据库，包含 SQL 数据库和原生数据库，其中的 SDB 用来导入 SQL 数据库，TDB 导入 RDF 三元组。数据库上层的是内键的和外联的推理接口，再上层就是 SPARQL 查询接口，通过直接使用 SPARQL 语言或通过 REFO 等模块转换成的 SPARQL 语言进行查询。在上方有一个 Fuseki 模块，它相当于一个服务器端，我们的这些操作就是在它提供的端口上进行的。

Apache Jena 的安装教程

Jena 是基于 Java 的语义网应用框架，所以在安装 Jena 之前需要先安装 Java，需要注意 Java 版本要与 Jena 版本匹配，例如 Java11 版本要与 Jena4.0 以上才可以匹配，Java8 版本要与 Jena3.0 以上才可以匹配。我们在 Java 官网下载之后按照默认选项一步一步安装，这里以 Java11 为例。接下来需要配置环境变量，右键“此电脑”——“属性”——“高级系统设置”——“高级”——“环境变量”——“系统变量”——“新建”，输入变量名：JAVA_HOME；变量值：D:\JDK\jdk1.8.0_161（此处是电脑 Java 安装目录路径，需要根据自己实际情况输入）。再

回到环境变量窗口的系统变量处的新建位置，新建环境变量。变量名：CLASSPATH；变量值：.;%JAVA_HOME %\lib\dt.jar;%JAVA_HOME%\lib\tools.jar。再回到环境变量窗口的系统变量处的新建位置，双击 Path 选项，点击新建，添加“%JAVA_HOME%\bin”；再次点击新建，添加“%JAVA_HOME%\jre\bin”。

在搜索引擎打开 Jena 官网，下载 Jena 和 Jena Fuseki 并把这两个软件放到一个文件夹内命名为 Jena（以 4.4.0 为例）。接下来需要配置环境变量，右键“此电脑”——“属性”——“高级系统设置”——“高级”——“环境变量”——“系统变量”——“新建”，输入变量名：JENA_HOME；变量值：D:\jena\apache-jena-4.4.0（此处是电脑 Jena 安装目录路径，需要根据自己实际情况输入）。再回到环境变量窗口的系统变量处的新建位置，双击 Path 选项，点击新建，添加“%JENA_HOME%\bin”。打开文件夹 Jena 下的 apache-jena-4.4.0 的 bin 文件夹下输入 cmd，在命令窗口 cmd 输入 sparql-version 来判断是否安装成功。如果命令窗口 cmd 出现 version4.4.0，则安装成功。打开文件夹 Jena 下的 apache-jena-fuseki-4.4.0，输入 cmd，在命令窗口 cmd 输入 fuseki-server.bat，如果命令窗口 cmd 出现 on port 3030，我们在在浏览器输入 $https://localhost:3030/$ 就可以使用 Jena。

案例：音乐知识图谱

首先利用 python 代码随机生成 1000 条三元组数据。代码如下：

```
import random
import sys
#随机生成带编码的示例数据
track_name = "<http://kg.course/music/track_%05d>
<http://kg.course/music/track_name> \"track_name_%05d\" ."
track_album = "<http://kg.course/music/track_%05d>
<http://kg.course/music/track_album>
<http://kg.course/music/album_%04d> ."
album_name = "<http://kg.course/music/album_%04d>
<http://kg.course/music/album_name> \"album_name_%04d\" ."
track_artist = "<http://kg.course/music/track_%05d>
<http://kg.course/music/track_artist>
<http://kg.course/music/artist_%03d> ."
artist_name = "<http://kg.course/music/artist_%03d>
<http://kg.course/music/artist_name>  \"artist_name_%03d\" ."
tag_name = "<http://kg.course/music/track_%05d>
<http://kg.course/music/track_tag>  \"tag_name_%02d\" ."
#生成一千个三元组
total_sum = 1000
triples_sum = 0
triples = []
if len(sys.argv) >= 2:
    print('sys.argv = ', sys.argv)
    try:
        total_sum = int(sys.argv[1])
    except:
```

```
        total_sum = 1000
```

接下来我们要给编码赋值，代码如下：

```
#给编码随机赋值，循环生成字符串
for i in range(1, total_sum):
    track_str = track_name % (i, i)
    s = random.randint(1, 10)
    album_str = track_album % (i, i/s + 1)
    album_name_str = album_name % (i/10 + 1, i/10 + 1)
    t = random.randint(1, 100)
    track_artist_str = track_artist % (i, i % t)
    artist_name_str = artist_name % (i % t, i % t)
    k = random.randint(1, 10)
    tag_name_str = tag_name % (i, i % k + 1)
 #将生成的数据随机加入序列
triples.append(track_str)
triples_sum += 1
if total_sum <= triples_sum:
   break
triples.append(album_str)
triples_sum += 1
if total_sum <= triples_sum:
   break
triples.append(album_name_str)
triples_sum += 1
if total_sum <= triples_sum:
   break
triples.append(track_artist_str)
triples_sum += 1
if total_sum <= triples_sum:
   break
"""
triples.append(artist_name_str)
triples_sum += 1
if total_sum <= triples_sum:
   break
"""
triples.append(tag_name_str)
triples_sum += 1
if total_sum <= triples_sum:
   break
#生成示例数据并保存为nt文件
filename = "music_%d_triples.nt" % total_sum
with open(filename, "w+") as fd:
fd.write("\n".join(triples))
```

SPARQL 查询语句示例：

```
1.查询某一艺术家的所有歌曲
PREFIX music: <http://kg.course/music/>
SELECT  DISTINCT   ?trackID
WHERE {?trackID  music:track_artist  music:artist_001}
2.查询某一首歌曲名的专辑信息，使用中文来当变量名
PREFIX music: <http://kg.course/music/>
SELECT ?歌曲id ?专辑id ?专辑名
WHERE {?歌曲id  music:track_name   "track_name_00001"   .
       ?歌曲id  music:track_album  ?专辑id  .
       ?专辑id  music:album_name   ?专辑名}
3.查询某一首歌曲名的专辑信息，对变量名添加描述
PREFIX music: <http://kg.course/music/>
SELECT ?歌曲id ?专辑id (CONCAT("专辑名",":",?专辑名) AS ?专辑信息)
WHERE {?歌曲id  music:track_name   "track_name_00001"   .
       ?歌曲id  music:track_album  ?专辑id  .
       ?专辑id  music:album_name   ?专辑名}
4.查询某个专辑里面的所有歌曲,限制前两条
PREFIX music: <http://kg.course/music/>
SELECT  ?trackID
WHERE {?albumID    music:album_name   "album_name_0002" .
       ?trackID    music:track_album   ?albumID limit 2}
5.对某个专辑里面的所有歌曲计数
PREFIX music: <http://kg.course/music/>
SELECT  (COUNT(?trackID) as ?num)
WHERE {?albumID    music:album_name   "album_name_0002" .
       ?trackID    music:track_album   ?albumID}
6.查询某一首歌是哪一个艺术家的作品
PREFIX music: <http://kg.course/music/>
SELECT ?trackID  ?artistID
WHERE {?trackID  music:track_name   "track_name_00001"  .
       ?trackID music:track_artist ?artistID}
7.查询某一艺术家唱过歌曲的所有类型并排序
PREFIX music: <http://kg.course/music/>
SELECT  DISTINCT ?tag_name
WHERE {?trackID    music:track_artist  music:artist_001  .
       ?trackID    music:track_tag    ?tag_name
       ORDER BY ?tag_name}
```

5.2.3 图数据库存储方法

图数据库介绍

图数据库起源于欧拉和图理论 (graph theory)，也可称为面向或基于图的数据库，对应的英文是 Graph Database。图数据库的基本含义是以“图”这种数据结构存储和查询数据。它的数

据模型主要是以节点和关系（边）来体现，也可处理键值对。它的优点是快速解决复杂的关系问题。图具有如下特征：1、包含节点和边。2、节点上有属性（键值对）。3、边有名字和方向，并总是有一个开始节点和一个结束节点。4、边也可以有属性。

图数据库分为原生数据库和非原生数据库两类，其中原生数据库包括基于 main memory 和基于 disk。非原生数据库包括基于关系的数据库（RDBMS）和基于 NoSQL 的数据库，例 Key-value: 如 Redis、Column family；Document store：如 Mongo DB；Graph database：如 Neo4j（广义上的原生数据库）。

原生图数据库–Neo4j

Neo4j 是一个高性能的 NOSQL 图数据库，它将结构化数据存储在网络上而不是表中。它是一个嵌入式的、基于磁盘的、具备完全的事务特性的 Java 持久化引擎，但是它将结构化数据存储在网络（从数学角度叫做图）上而不是表中。Neo4j 也可以被看作是一个高性能的图引擎，该引擎具有成熟数据库的所有特性。在一个图中包含两种基本的数据类型：Nodes (节点) 和 Relationships (关系)。Nodes 和 Relationships 包含 key/value 形式的属性。Nodes 通过 Relationships 所定义的关系相连起来，形成关系型网络结构。

Neo4j 的安装教程

Neo4j 是一个具备完全的事务特性的 Java 持久化引擎，所以在安装 Neo4j 之前需要先安装 Java，需要注意 Java 版本要与 Neo4j 版本匹配，例如 Java11 版本要与 Neo4j4.0 以上才可以匹配，Java8 版本要与 Neo4j3.0 以上才可以匹配。我们在 Java 官网下载之后按照默认选项一步一步安装，这里以 Java11 为例。接下来需要配置环境变量，右键“此电脑”——“属性”——“高级系统设置”——“高级”——“环境变量”——“系统变量”——“新建”，输入变量名：JAVA_HOME；变量值：D:\JDK\jdk1.8.0_161（此处是电脑 Java 安装目录路径，需要根据自己实际情况输入）。再回到环境变量窗口的系统变量处的新建位置，新建环境变量。变量名：CLASSPATH；变量值：.;%JAVA_HOME%\lib\dt.jar;%JAVA_HOME%\lib\tools.jar。再回到环境变量窗口的系统变量处的新建位置，双击 Path 选项，点击新建，添加 “%JAVA_HOME%\bin”；再次点击新建，添加 “%JAVA_HOME%\jre\bin”。

在搜索引擎打开 Neo4j 官网，下载 Neo4j（以 4.0.4 为例）。接下来需要配置环境变量，右键“此电脑” —— “属性” —— “高级系统设置” —— “高级” —— “环境变量” —— “系统变量” —— “新建”，输入变量名：NEO4J_HOME；变量值：D:\neo4j\neo4j-community-4.0.4（此处是电脑 Neo4j 安装目录路径，需要根据自己实际情况输入）。再回到环境变量窗口的系统变量处的新建位置，双击 Path 选项，点击新建，添加 “\%NEO4J_HOME%\bin”。打开命令窗口 cmd 输入 neo4j.bat console 或者 neo4j console，如果命令窗口 cmd 出现 $https://localhost:7474/$，则安装成功。我们在在浏览器输入 $https://localhost:7474/$，就会进入 Neo4j 的页面，就可以使用 Neo4j。注意初次打开需要输入账户名字和密码，默认的用户名和密码均为 neo4j，且登陆后可以自行设置密码。

简单 Cypher 语法介绍

```
1.插入节点。插入一个Person类别的节点，且这个节点有几个属性值。
CREATE(:Person{name:'张杰',birthdate:'1983年',born_in:'四川成都'})
```

```
CREATE(:Person{name:'谢娜',birthdate:'1981年',born_in:'四川德阳'})
CREATE(:Person{name:'何炅',birthdate:'1974年',born_in:'湖南长沙'})
2.插入边。插入一条a到b的有向边，且边的类别为Follow，r后面是关系。
MATCH(a:Person),(b:Person)
WHERE a.name = '张杰' AND b.name = '谢娜'
CREATE(a)-[r:妻子]->(b);
MATCH(a:Person),(b:Person)
WHERE a.name = '张杰' AND b.name = '谢娜'
CREATE(b)-[r:丈夫]->(a);
MATCH(a:Person),(b:Person)
WHERE a.name = '张杰' AND b.name = '何炅'
CREATE(a)-[r:朋友]->(b);
注意:
CREATE(a)-[r:4]->(b);          r后面不能直接是数字
CREATE(a)-[r:"4"]->(b);        加双引号、单引号都不行
CREATE(a)-[r:权重4]->(b);      可以这样
3.更新节点。更新一个Person类别的节点，设置新的属性值。
MATCH(n:Person { name: '张杰' })
SET n.birthdate = '1982年';
4.删除节点。删除这个节点和这个节点有关的联系。
MATCH(n:Person { name:'何炅' })
DETACH DELETE n;
5.查询两个节点之间的关系。
MATCH(a:Person { name:'张杰' })-[r]->(b:Person { name:'谢娜' })
RETURN type(r);
6.查询一个节点的其中一个关系的所有节点。
MATCH(:Person {name:'张杰' })-[r:妻子]->(Person)
RETURN Person.name;
7.删除边
MATCH(a:Person)-[r:妻子]->(b:Person)
WHERE a.name = '张杰' AND b.name = '谢娜'
DELETE r;
```

案例：豆瓣电影知识图谱

首先利用 lstlisting 爬取目标网站 https://book.douban.com/top250 中的书籍相关说明，生成 csv 格式的数据。代码如下：

```
# 导入相应的库文件
from lxml import etree
import requests
import csv
# 创建csv
fp = open('data.csv', 'wt', newline='', encoding='utf-8')
# 写入header
writer = csv.writer(fp)
writer.writerow(('name', 'author',  'rate', 'comment'))
```

```
# 构造urls
urls = ['https://book.douban.com/top250?start={}'.format(str(i)) for i in range
    (0,250,25)]
# 加入请求头
headers = {'User-Agent':'Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36'
    '(KHTML, like Gecko) Chrome/55.0.2883.87 Safari/537.36'}
for url in urls:
    html = requests.get(url, headers=headers)
    selector = etree.HTML(html.text)
    # 取大标签，以此循环
    infos = selector.xpath('//tr[@class="item"]')
for info in infos:
    name = info.xpath('td/div/a/@title')[0]  # 书名
    book_infos = info.xpath('td/p/text()')[0]  # 作者、出版社等
    author = book_infos.split('/')[0]  # 作者
    rate = info.xpath('td/div/span[2]/text()')[0]  # 评分
    comments = info.xpath('td/p/span/text()')  #评语
    comment = comments[0] if len(comments) != 0 else "空"
# 写入数据
writer.writerow((name, author, rate,comment))
# 关闭csv文件
fp.close()
```

此代码跑完后在当前目录会生成 data.csv。然后把 data.csv 放入 neo4j-community-3.3.1（版本号可能不同）文件夹下的 import 文件夹中。打开 neo4j，通过输入命令行进行数据导入。接下来把书籍名字单独作为一个类别，把书籍的三个属性放在了一起，导入图数据库，让数据变得可视化起来。Cypher 语句如下：

```
#插入“书籍名字”、“书籍评分”、“书籍作者”三个节点。
LOAD CSV WITH HEADERS FROM "file:///data.csv" AS line
CREATE(:书籍名字 {name:line.name,class:line.comment})
CREATE(:书籍评分 {name:line.rate})
CREATE(:书籍作者 {name:line.author})
#在“书籍名字”和“书籍评分”之间插入"rate"这条边。
LOAD CSV WITH HEADERS FROM "file:///data.csv" AS line
MATCH (entity1:书籍名字{name:line.name}) , (entity2:书籍评分{name:line.rate})
CREATE(entity1)-[:rate{type:line.relation,name:'评分'}]->(entity2)
#在“书籍名字”和“书籍作者”之间插入“作者”这条边。
LOAD CSV WITH HEADERS FROM "file:///data.csv" AS line
MATCH (entity1:书籍名字{name:line.name}),(entity2:书籍作者{name:line.author})
CREATE(entity1)-[:作者{type:line.relation}]->(entity2)
```

5.3 知识表示与建模

5.3.1 知识表示

知识表示可以从以下四个角度理解：

(1) 知识表示是一种代理，基于对事物的表示，我们无需实践而是通过思考和推理就可以得到有关外部世界的结论。

(2) 知识表示是一组本体论约定的集合，说明我们以什么样的方式来思考世界。

(3) 知识表示是智能推理的组成部分，推理需要对知识进行表示，但知识表示不是推理的全部。知识表示是高效计算的媒介，通过对知识进行有效组织，支持高效的推理。

(4) 知识表示是人类表达的媒介，基于通用表示框架，方便人们表达和分享对世界的认知。

5.3.2 AI 早期知识表示方法

一阶谓词逻辑 (First-Order-Logic)

一阶谓词逻辑是最早出现的一种语言表示形式，是一种形式系统 (Formal System)，即形式符号推理系统，也叫一阶谓词演算、低阶谓词演算 (Predicate Calculus)、限量词 (Quantifier) 理论，也有人称其为“谓词逻辑”。一阶谓词逻辑是原子命题的下一层级，可以看成是对原子命题做进一步分析，分析出其中的个体词、谓词、量词，研究它们的形式结构的逻辑关系、正确的推理形式和规则。比如，穷人不可能是皇帝，某一个人是皇帝，推导出这个人不是穷人类似这种知识推理的方案。有了一阶谓词逻辑，可以把它经过联立，形成所谓的高阶谓词逻辑，就类似线性代数再加上激活函数能得到所谓的非线性代数，所以没有高阶谓词逻辑这个说法，一阶谓词逻辑足够使用了。

产生式规则 (Production Rule)

产生式系统是一种更广泛的规则系统，和一阶谓词逻辑有关联，也有区别。产生式规则在一阶谓词逻辑表示的基础上，进一步解决了不确定性知识的表示，产生式规则以三元组（对象，属性，值）或者（关系，对象 1，对象 2），通过进一步加入置信度形成四元组（对象，属性，值，置信度）或者（关系，对象 1，对象 2，置信度）的形式来表示事实，并使用 P→Q 或者 IF P THEN Q 的形式用于表示规则，这种表示方法可以表示不确定性知识和过程性知识，具有一致性和模块化等优点，通过规则可以实现推理功能，广泛运用于上世纪 70 年代的专家系统当中。

例如，若已知一个图形有三边，且三边相等，那此图形是等边三角形便是用产生式规则得到的推理结果。基本形式：如事实“老李年龄是 35 岁”，便写成（Lee,age,35）；事实“老李、老张是朋友”，可写成（friend,Lee,Zhang）。常用结构：原因 → 结果：天下雨，地上湿；条件 → 结论：将冰加热到 0 度以上，冰会融化成水；前提 → 操作：如果能找到合适的杠杆和支点，则可以翘起地球；事实 → 进展：夜来风雨声，花落知多少；情况 → 行为：手机开机了，则意味着可以收到别人发我的信息。

框架 (Framework)

框架表示法是明斯基于 1975 年提出来的，特点是善于表示结构性知识，能把知识的内部结构关系以及知识之间的特殊关系表示出来，并把与某个实体的相关特性集中在一起，而一阶谓词逻辑和产生式规则只能一条一条的表示知识，即使这些知识是对同一个实体而言。框架表示法认为人们对现实世界中各种事物的认识都是以一种类似于框架的结构存储在记忆中的。当面临一个新事物时，就从记忆中找出一个合适的框架，并根据实际情况对其细节加以修改、补充，从而形成对当前事物的认识。

框架是描述对象（一个事物、一个事件、一个概念）属性的一种数据结构，在框架表示法中，框架被认为是知识表示的最基本单元。框架是由若干结点和关系（统称为槽 slot）构成的网络，是语义网络一般化和形式化的一种结构，同语义网络没有本质区别，将语义网络中结点间弧上的标注也放入槽内就成了框架表示法，框架基本组成如表6.1展示，表6.2是关于洪水的框架实例。

框架表示法的优点：对于知识的描述非常完整和全面、基于框架的知识库质量非常高、允许数值计算。缺点：构建成本非常高、对知识库的质量要求非常高、框架的表达形式不灵活，很难同其它形式的数据集相互关联使用。

表 5.6: 框架基本组成

< 框架名 >		
槽名 1:	侧面名 1	值 1, 值 2,. . ., 值 p1
	侧面名 2	值 1, 值 2,. . ., 值 p2
	. . .	. . .
	侧面名 m1	值 1, 值 2,. . ., 值 pm1
. . .	. . .	. . .
槽名 n:	侧面名 1	值 1, 值 2,. . ., 值 r1
约束:	约束条件 1	
	. . .	
	约束条件 n	

表 5.7: 河南洪水

洪水实例：河南洪水		
槽 1:	< 时间 >:	2021 年 7 月 17 日至 8 月 2 日
槽 2:	< 地点 >:	河南
槽 3:	< 伤亡 >	
	侧面 3.1:	< 死亡人数 >:302 人
	侧面 3.2:	< 失踪人数 >:50 人

	侧面 3.2:	< 受灾人数 >:1481.4 万人
槽 4:	< 损失 >	
	侧面 4.1:	< 直接经济损失 >:1337.15 亿元
槽 5:	< 震级 >:	8 级

语义网络 (Semantic Network)

语义网络 (semantic network) 是一种以网络格式表达人类知识构造的形式，是人工智能程序运用的表示方式之一，它是由节点和连接节点之间的弧组成。语义网络中的节点表示各种事物、概念、情况、属性、动作、状态等，每个节点可以带有若干属性，一般用框架或元组表示，此外，节点还可以是一个语义子网络，形成一个多层次的嵌套结构。语义网络中的弧表示各种语义联系，指明它所连接的节点间某种语义关系。节点和弧都必须带有标识，以便区分各种不同对象以及对象间各种不同的语义联系。最简单的语义网络是一个三元组：(节点 1，关系，节点 2)。

例子“每个老师都教过一个学生”用语义网络表示：

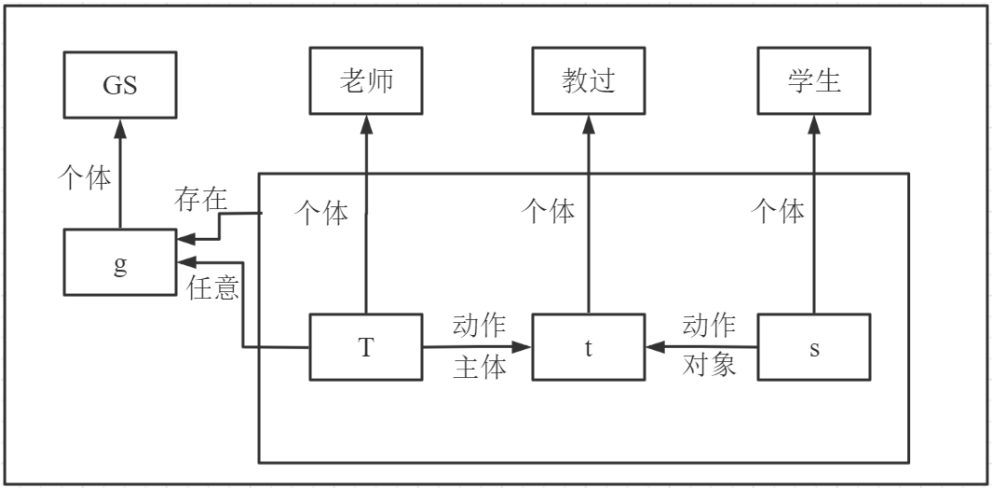


图 5.9

语义网络的优点：把各个节点之间的联系以明确、简洁的方式表示出来，是一种直观的表示方法；着重强调事物间的语义联系，体现了人类思维的联想过程，符合人们表达事物间的关系，因此把自然语言转换成语义网络较为容易；具有广泛的表示范围和强大的表示能力，用其他形式的表示方法能表达的知识几乎都可以用语义网络来表示；把事物的属性以及事物间的各种语义联系显示地表示出来，是一种结构化的知识表示法。

缺点：推理规则不明了，不能充分保证网络操作所得推论的严格性和有效性；一旦节点个数太多，网络结构复杂，推理就难以进行；不便于表达判断性知识与深层知识。

5.3.3 基于语义网的知识表示框架

RDF(Resource Description Framework)

RDF(Resource Description Framework) 是一种资源描述框架，其中 Resource 指页面、图片、视频等任何具有 URI 标识符的事物，都可统称为资源；Description 表示属性、特征和资源之间的关系；Framework 是模型、语言和描述的语法。

RDF 假定任何复杂的语义都可通过若干个三元组的组合来表达，并定义这种三元组的形式为“对象——属性——值”或“对象——对象——关系”，其中需要公开或通用的资源，都会绑定一个可识别的通用资源表示符 (URI)。而这两种表示形式都可看成是主、谓、宾这种结构，即每份知识可以被分解为此形式 (subject（主），predicate（谓），object（宾））。

RDF Schema(RDFS) 简介

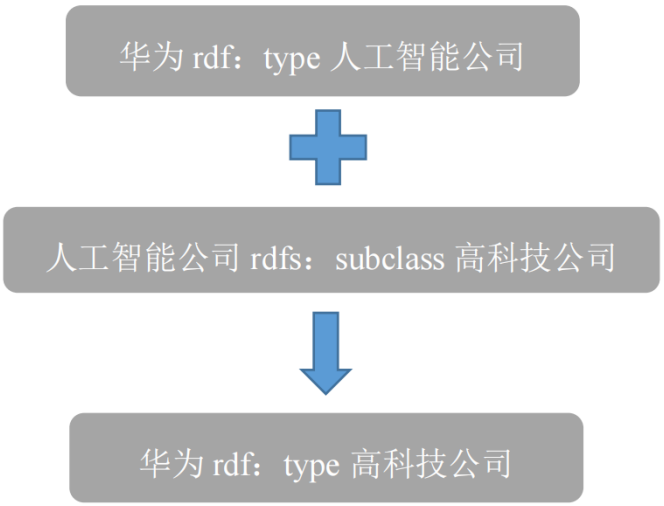


图 5.10

RDFS 可以看成是 RDF 的扩展，RDFS 在 RDF 基础上提供了一个术语、概念等的定义方式，以及哪些属性可以应用到哪些对象上，主要关注类别和属性的层次结构以及继承关系等。即在 RDF 的基础上定义了如下词汇：Class（类），subClassOf（子类），type（类型），Property（属性），subPropertyOf（子属性），Domain，Range。例如“人”是一个类，而具体的实体例如“小红”便是类“人”的子类，即类之间有继承性，而类型指的是属性值的类型，属性是指类具有的属性，例如人具有身高、体重等属性，属性之间也存在继承关系，Domain 指属性的主语的范围，Range 指属性的宾语的范围，注意这里的属性指的都是对象属性。属性可分为对象属性和数据属性，对象属性和数据属性的定义可以这么理解：假如有一对夫妻小红和小绿，那么我们可以先定义两个类——男人、女人，小红是类“女人”的一个实例，小绿是类“男人”的一个实例。之后我们可以定义小红和小绿之间的夫妻关系，这个关系就是对象属性“夫妻”，同时我们又知

道小红今年 30 岁，那么可以定义小红的一个数据属性 “年龄”，属性值是 “30”。注意我们的数据是存在两个地方的，一个是实体，一个是属性。基于 RDFS 这样一套对本体进行描述的语言体系，事实上是可以进行一定的推理的，如图6.2所示，由华为是人工智能公司和人工智能公司是高科技公司的子类，便可以推断出华为是高科技公司。

通过 RDFS 可以表示一些简单的语义，但在更复杂的场景下，RDFS 语义表达能力显得太弱，还缺少诸多常用的特征。比如：(1) 对于局部值域的属性定义：RDFS 中通过 rdfs:range 定义了属性的值域，值域是全局性的，无法说明该属性应用于某些具体的类时具有的特殊值域限制。(2) 类、属性、个体的等价性：RDF(S) 中无法声明两个或多个类、属性和个体是等价还是不等价。(3) 不相交类的定义：在 RDFS 中只能声明子类关系，如男人和女人都是人的子类，但无法声明这两个类是不相交的。(4) 基数约束：即对某属性值可能或必须的取值范围进行约束，如说明一个人有双亲（包括两个人），一门课至少有一名教师等。(5) 关于属性特性的描述：即声明属性的某些特性，如传递性、函数性、对称性，以及声明一个属性是另一个属性的逆属性等。故 W3C 提出了 OWL 语言扩展 RDFS，作为在语义网上表示本体的推荐语言。

OWL 简介

表 5.8: OWL 三种子语言简介

子语言	特征	使用限制举例
OWL Lite	用于提供给那些只需要一个分类层次和简单的属性约束的用户。	支持基数，但允许基数为 0 或 1。
OWL DL	在 OWL Lite 基础上包括了 0WL 语言的所有约束。该语言上的逻辑蕴涵是可判定的。	当一个类可以是多个类的一个子类时，它被约束不能是另外一个类的实例。
OWL Full	它允许在预定义的 (RDF、OWL) 词汇表上增加词汇，从而任何推理软件均不能支持 OWL FULL 的所有 feature。OWLFull 语言上的逻辑蕴涵通常是不可判定的。	一个类可以被同时表达为许多个体的一个集合以及这个集合中的一个个体。具有二阶逻辑特点。

OWL 指的是 web 本体语言, 是一项 Web 标准, 用于处理 Web 上的信息, 由 XML 来编写，被设计为供计算机进行解释，不是被设计为供人类进行阅读的，OWL 构建在 RDF 的顶端之上，拥有三种子语言，三种子语言如表6.3所示，OWL 弥补了 RDFS 的不足，添加了更多的用于描

述属性和类的词汇，例如类之间的不相交性 (disjointness)、基数 (cardinality)、等价性、属性的特征 (例如对称性)、属性的等价性以及枚举类 (enumerated classes) 等等。

OWL 词汇

(1) 声明等价性

exp: 演员 owl:equivalentClassexp: 明星，声明两个类是等价的；

exp: 推测 owl:equivalentPropertyexp: 猜测，声明两个属性是等价的；

exp: 演员 Aowl:sameIndividualAsexp: 张子枫，声明两个个体是等价的。

(2) 声明属性的局部约束：基数限定

exp:Person owl:cardinality “1” 8sd:integer；

exp:Person owl:onPropertyexp:hasMother；

exp:hasMother 在主语属于 exp:Person 类的时候，宾语的取值只能有一个；“1” 的数据类型被声明为 xsd:integer；这是基数约束，本质上属于属性的局部约束。

(3) 声明属性的传递性

exp:is_part_of rdf:typeowl:TransitiveProperty；

exp:is_part_of 是一个传递关系。比如：exp:Aexp:is_part_of exp:B；

exp:Bexp:is_part_of exp:C，那么根据上述声明，有 exp:Aexp:is_part_of exp: C。

(4) 声明两个属性互反

exp:parent owl:inverseOfexp:children；

exp:parent 和 exp:children 是互反关系。比如：exp: 小莉 exp:parent；exp: 小亮，那么根据上述声明，有 exp: 小亮 exp:children exp: 小莉。

(5) 声明属性的函数性

exp:hasFather rdf:typeowl:FunctionalProperty；

exp:hasMother 是一个具有函数性的属性，因为每个人只有一个父亲。

(6) 声明属性的对称性

exp:classmate 是一个具有对称性的属性。比如：exp: 小明 exp:classmate exp: 小丽，那么根据上述声明，有 exp: 小丽 exp:classmate exp: 小明。

(7) 声明属性的局部约束：全称限定

exp:Person owl:allValuesFromexp:Men；

exp:Person owl:onPropertyexp:hasFather；

exp: hasFather 在主语属于 exp: Person 类的时候，宾语的取值只能来自 exp: Men 这个类。

(8) 声明属性的局部约束：存在限定

exp:APaper owl:someValuesFromexp:EI；

exp:APaper owl:onPropertyexp:publishedIn；

exp:publishedIn 在主语属于 exp:APaper 类的时候，宾语的取值部分来自 exp:EI 这个类。上面的三元组相当于：A 论文部分发表在 EI 上。

(9) 声明相交的类

exp:Father owl:intersectionOf tmp；

tmp rdf:type rdfs:Collection；

tmp rdfs:member exp:Person;
tmp rdfs:member exp:HasChildren;
tmp 是临时资源;是 rdfs:Collection 类型,是一个容器,它的两个成员是 exp:Person,exp:HasChildren。上述三元组说明 exp:Father 是 exp:Person,exp:HasChildren 这两个类的交集。

(10) OWL 其他词汇见表6.4

表 5.9: OWL 中的其它词汇描述

OWL 中的其它词汇	描述
owl:oneOf	声明枚举类型
owl:disjointWith	声明两个类不相交
owl:unionOf	声明类的并运算
owl:minCardinality	最小的基数限定
owl:maxCardinality	最大的基数限定
owl:InverseFunctionalProperty	声明互反类具有函数属性
owl:hasValue	属性的局部约束时，声明所约束类必有一个取值

RDF、RDFS、OWL 描述的是关系的关系，赋予了关系的定义，相当于给了标准的标准，其实只有给定了实体之间和属性之间可能存在的关系的关系才能进行知识推理，因为没有约束的环境是混乱的。

SPARQL(SPARQL Protocol and RDF Query Language)

SPARQL 是一种 RDF 查询语言，可以对不同的数据集撰写复杂的连接 (joins)，主要包括数据的插入、删除和查询操作，被所有主流图数据库支持。以下是关于 SPARQL 语言的一些具体操作。

```
1 数据插入:例如添加ns:孔子ns:老师ns:老子; ns:老子ns:类型ns:哲学家; ns:老子ns:类
    型ns:史学家
prefix ns:<http://example.org/ns#>
INSERT DATA{
ns:孔子ns:老师ns:老子.
ns:老子ns:类型ns:哲学家;
ns:类型ns:史学家}
2 数据删除:例如从图中删除(ns:孔子ns:类型ns:教育家)
prefix ns:<http://example.org/ns#>
DELETE DATA{ns:孔子ns:类型ns:教育家.}
3 数据修改:例如修改(ns:小明ns:出生日期`“1903/03/21”`)中的`“1903/03/21”`为`“
    1902/03/21”`
prefix ns:<http://example.org/ns#>
DELETE DATA{ns:小明ns:出生日期`“1903/03/21”`.}
INSERTDATA{ns:费米ns:出生日期`“1902/03/21”`.}
```

```
4 查询语言关键字简介
4.1 OPTIONAL
#查询所有选修`A`课程的学生姓名，以及他们的邮箱，关键字`OPTIONAL`指示如果没有邮
    箱，那么依然返回学生姓名，邮箱处空缺。
SELECT ?student ?email
WHERE{?student exp:studies exp:A .
OPTIONAL{?student foaf:mbox ?email .}}
4.2 FILTER
#查询学生姓名，选修课程，以及他们的年龄，如果有年龄，那么年龄必须大于`20`岁。
SELECT ?module ?name?age
WHERE{?student exp:studies ?module .
      ?student foaf:name ?name .
OPTIONAL{?student exp:age ?age .
FILTER (?age $>$ 20) }}
4.3.UNION
#查询选修课程`A1`或`A2`的学生姓名以及邮件，注意，这里的邮件是必须返回的，如果没
    有邮件值，那么就不返回这条记录，注意和`OPTIONAL`区别。
SELECT ?student?email
WHERE{?student foaf:mbox ?email .
     {?student exp:studies exp:A1 }
UNION {?student exp:studies exp:A2 }}
4.4.FROM
#查询选修课程`A`的学生姓名以及邮件和住址，`FROM` 关键字引入了其它本体或者可访问
    的知识库。
SELECT ?student ?email ?home
FROM (http://www2.warwick.ac.uk/rdf/student)
WHERE{?student exp:studies exp:A .
OPTIONAL {?student foaf:mbox?email.
          ?student foaf:homepage ?home. }}
```

5.3.4 基于本体工具 (Protégé) 的知识建模实践

Protégé 简介

Protégé 软件是斯坦福大学医学院生物信息研究中心基于 Java 语言开发的本体编辑和本体开发工具，也是基于知识的编辑器，属于开放源代码软件，实现了 OWL 功能，它主要用于语义网中本体的构建，是语义网中本体构建的核心开发工具，目前最新版本是 5.5.0（截止到 2021-08-13），Protégé 提供了本体概念类，关系，属性和实例的构建，并且屏蔽了具体的本体描述语言，用户只需在概念层次上进行领域本体模型的构建。

Protégé 特点

Protégé 是一组自由开源的工具软件，用于构建域模型与基于知识的本体化应用程序；提供了大量的知识模型架构与动作，用于创建、可视化、操纵各种表现形式的本体；可以通过用户定制实现域——友好（领域相关）的支持，用于创建知识模型并填充数据；Protégé 可以通过两种

方式进行扩展：插件和基于 java 的 API；相比其他的本体构建工具，Protégé 最大的好处在于支持中文，在插件上，用 Graphviz 可实现中文关系的显示。

Protégé 用途

(1) 类建模 (Class modeling)：Protégé 提供了一个图形化用户界面来类建模（领域概念）和它们的属性及关系。

(2) 实例编辑 (Instance editing)：从这些类中，Protégé 自动产生交互式的形式，全用户或领域专家进行有效实例编辑成为可能。

(3) 模型处理 (Model processing)：Protégé 有一些插件库，可以定义语义、解答询问以及定义逻辑行为。

(4) 模型交换 (Model exchange)：最终的模型（类和实例）能以各种各样的格式被装载和保存，包括 XML、UML 和资源描述框架 RDF。

Protégé 的具体使用步骤

1. 首先下载安装 Protégé，由于 Protégé 基于 Java，所以要先安装配置 Java 环境，Java 从官网下载即可。

2.protege 的具体使用：

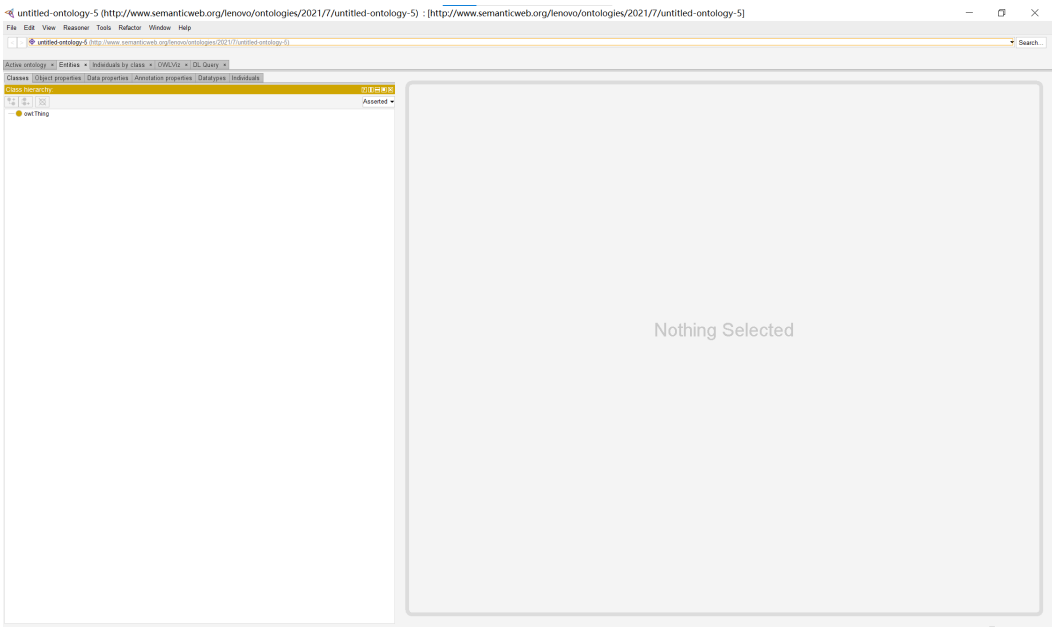


图 5.11

(1) 首先打开界面如图6.3所示，在建立本体时，只需用到 Entities 选项下的一些按钮。

(2) 构建类：其中 classes 是用于构建类，在 classes 页面，右键点击 owl:Thing ，选择 add subclasses，在出现的对话框中的 name 输入类的名字，点击确定。此外，还可以右键点击一个类，创建它的子类。点击左侧的 Thing，这个是所有类的最高级的父类，我们建立的所有类都是

它的子孙，这时候 Thing 上面的三个小图标中的第一个就会亮起，它是创建子类的快捷按钮，第二个是创建兄弟类，第三个是删除选中的类。

(3) 构建类之间的关系：点击 classes 页面中的某一个类，在右面的 describtion 可以观察和修改这个类的属性，比如它的父类是什么，它与什么互斥，我们可以利用这个功能设置类之间的关系。比如：动物和植物是完全不同的事物，它们之间具有排他性（owl:DishointWith）。

(4) 建立对象属性：在 Object Properties 页面下完成，这一步是最重要的一环，因为这是三元组中关于“关系”的设置，这一步的基本操作和构建类是一样的，除此之外，对象属性还要传递性、函数性、对称性等特征，这可以在右侧的 Characteristics 中点击左侧的小方框即可。

(5) 建立数据属性：实体之间不仅有关系相连接，还有属于自己的一些属性，比如：对于禅师来说，会有法号，法号必须是 string 类型。切换到 “Data Properties”，我们在该界面创建类的属性，即数据属性。其定义方法和对象属性类似。

5.4 知识抽取——实体识别

通过知识抽取技术，可以从一些公开的半结构化、非结构化和第三方结构化数据库的数据中提取出实体、关系、属性等知识要素，并形成结构化数据输出的文本处理技术，包括：实体识别与抽取、实体消歧、事件抽取和关系抽取，我们主要介绍实体识别。

5.4.1 命名实体的定义

狭义地讲，命名实体指现实世界中具体或抽象的实体，如人（张三）、机构（中国中文信息学会、阿里巴巴网络技术有限公司）、地点等，通常用唯一的标志符（专有名称）表示，如人名、机构名、地名等；广义地讲，命名实体还可以包含时间（12：00）、日期（2017 年 10 月 17 日）、数量表达式（100）、金钱（一亿美金）等。至于命名实体的确切含义，只能根据具体应用来确定。比如，在具体应用中，可能需要把住址、电子信箱地址、电话号码、舰船编号、会议名称等命名实体。

5.4.2 命名实体识别的任务

一般而言，主要是识别出待处理文本中七类（人名、机构名、地名、时间、日期、货币和百分比）命名实体。两个任务：实体边界识别和实体类别标注 (Entity Typing)，实体边界识别即识别出从哪儿开始为某个实体的第一个字，这个实体到哪个字结束，实体类别标注即把识别出来的实体做个标记，让我们清晰的知道它是实体。

5.4.3 命名实体识别的特点

时间、日期、货币和百分比的构成有比较明显的规律，识别起来相对容易。而人名、地名、机构名等识别的难度很大，主要体现在以下三方面 (1) 数量巨大，不能枚举，难以全部收录在词典中；(2) 某些类型的实体名称用字灵活，表达形式多样，而且没有严格的规律可以遵循；(3) 首次出现后往往采用缩写形式。

5.4.4 命名实体识别的方法

(1) 基于词典的命名实体识别方法

方法概述：按照一定的策略将待分析的汉字串与一个充分大的词典中的词条进行匹配，若在词典中找到某个字符串，则匹配成功。典型方法有正向最大匹配法、反向最大匹配法和最短路径法（最少分词法）。

(2) 无词典的命名实体识别方法

在分词的过程中使用词典的方法是有词典切分，反之是无词典切分，有词典切分的方法一般是基于规则的，无词典切分的方法一般是基于统计的。

(3) 基于规则的方法

基于规则的方法不需要标注训练语料，能够直接根据词典和规则进行分词。

(4) 基于统计的方法

基于统计的方法需要标注训练语料来训练模型。另外基于统计的方法可以分为生成式统计命名实体识别和判别式命名实体识别。生成式方法的原理是首先建立学习样本的生成模型，再利用模型对预测结果进行间接推理，典型算法为 HMM 等；判别式方法的原理是由字构词的命名实体识别理念，将 NER 问题转化为判别式分类问题（序列标注问题），典型算法有 Maxent 、SVM 、CRF 、CNN 、RNN、LSTM+CRF。

注意在知识图谱中不需要用以上提到的算法，利用 lstlisting 中的 spacy 包就可以完成命名实体识别的任务

5.4.5 相关代码

(1) 基本介绍

```
#首先导入spacy
import spacy
#接下来我们需要加载一个`NLP`模型对象，使用`spacy.load()`函数
#此模型需要一个参数，我们使用小型英文预训练模型
nlp = spacy.load("en_core_web_sm")
#`nlp`对象创建后，我们就可以用它来解析文本了.
#使用`nlp`()得到`doc`对象.
#`doc`包含了文本中的诸多数据.
text = "Martin J. Thompson is known for his writing skills. He is also good at
    programming."
doc = nlp(text)
#显示doc对象:
print (doc)
#虽然这看起来和上面的text字符串没什么不同，其实却完全不同.
#我们用分句器来展示这样不同。
for sent in doc.sents:
    print (sent)
#以下内容会对这两部分进行详细说明
#命名实体识别
#for ent in doc.ents:
```

```
    #print(ent.text, ent.label_)
#词性标注
#for token in doc:
    #print(token.text, token.pos_)
```

输出结果为：

```
Martin J. Thompson is known for his writing skills.
He is also good at programming.
```

图 5.12

我们使用 spacy 分句器生成所需的输出：将文本正确地分解成句子。

(2) 中文分词断句

```
#首先导入spacy和zh_core_web_sm
import spacy
import zh_core_web_sm
#加载一个`NLP`模型对象
spacy_nlp = zh_core_web_sm.load()
#`novel.txt`里面保存了一段中文文字
f = open('novel.txt', encoding='GBK')
data = f.read()
print('原文：\n', data)
print()
f.close()
doc = spacy_nlp(data)
# 分词
print('分词：')
for token in doc:
    print(token.text, end='|')
print()
# 断句
print('断句：')
for s in doc.sents:
    print(s)
#注意不能同时加载中文和英文训练模型
```

输出结果为：

```
原文：
 毛泽东，字润之，笔名子任，湖南湘潭人。中国人民的领袖，伟大的马克思主义者，无产阶级革命家、战略家和理论家。

分词：
毛泽东|，|字润之|，|笔名|子任|，|湖南|湘潭人|。|中国|人民|的|领袖|，|伟大|的|马克思主义者|，|无产阶级|革命家|、|战略家|和|理论家|。
断句：
毛泽东，字润之，笔名子任，湖南湘潭人。
中国人民的领袖，伟大的马克思主义者，无产阶级革命家、战略家和理论家。
```

图 5.13

(3) 英文分词断句

```
#首先导入spacy和en_core_web_sm
import spacy
import en_core_web_sm
#加载一个`NLP`模型对象
spacy_nlp = en_core_web_sm.load()
s = "Next week I'll be in Shanghai."
print('英文原文: ', s)
doc = spacy_nlp(s)
#切词
print('切词: ')
for t in doc:
    print(t.text, end=' | ')
print()
```

输出结果为:

```
英文原文:  Next week I'll be in Shanghai.
切词:
Next | week | I | 'll | be | in | Shanghai | .
```

图 5.14

(4) 词性标注

```
import spacy
import pandas as pd
#设置指定选项的值
#无论列数多少, 最后显示的是`1000`行
pd.set_option('display.max_columns', 1000)
pd.set_option('display.max_rows', 1000)
#每一行列中显示出来的数据宽度
pd.set_option('display.width', 1000)
#nlp = spacy.load('zh_core_web_sm')
nlp = spacy.load('en_core_web_sm')
# 每个`token`对象有着非常丰富的属性, 如下的方式可以取出其中的部分属性。
doc = nlp("Next week I'll be in Shanghai.")
for token in doc:
    print("{0}\t{1}\t{2}\t{3}\t{4}\t{5}\t{6}\t{7}\t{8}\t{9}".format(
        token.text,      #单词
        token.idx,       #单词起始索引
        token.lemma_,    #单词的原型
        token.is_punct, #是否为标点
        token.is_space, #是否为空格
        token.shape_,    #形式 Xxx(如第一个字母大写, 其余小写)
```

```
        token.pos_,      #词性标注
        token.tag_,      #标签
        token.is_stop,   #是否为停用词
        token.dep_       #标记单词是主语，谓语，宾语还是连接词
    ))
print()
#以下是一篇英文文章的词性标注
# col_names = ['词', '起始位置', '词形还原', '是否为标点', '是否为空格', '形式',
    '词性', 'TAG', '是否为停用词', '依存关系']
# result = pd.DataFrame(columns=col_names)
# f = open('novel.txt', encoding='GBK')
# data = f.read()
# f.close()
# doc = nlp(data)
# for i, token in enumerate(doc):
#     result.loc[i] = [token.text, token.idx, token.lemma_, token.is_punct,
    token.is_space, token.shape_, token.pos_, token.tag_, token.is_stop, token.
    dep_]
# result.to_excel('result.xls', encoding='utf-8')
# print(result)
```

```
Next    0   next    False   False   Xxxx    ADJ JJ  True    amod
week    5   week    False   False   xxxx    NOUN    NN  False   npadvmod
I   10  I   False   False   X   PRON    PRP True    nsubj
'll 11  'll False   False   'xx AUX MD  True    aux
be  15  be  False   False   xx  VERB    VB  True    ROOT
in  18  in  False   False   xx  ADP IN  True    prep
Shanghai    21  Shanghai    False   False   Xxxxx   PROPN   NNP False   pobj
.   29  .   True    False   .   PUNCT   .   False   punct
```

图 5.15

例如：输出结果中的第一行表示 Next 的起始索引是 0，原型是 next，不是标点，不是空格，形式为 Xxxx，是形容词，标签是 JJ，是停用词，是连接词。

(5) 组块分析

```
#加载spacy和pandas
import spacy
import pandas as pd
nlp = spacy.load('en_core_web_sm')  #加载小型英文训练模型
#对括号中的内容进行nlp处理
doc = nlp("Wall Street Journal just published an interesting piece on crypto
    currencies")
#组块分析
for chunk in doc.noun_chunks:
    print(chunk.text, '|', chunk.label_, '|', chunk.root.text)
#以下是对一篇较长文章的组块分析
```

```
#输出结果的行名
col_names = ['词', '标签', '词根']
result = pd.DataFrame(columns=col_names)
#打开`Harry Potter.txt`文件
f = open('Harry Potter.txt')
data = f.read()
f.close()
#对文章内容进行`nlp`处理
doc = nlp(data)
#组块分析
for i, chunk in enumerate(doc.noun_chunks):
    result.loc[i] = [chunk.text, chunk.label_, chunk.root.text]
print(result)
```

```
Wall Street Journal | NP | Journal
an interesting piece | NP | piece
crypto currencies | NP | currencies
                  词  标签         词根
0                Mr  NP         Mr
1       Mrs Dursley  NP    Dursley
2            number  NP     number
3    , Privet Drive  NP      Drive
4              they  NP       they
5               you  NP        you
6              They  NP       They
7   the last people  NP     people
8               you  NP        you
9          anything  NP  anything
10             they  NP       they
11    such nonsense  NP  nonsense
```

图 5.16

从图中可以看出，输出结果是进行组块分析的，例如 the last people 是在一起的，并没有分开，这个块的标签是 NP，词根是 people。

(6) 命名实体识别

```
#加载spacy和displacy
import spacy
from spacy import displacy
#加载小型中文训练模型
nlp = spacy.load('zh_core_web_sm')
#给data赋值
data = '`2011年4月11日17点16分`，`日本东北部的福岛和茨城地区发生里氏7.0级强烈地
    震`（`震中北纬36.9 度`、`东经140.7度`，`即福岛西南30公里左右的地方`，`震源深
    度10 公里`，属于浅层地震）`当局已经发布海啸预警震后约30分钟后在日本海地区发生
```

```
    巨型海啸`，同时造成福岛核电站出现核泄漏震后第十天，国际原子能机构对于日本政府
    反应迟钝进行了谴责'
#对data进行nlp处理
doc = nlp(data)
result = set()
#实体识别
for ent in doc.ents:
    print(ent.text, ent.label_)
    if ent.label_ == 'PERSON':
        result.add(ent.text)
print(len(result), result)
# PERSON 人名;
# NORP是国籍或宗教团体;
# GPE标识位置（城市、国家等等）;
# DATE 标识特定的日期或日期范围,
# ORDINAL标识一个表示某种类型的顺序的单词或数字
```

```
2011年4月11日17点16分 DATE
日本东北部 ORG
茨城地区 LOC
北纬 LOC
9 度 QUANTITY
7度 QUANTITY
30公里 QUANTITY
10公里 QUANTITY
30分 DATE
日本海地区 LOC
第 十天 DATE
日本 GPE
0 set()
```

图 5.17

输出结果为如图6.9，从结果中可以得知 2011 年 4 月 11 日 17 点 16 分为 DATE，日本为 GPE。

(7) 句法依存分析

```
#加载spacy和displacy
import spacy
from spacy import displacy
#加载小型中文训练模型
nlp = spacy.load('zh_core_web_sm')
#对括号中的句子进行nlp处理
doc = nlp('1930年，胡🄴青正在北京师范大学念书。')
#对doc进行句法依存分析
html = displacy.render(doc, style='dep', jupyter=False, options={'distance':
    80})
```

```
f = open('result.html', mode='w', encoding='ansi')
f.write(html)
f.close()
print(html)
```

输出结果如图6.10

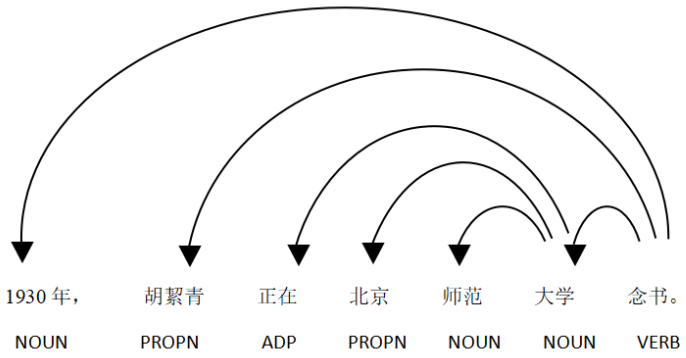


图 5.18

5.5 知识推理

5.5.1 知识推理的定义与任务

推理是逻辑学、哲学、心理学、人工智能等学科中的重要概念，早在古希腊时期，著名哲学家亚里斯多德就提出三段论作为现代演绎推理的基础，在计算机科学及人工智能领域，推理是一个按照某种策略从已知事实出发去推出结论的过程，知识推理亦是如此。

结构化知识图谱的一大优势就是能够支撑高效的推理，典型的推理任务包括知识库补全和知识库问答。知识库补全也称知识图谱上的链接预测，其本质是根据知识库中已有的知识推断出新的、未知的知识，知识库补全可以用来建立更全面的知识库，是知识库构建的重要手段之一；知识库问答即基于知识库的问答主要是通过对自然语句的解析，再从知识库中寻找答案的过程，但由于知识库中知识不完备等原因，知识库问答也需要推理技术的支撑。

5.5.2 知识推理的分类

归纳推理和演绎推理

归纳推理

归纳推理是根据部分对象所具有的性质，推出一类事物中所有对象都具有这类性质的推理方式，可分为三个步骤：(1) 对部分资料进行观察、分析和归纳整理；(2) 得出规律性的结论，即猜想；(3) 检验猜想。例如：我们看到第一只天鹅是白色的，第二只天鹅也是白色的，第三只天鹅也是白色的，看了几千几万只天鹅，发现这些天鹅都是白色的，所以我们下了一个结论，认为

所有的天鹅都是白色的。归纳推理是从特殊到一般的过程，即使推理的前提是正确的，也不能保证推理的结论是正确的，但是有些时候是可能得到结论的。

演绎推理

演绎推理是从一般性的前提出发，通过推导即“演绎”，得出具体陈述或个别结论的过程。亚里士多德提出的著名三段论：(1) 一个一般性的原则（大前提）；(2) 大前提的特殊化陈述（小前提）；(3) 引申出的特殊化陈述符合一般性原则的结论。例如：老马是认识道路的（大前提），这几匹马是老马（小前提），可以演绎出这几匹马是认识道路的（结论）。演绎推理是从一般到特殊的过程，只要演绎的前提是正确的，一般来讲，演绎的结果是不会出错的。但演绎推理不仅仅局限于三段论，也不只是从一般到特殊的过程，它有着强烈的“演绎”特性，重在通过利用每一个证据，逐步地推导到目标或意外的结论。例如：A 盒子说“礼物不在此盒中”，B 盒子说“礼物在 C 盒中”，C 盒子说“礼物不在此盒中”，上述三句话中，最多只有一句话是真的，这个推理便可以用演绎法逐步地进行。

确定性推理与不确定性推理

确定性推理

确定性推理大多指确定性逻辑推理，它具有完备的推理过程和充分的表达能力，可以严格地按照专家预先定义好的规则准确地推导出最终结论。但是确定性推理很难应对真实世界中，尤其是存在于网络大规模知识图谱中的不确定甚至不正确的事实和知识，例如“一个人和其父亲拥有同样的国籍”这条规则在现实生活中大部分情况下都是正确的，但也不排除移民、母方国籍等因素使得少量事实不满足（不确定的知识）；大规模知识图谱 YAGO 根据抽样统计宣布其中含有 5% 左右的错误事实（不正确的事实）。

不确定性推理

不确定性推理也被称为概率推理，是统计机器学习中一个重要的议题。它并不是严格地按照规则进行推理，而是根据以往的经验和分析，结合专家先验知识构建概率模型，并利用统计计数、最大化后验概率等统计学习的手段对推理假设进行验证或推测。不确定性推理可以有效建模真实世界中的不确定性。

符号推理与数值推理

符号推理

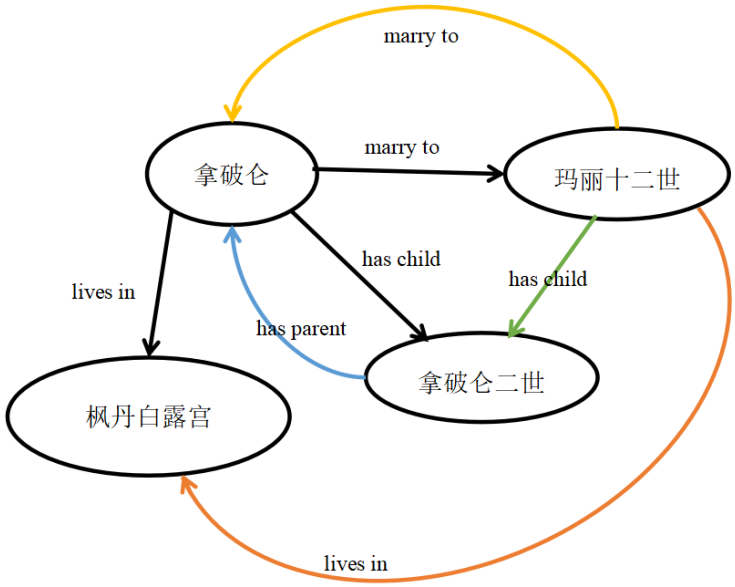


图 5.19

符号推理又可以看成是传统的逻辑推理，它的特点就是在知识图谱中的实体和关系符号上直接进行推理操作。符号推理的学习推理规则是从现实的大规模网络知识图谱中自动地总结出逻辑规则，这一任务也被称为逻辑规则挖掘。例如 ∀ x,y CapitalOf(x,y)⟹LocatedIn(x,y)，便是一个谓词逻辑规则挖掘的实例，其中，谓词是 CapitalOf，LocatedIn；个体变量是 x，y，逻辑蕴含 ⟹ 表示 “若 . . . ，则 . . . ” 的语义；CapitalOf(x,y) 表示该规则的前提；LocatedIn(x,y) 表示该规则的结论。若我们有了以下推理规则：

hasChild(A,B)⟹hasParent(B,A)，

marryTo(A,B)⟹marryTo(B,A)，

marryTo(A,B)∩hasChildren(A,C)⟹hasChild(B,C)，

marryTo(A,B)∩livesIn(A,C)⟹livesIn(B,C)

我们从图6.11便能推导出玛丽十二世住在枫丹白露宫的概率极大。

符号演算中比较特殊的一类挖掘是频繁子图挖掘，频繁子图挖掘是搜集知识图谱的规则实例，它将规则实例中的实体替换成变量，同时设定一些约束，以快速地确定规则的实用性，并根据知识图谱中规则实例的支撑来快速评价挖掘到的规则。例如若知识图谱中存在以下这种大量的规则实例，我们便能得到这样一条规则：一个人的父亲的父亲是这个人的爷爷，用符号表示是：

父亲（x，y）∩ 父亲（y，z）⟹ 爷爷（x，z）

父亲（向芷, 向佐）∩ 父亲（向佐, 向华强）⟹ 爷爷（向芷, 向华强）

父亲（姚沁蕾, 姚明）∩ 父亲（姚明, 姚志源）⟹ 爷爷（姚沁蕾, 姚志源）

父亲（霍中曦, 霍启刚）∩ 父亲（霍启刚, 霍震霆）⟹ 爷爷（霍中曦, 霍震霆）

父亲（步步, 吴奇隆）∩ 父亲（吴奇隆, 吴飞元）⟹ 爷爷（步步, 吴飞元）

数值推理

与符号推理相对应的就是数值推理，数值推理就是使用数值，尤其是向量矩阵等数值计算的方法，捕捉知识图谱上隐式的关联，模拟推理的过程，基于分布式知识表示的推理就是典型的数值推理方法。数值推理主要用于捕捉实体和关系之间的隐式关联，相对于符号推理，数值推理可以直接使用数据参与运算且计算速度非常快。

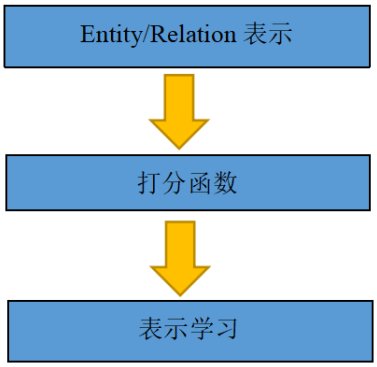


图 5.20

分布式知识表示 (Knowledge Graph Embedding) 的核心思想是将符号化的实体和关系在低维连续向量空间进行表示，在简化计算的同时最大程度保留原始的图结构，分布式空间映射对特征间的复杂关系进行了解耦，减少了维数灾难问题 (curse of dimensionality), 基本步骤：

(1) 实体关系表示：定义实体和关系在向量空间中的表示形式。

(2) 打分函数定义：定义打分函数，衡量每个三元组成立的可能性。

(3) 表示学习：构造优化问题，学习实体和关系的低维连续向量表示。

基于分布式知识表示的推理主要是知识图谱上的链接预测即预测一个三元组的头实体或者尾实体。一个正确的三元组，移除它的头实体或者尾实体，检验模型能否预测出正确的头实体或者尾实体。可用于知识补全时的检验，在补全知识图谱缺失关系时，可以选一条边连接任意两个实体，构成新的三元组，再判断该三元组是否正确。分布式知识表示方法可以分为翻译模型 (translationmodels) 和匹配模型 (semanticmatching models)，翻译模型是采用基于距离的打分函数来衡量三元组成立的可能性，语义匹配模型是采用基于相似度的打分函数来衡量三元组成立的可能性。

5.5.3 频繁项集的一个案例

```
# -*- coding: utf-8 -*-
# 分析某导演的电影的 频繁项集和关联规则
from efficient_apriori import apriori
import csv
```

```
director = u'宁浩'
file_name = './' + director + '.csv'
lists = csv.reader(open(file_name, 'r', encoding='utf-8-sig'))
# 数据加载
data = []
for names in lists:
        name_new = []
        for name in names:
                # 去掉演员数据中的空格
                name_new.append(name.strip())
        data.append(name_new[1:])
# 挖掘频繁项集和关联规则
itemsets, rules = apriori(data, min_support=0.4, min_confidence=0.3)
print('频繁项集：', itemsets)
print('关联规则：', rules)
```

输出结果为：

```
频繁项集： {1: {('黄渤',): 7, ('徐峥',): 6}, 2: {('徐峥', '黄渤'): 6}}
关联规则： [{黄渤} -> {徐峥}, {徐峥} -> {黄渤}]
```

图 5.21

从结果中可以得知在宁浩导演的电影中一共有两个频繁项集，第一个是黄渤单独出现了 7 次，徐峥单独出现了 6 次，第二个是黄渤和徐峥共同出现了 6 次，故能得到如下关联规则：在黄渤出现的情况下，徐峥大概率会出现，反之亦然。

第 6 章　知识图谱（二）

6.1　知识表示与建模

6.1.1　知识表示

知识表示可以从以下四个角度理解：

(1) 知识表示是一种代理，基于对事物的表示，我们无需实践而是通过思考和推理就可以得到有关外部世界的结论。

(2) 知识表示是一组本体论约定的集合，说明我们以什么样的方式来思考世界。

(3) 知识表示是智能推理的组成部分，推理需要对知识进行表示，但知识表示不是推理的全部。知识表示是高效计算的媒介，通过对知识进行有效组织，支持高效的推理。

(4) 知识表示是人类表达的媒介，基于通用表示框架，方便人们表达和分享对世界的认知。

6.1.2　AI 早期知识表示方法

一阶谓词逻辑 (First-Order-Logic)

一阶谓词逻辑是最早出现的一种形式语言表示形式，是一种形式系统 (Formal System)，即形式符号推理系统，也叫一阶谓词演算、低阶谓词演算 (Predicate Calculus)、限量词 (Quantifier) 理论，也有人称其为 “谓词逻辑”。一阶谓词逻辑是原子命题的下一层级，可以看成是对原子命题做进一步分析，分析出其中的个体词、谓词、量词，研究它们的形式结构的逻辑关系、正确的推理形式和规则。比如，穷人不可能是皇帝，某一个人是皇帝，推导出这个人不是穷人类似这种知识推理的方案。有了一阶谓词逻辑，可以把它经过联立，形成所谓的高阶谓词逻辑，就类似线性代数再加上激活函数能得到所谓的非线性代数，所以没有高阶谓词逻辑这个说法，一阶谓词逻辑足够使用了。

产生式规则 (Production Rule)

产生式系统是一种更广泛的规则系统，和一阶谓词逻辑有关联，也有区别。产生式规则在一阶谓词逻辑表示的基础上，进一步解决了不确定性知识的表示，产生式规则以三元组 (对象，属性，值) 或者 (关系，对象 1，对象 2)，通过进一步加入置信度形成四元组 (对象，属性，值，置信度) 或者 (关系，对象 1，对象 2，置信度) 的形式来表示事实，并使用 P→Q 或者 IF P THEN

Q 的形式用于表示规则，这种表示方法可以表示不确定性知识和过程性知识，具有一致性和模块化等优点，通过规则可以实现推理功能，广泛运用于上世纪 70 年代的专家系统当中。

例如，如果已知一个图形有三边，且这三条边相等，那么这个图形是等边三角形便是用产生式规则得到的推理结果。基本形式：如事实“老李年龄是 35 岁”，便写成（Lee,age,35）；事实“老李、老张是朋友”，可写成（friend,Lee,Zhang）。常用结构：原因 → 结果：天下雨，地上湿；条件 → 结论：将冰加热到 0 度以上，冰会融化成水；前提 → 操作：如果能找到合适的杠杆和支点，则可以翘起地球；事实 → 进展：夜来风雨声，花落知多少；情况 → 行为：手机开机了，则意味着可以收到别人发我的信息了。

框架 (Framework)

框架表示法是明斯基于 1975 年提出来的，其最突出的特点是善于表示结构性知识，能够把知识的内部结构关系以及知识之间的特殊关系表示出来，并把与某个实体或实体集的相关特性都集中在一起，而一阶谓词逻辑和产生式规则只能一条一条的表示知识，即使这些知识是对同一个实体而言。框架表示法认为人们对现实世界中各种事物的认识都是以一种类似于框架的结构存储在记忆中的。当面临一个新事物时，就从记忆中找出一个合适的框架，并根据实际情况对其细节加以修改、补充，从而形成对当前事物的认识。

框架是描述对象 (一个事物、一个事件、一个概念) 属性的一种数据结构，在框架表示法中，框架被认为是知识表示的最基本单元。框架是由若干结点和关系 (统称为槽 slot) 构成的网络，是语义网络一般化形式化的一种结构，同语义网络没有本质区别，将语义网络中结点间弧上的标注也放入槽内就成了框架表示法，框架基本组成如表6.1展示，表6.2是关于洪水的框架实例。

框架表示法的优点：对于知识的描述非常完整和全面、基于框架的知识库质量非常高、允许数值计算。缺点：构建成本非常高、对知识库的质量要求非常高、框架的表达形式不灵活，很难同其它形式的数据集相互关联使用。

表 6.1: 框架基本组成

< 框架名 >		
槽名 1:	侧面名 1	值 1, 值 2,..., 值 p1
	侧面名 2	值 1, 值 2,..., 值 p2
	...	...
	侧面名 m1	值 1, 值 2,..., 值 pm1
...	...	...
槽名 n:	侧面名 1	值 1, 值 2,..., 值 r1
约束:	约束条件 1	
	...	
	约束条件 n	

表 6.2: 河南洪水

洪水实例：河南洪水		
槽 1:	< 时间 >:	2021 年 7 月 17 日至 8 月 2 日
槽 2:	< 地点 >:	河南
槽 3:	< 伤亡 >	
	侧面 3.1:	< 死亡人数 >:302 人
	侧面 3.2:	< 失踪人数 >:50 人
	侧面 3.2:	< 受灾人数 >:1481.4 万人
槽 4:	< 损失 >	
	侧面 4.1:	< 直接经济损失 >:1337.15 亿元
槽 5:	< 震级 >:	8 级

语义网络 (Semantic Network)

语义网络 (semantic network) 是一种以网络格式表达人类知识构造的形式，是人工智能程序运用的表示方式之一，它是由节点和连接节点之间的弧组成。语义网络中的节点表示各种事物、概念、情况、属性、动作、状态等，每个节点可以带有若干属性，一般用框架或元组表示，此外，节点还可以是一个语义子网络，形成一个多层次的嵌套结构。语义网络中的弧表示各种语义联系，指明它所连接的节点间某种语义关系。节点和弧都必须带有标识，以便区分各种不同对象以及对象间各种不同的语义联系。最简单的语义网络是一个三元组：(节点 1，关系，节点 2)。

例子“每个老师都教过一个学生”用语义网络表示：

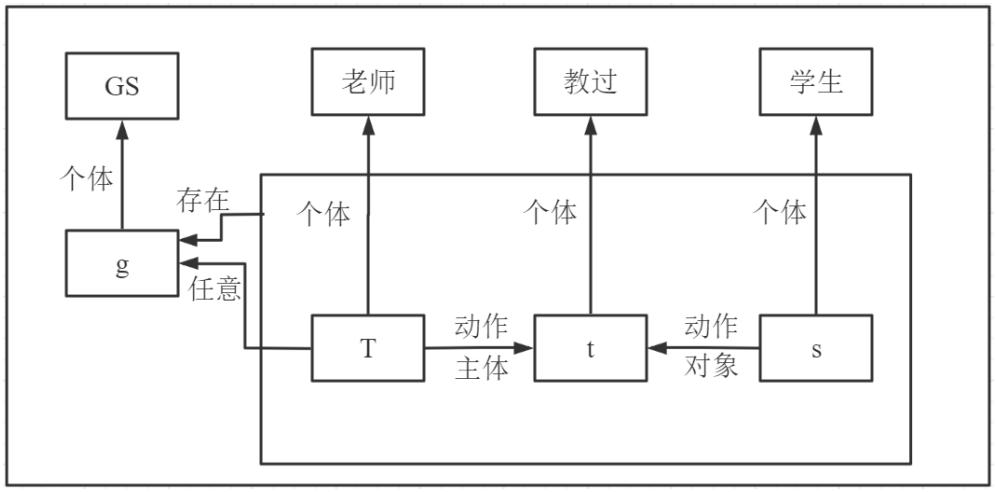


图 6.1

语义网络的优点：把各个节点之间的联系以明确、简洁的方式表示出来，是一种直观的表示方法；着重强调事物间的语义联系，体现了人类思维的联想过程，符合人们表达事物间的关系，因此把自然语言转换成语义网络较为容易；具有广泛的表示范围和强大的表示能力，用其他形式

的表示方法能表达的知识几乎都可以用语义网络来表示；把事物的属性以及事物间的各种语义联系显示地表示出来，是一种结构化的知识表示法。

缺点：推理规则不十分明了，不能充分保证网络操作所得推论的严格性和有效性；一旦节点个数太多，网络结构复杂，推理就难以进行；不便于表达判断性知识与深层知识。

6.1.3 基于语义网的知识表示框架

RDF(Resource Description Framework)

RDF(ResourceDescriptionFramework) 是一种资源描述框架, 其中 Resource 指页面、图片、视频等任何具有 URI 标识符的事物, 都可统称为资源;Description 表示属性、特征和资源之间的关系；Framework 是模型、语言和描述的语法。

RDF 假定任何复杂的语义都可以通过若干个三元组的组合来表达，并定义这种三元组的形式为“对象一属性一值”或“对象一对象一关系”，其中需要公开或通用的资源，都会绑定一个可识别的通用资源表示符 (URI)。而这两种表示形式都可以看成是主、谓、宾这种结构，即每一份知识可以被分解为如下形式：
(subject (主) ,predicate (谓), object (宾))。

RDF Schema(RDFS) 简介

RDFS 可以看成是 RDF 的扩展，RDFS 在 RDF 基础上提供了一个术语、概念等的定义方式，以及哪些属性可以应用到哪些对象上，主要关注类别和属性的层次结构以及继承关系等。即在 RDF 的基础上定义了如下词汇：Class(类),subClassOf(子类),type(类型),Property(属性),subPropertyOf(子属性),Domain,Range。例如“人”是一个类，而具体的实体例如“小红”便是类“人”的子类，即类之间有继承性，而类型指的是属性值的类型，属性是指类具有的属性，例如人具有身高、体重等属性，属性之间也存在继承关系，Domain 指属性的主语的范围，Range 指属性的宾语的范围，注意这里的属性指的都是对象属性。属性可分为对象属性和数据属性，对象属性和数据属性的定义可以这么理解：假如有一对夫妻小红和小绿，那么我们可以先定义两个类——男人、女人，小红是类“女人”的一个实例，小绿是类“男人”的一个实例。之后我们可以定义小红和小绿之间的夫妻关系，这个关系就是对象属性“夫妻”，同时我们又知道小红今年 30 岁，那么可以定义小红的一个数据属性“年龄”，属性值是“30”。注意我们的数据是存在两个地方的，一个是实体，一个是属性。基于 RDFS 这样一套对本体进行描述的语言体系，事实上是可以进行一定的推理的，如图6.2所示，由华为是人工智能公司和人工智能公司是高科技公司的子类，便可以推断出华为是高科技公司。

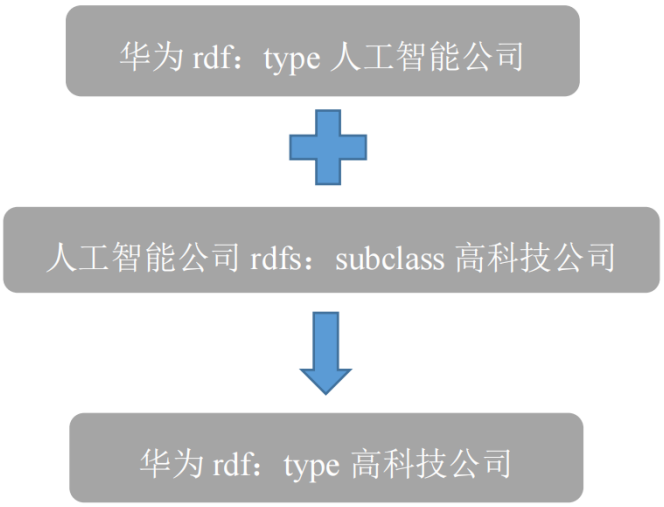


图 6.2

通过 RDFS 可以表示一些简单的语义，但在更复杂的场景下，RDFS 语义表达能力显得太弱，还缺少诸多常用的特征。比如：(1) 对于局部值域的属性定义：RDFS 中通过 rdfs:range 定义了属性的值域，值域是全局性的，无法说明该属性应用于某些具体的类时具有的特殊值域限制。(2) 类、属性、个体的等价性：RDF(S) 中无法声明两个或多个类、属性和个体是等价还是不等价。(3) 不相交类的定义：在 RDFS 中只能声明子类关系，如男人和女人都是人的子类，但无法声明这两个类是不相交的。(4) 基数约束：即对某属性值可能或必须的取值范围进行约束，如说明一个人有双亲（包括两个人），一门课至少有一名教师等。(5) 关于属性特性的描述：即声明属性的某些特性，如传递性、函数性、对称性，以及声明一个属性是另一个属性的逆属性等。故 W3C 提出了 OWL 语言扩展 RDFS，作为在语义网上表示本体的推荐语言。

OWL 简介

OWL 指的是 web 本体语言，是一项 web 标准，用于处理 web 上的信息，由 XML 来编写，被设计为供计算机进行解释，不是被设计为供人类进行阅读的，OWL 构建在 RDF 的顶端之上，拥有三种子语言，三种子语言如表6.3所示，OWL 弥补了 RDFS 的不足，添加了更多的用于描述属性和类的词汇，例如类之间的不相交性 (disjointness)、基数 (cardinality，如刚好一个)、等价性、属性的特征 (例如对称性)、属性的等价性以及枚举类 (enumerated classes) 等等。

OWL 词汇

(1) 声明等价性

exp: 演员 owl:equivalentClassexp: 明星，声明两个类是等价的。

exp: 推测 owl:equivalentPropertyexp: 猜测，声明两个属性是等价的

exp: 演员 Aowl:sameIndividualAsexp: 张子枫，声明两个个体是等价的

(2) 声明属性的局部约束：基数限定

exp:Person owl:cardinality “1” 8sd:integer

表 6.3: OWL 三种子语言简介

子语言	特征	使用限制举例
OWL Lite	用于提供给那些只需要一个分类层次和简单的属性约束的用户。	支持基数，但允许基数为 0 或 1。
OWL DL	在 OWL Lite 基础上包括了 0WL 语言的所有约束。该语言上的逻辑蕴涵是可判定的。	当一个类可以是多个类的一个子类时，它被约束不能是另外一个类的实例。
OWL Full	它允许在预定义的 (RDF、OWL) 词汇表上增加词汇，从而任何推理软件均不能支持 OWL FULL 的所有 feature。OWLFull 语言上的逻辑蕴涵通常是不可判定的。	一个类可以被同时表达为许多个体的一一个集合以及这个集合中的一个个体。具有二阶逻辑特点。

exp:Person owl:onPropertyexp:hasMother

exp:hasMother 在主语属于 exp:Person 类的时候，宾语的取值只能有一个；“1”的数据类型被声明为 xsd:integer；这是基数约束，本质上属于属性的局部约束。

(3) 声明属性的传递性

exp:is_part_of rdf:typeowl:TransitiveProperty

exp:is_part_of 是一个传递关系。比如：exp:Aexp:is_part_of exp:B；

exp:Bexp:is_part_of exp:C, 那么根据上述声明，有 exp:Aexp:is_part_of exp: C。

(4) 声明两个属性互反

exp:parent owl:inverseOfexp:children

exp:parent 和 exp:children 是互反关系。比如：exp: 小莉 exp:parent exp: 小亮，那么根据上述声明，有 exp: 小亮 exp:children exp: 小莉

(5) 声明属性的函数性

exp:hasFather rdf:typeowl:FunctionalProperty

exp:hasMother 是一个具有函数性的属性，因为每个人只有一个父亲。

(6) 声明属性的对称性

exp:classmate 是一个具有对称性的属性。比如：exp: 小明 exp:classmate exp: 小丽，那么根据上述声明，有 exp: 小丽 exp:classmate exp: 小明.

(7) 声明属性的局部约束：全称限定

exp:Person owl:allValuesFromexp:Men
exp:Person owl:onPropertyexp:hasFather
exp: hasFather 在主语属于 exp: Person 类的时候，宾语的取值只能来自 exp: Men 这个类。
(8) 声明属性的局部约束：存在限定
exp:APaper owl:someValuesFromexp:EI
exp:APaper owl:onPropertyexp:publishedIn
exp:publishedIn 在主语属于 exp:APaper 类的时候，宾语的取值部分来自 exp:EI 这个类。上面的三元组相当于：A 论文部分发表在 EI 上。
(9) 声明相交的类
exp:Father owl:intersectionOf tmp
tmp rdf:type rdfs:Collection
tmp rdfs:member exp:Person
tmp rdfs:member exp:HasChildren
tmp 是临时资源；是 rdfs:Collection 类型，是一个容器，它的两个成员是 exp:Person，exp:HasChildren。上述三元组说明 exp:Father 是 exp:Person,exp:HasChildren 这两个类的交集。
(10)OWL 其他词汇见表6.4

表 6.4: OWL 中的其它词汇描述

OWL 中的其它词汇	描述
owl:oneOf	声明枚举类型
owl:disjointWith	声明两个类不相交
owl:unionOf	声明类的并运算
owl:minCardinality	最小的基数限定
owl:maxCardinality	最大的基数限定
owl:InverseFunctionalProperty	声明互反类具有函数属性
owl:hasValue	属性的局部约束时，声明所约束类必有一个取值

RDF、RDFS、OWL 描述的是关系的关系，赋予了关系的定义，相当于给了标准的标准，其实只有给定了实体之间和属性之间可能存在的关系的关系才能进行知识推理，因为没有约束的环境是混乱的。

SPARQL(SPARQL Protocol and RDF Query Language)

SPARQL 是一种 RDF 查询语言，可以对不同的数据集撰写复杂的连接 (joins)，主要包括数据的插入、删除和查询操作，被所有主流图数据库支持。以下是关于 SPARQL 语言的一些具体操作。

```
1 数据插入:例如添加ns:孔子ns:老师ns:老子; ns:老子ns:类型ns:哲学家; ns:老子ns:类
    型ns:史学家
```

```
prefix ns:<http://example.org/ns#>
INSERT DATA{
  ns:孔子ns:老师ns:老子.
  ns:老子ns:类型ns:哲学家;
  ns:类型ns:史学家
}

2 数据删除:例如从图中删除(ns:孔子ns:类型ns:教育家)

prefix ns:<http://example.org/ns#>
DELETE DATA{
  ns:孔子ns:类型ns:教育家.
}

3 数据修改:例如修改(ns:小明ns:出生日期`“1903/03/21”`)中的`“1903/03/21”`为`“
    1902/03/21”`

prefix ns:<http://example.org/ns#>
DELETE DATA{ns:小明ns:出生日期`“1903/03/21”`.}
INSERTDATA{ns:费米ns:出生日期`“1902/03/21”`.}

4 查询语言关键字简介
4.1 OPTIONAL
#查询所有选修`A`课程的学生姓名，以及他们的邮箱，关键字`OPTIONAL`指示如果没有邮
    箱，那么依然返回学生姓名，邮箱处空缺。

SELECT ?student ?email
WHERE{
  ?student exp:studies exp:A .
OPTIONAL{
  ?student foaf:mbox ?email .
  }
}

4.2 FILTER
#查询学生姓名，选修课程，以及他们的年龄，如果有年龄，那么年龄必须大于`20`岁。

SELECT ?module ?name?age
WHERE{
  ?student exp:studies ?module .
  ?student foaf:name ?name .
OPTIONAL{
  ?student exp:age ?age .
FILTER (?age $>$ 20) }
 }
```

```
4.3.UNION
#查询选修课程`A1`或`A2`的学生姓名以及邮件，注意，这里的邮件是必须返回的，如果没
    有邮件值，那么就不返回这条记录，注意和`OPTIONAL`区别。

SELECT ?student?email
WHERE{
  ?student foaf:mbox ?email .
  { ?student exp:studies exp:A1 }
UNION {?student exp:studies exp:A2 }
 }

4.4.FROM
#查询选修课程`A`的学生姓名以及邮件和住址，`FROM` 关键字引入了其它本体或者可访问
    的知识库。

SELECT ?student ?email ?home
FROM (http://www2.warwick.ac.uk/rdf/student)
WHERE{
 ?student exp:studies exp:A .
OPTIONAL {
 ?student foaf:mbox?email.
 ?student foaf:homepage ?home. }
 }
```

6.1.4 基于本体工具 (Protégé) 的知识建模实践

Protégé 简介:

Protégé 软件是斯坦福大学医学院生物信息研究中心基于 Java 语言开发的本体编辑和本体开发工具，也是基于知识的编辑器，属于开放源代码软件，实现了 OWL 功能，它主要用于语义网中本体的构建，是语义网中本体构建的核心开发工具，目前最新版本是 5.5.0（截止到 2021-08-13），Protégé 提供了本体概念类，关系，属性和实例的构建，并且屏蔽了具体的本体描述语言，用户只需在概念层次上进行领域本体模型的构建。

Protégé 特点:

Protégé 是一组自由开源的工具软件，用于构建域模型与基于知识的本体化应用程序；提供了大量的知识模型架构与动作，用于创建、可视化、操纵各种表现形式的本体；可以通过用户定制实现域-友好 (领域相关) 的支持，用于创建知识模型并填充数据；Protégé 可以通过两种方式进行扩展：插件和基于 java 的 API；相比与其他的本体构建工具而言，Protégé 最大的好处在于支持中文，在插件上，用 Graphviz 可实现中文关系的显示。

Protégé 用途:

(1) 类建模 (Class modeling)：Protégé 提供了一个图形化用户界面来建模类 (领域概念) 和它们的属性及关系。(2) 实例编辑 (Instance editing)：从这些类中，Protégé 自动产生交互式的形式，全用户或领域专家进行有效实例编辑成为可能。(3) 模型处理 (Model processing)：Protégé

有一些插件库，可以定义语义、解答询问以及定义逻辑行为。(4) 模型交换 (Model exchange)：最终的模型 (类和实例) 能以各种各样的格式被装载和保存，包括 XML、UML 和资源描述框架 RDF。

Protégé 的具体使用步骤:

1. 首先下载安装 protege，由于 Protege 基于 JAVA，所以要先安装配置 JAVA 环境，JAVA 从官网下载即可。

2.protege 的具体使用：

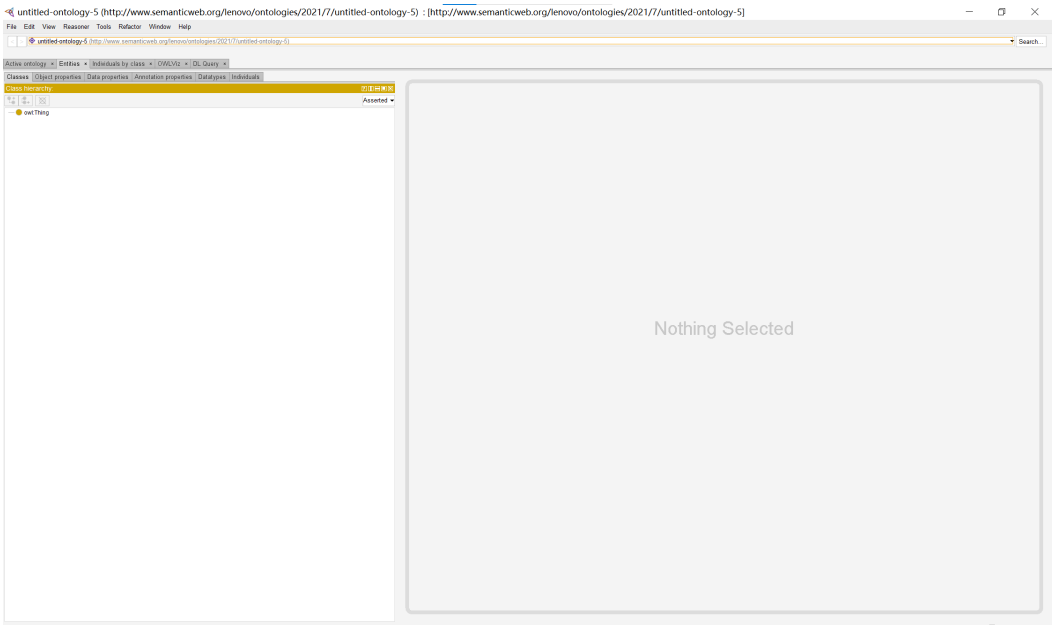


图 6.3

(1) 首先打开界面如图6.3所示，在建立本体时，只需用到 Entities 选项下的一些按钮。

(2) 构建类：其中 Classes 是用于构建类，在 classes 页面，右键点击 owl:Thing ，选择 add subclasses，在出现的对话框中的 name 输入类的名字，点击确定。此外，还可以右键点击一个类，创建它的子类。点击左侧的 Thing，这个是所有类的最高级的父类，我们建立的所有类都是它的子孙，这时候 Thing 上面的三个小图标中的第一个就会亮起，它是创建子类的快捷按钮，第二个是创建兄弟类，第三个是删除选中的类。

(3) 构建类之间的关系：点击 classes 页面中的某一个类，在右面的 describtion 可以观察和修改这个类的属性，比如它的父类是什么，它与什么互斥，我们可以利用这个功能设置类之间的关系。比如：动物和植物是完全不同的事物，它们之间具有排他性（owl:DishointWith）。

(4) 建立对象属性：在 Object properties 页面下完成，这一步是最重要的一环，因为这是三元组中关于“关系”的设置，这一步的基本操作和构建类是一样的，除此之外，对象属性还要传递性、函数性、对称性等特征，这可以在右侧的 Characteristics 中点击左侧的小方框即可。

(5) 建立数据属性：实体之间不仅有关系相连接，还有属于自己的一些属性，比如：对于禅师来说，会有法号，法号必须是 string 类型。切换到 “Data properties”，我们在该界面创建类的属性，即数据属性。其定义方法和对象属性类似。

6.2 知识抽取—实体识别

通过知识抽取技术，可以从一些公开的半结构化、非结构化和第三方结构化数据库的数据中提取出实体、关系、属性等知识要素，并形成结构化数据输出的文本处理技术，包括：实体识别与抽取、实体消歧、事件抽取和关系抽取，我们主要介绍实体识别。

6.2.1 命名实体的定义

狭义地讲，命名实体指现实世界中具体或抽象的实体，如人 (张三)、机构 (中国中文信息学会、阿里巴巴网络技术有限公司)、地点等，通常用唯一的标志符 (专有名称) 表示，如人名、机构名、地名等；广义地讲，命名实体还可以包含时间 (12：00)、日期 (2017 年 10 月 17 日)、数量表达式 (100)、金钱 (一亿美金) 等。至于命名实体的确切含义，只能根据具体应用来确定。比如，在具体应用中，可能需要把住址、电子信箱地址、电话号码、舰船编号、会议名称等作为命名实体。

6.2.2 命名实体识别的任务

一般而言，主要是识别出待处理文本中七类 (人名、机构名、地名、时间、日期、货币和百分比) 命名实体。两个任务：实体边界识别和实体类别标注 (Entity Typing)，实体边界识别即识别出从哪儿开始为某个实体的第一个字，这个实体到哪个字结束，实体类别标注即把识别出来的实体做个标记，让我们清晰的知道它是实体。

6.2.3 命名实体识别的特点

时间、日期、货币和百分比的构成有比较明显的规律，识别起来相对容易。而人名、地名、机构名等识别的难度很大，主要体现在以下三方面 (1) 数量巨大，不能枚举，难以全部收录在词典中；(2) 某些类型的实体名称用字灵活，表达形式多样，而且没有严格的规律可以遵循；(3) 首次出现后往往采用缩写形式。

6.2.4 命名实体识别的方法

(1) 基于词典的命名实体识别方法

方法概述：按照一定的策略将待分析的汉字串与一个充分大的词典中的词条进行匹配，若在词典中找到某个字符串，则匹配成功。典型方法有正向最大匹配法、反向最大匹配法和最短路径法 (最少分词法)。

(2) 无词典的命名实体识别方法

在分词的过程中使用词典的方法是有词典切分，反之是无词典切分，有词典切分的方法一般是基于规则的，无词典切分的方法一般是基于统计的。

(3) 基于规则的方法

基于规则的方法不需要标注训练语料，能直接根据词典和规则进行分词。

(4) 基于统计的方法

基于统计的方法需要标注训练语料训练模型。另外基于统计的方法可以分为生成式统计命名实体识别和判别式命名实体识别。生成式方法的原理是首先建立学习样本的生成模型，再利用模型对预测结果进行间接推理，典型算法为 HMM 等；判别式方法的原理是由字构词的命名实体识别理念，将 NER 问题转化为判别式分类问题 (序列标注问题)，典型算法有 Maxent 、SVM 、CRF 、CNN 、RNN、LSTM+CRF。

注意在知识图谱中不需要用以上提到的算法，利用 lstlisting 中的 spacy 包就可以完成命名实体识别的任务

6.2.5 相关代码

(1) 基本介绍

```
#首先导入spacy
import spacy

#接下来我们需要加载一个`NLP`模型对象，使用`spacy.load()`函数
#此模型需要一个参数，我们使用小型英文预训练模型
nlp = spacy.load("en_core_web_sm")
#`nlp`对象创建后，我们就可以用它来解析文本了.
#使用`nlp`()得到`doc`对象.
#`doc`包含了文本中的诸多数据.
text = "Martin J. Thompson is known for his writing skills. He is also good at
    programming."
doc = nlp(text)
#显示doc对象:
print (doc)
#虽然这看起来和上面的text字符串没什么不同，其实却完全不同.
#我们用分句器来展示这样不同。
for sent in doc.sents:
    print (sent)
#以下内容会对这两部分进行详细说明
#命名实体识别
#for ent in doc.ents:
     #print(ent.text, ent.label_)
#词性标注
#for token in doc:
    #print(token.text, token.pos_)
```

输出结果为

```
Martin J. Thompson is known for his writing skills.
He is also good at programming.
```

图 6.4

我们使用 spaCy 分句器生成所需的输出: 将文本正确地分解成句子。

(2) 中文分词断句

```
#首先导入spacy和zh_core_web_sm
import spacy
import zh_core_web_sm
#加载一个`NLP`模型对象
spacy_nlp = zh_core_web_sm.load()
#`novel.txt`里面保存了一段中文文字
f = open('novel.txt', encoding='GBK')
data = f.read()
print('原文: \n', data)
print()
f.close()
doc = spacy_nlp(data)
# 分词
print('分词: ')
for token in doc:
    print(token.text, end='|')
print()
# 断句
print('断句: ')
for s in doc.sents:
    print(s)
#注意不能同时加载中文和英文训练模型
```

输出结果为

```
原文:
 毛泽东，字润之，笔名子任，湖南湘潭人。中国人民的领袖，伟大的马克思主义者，无产阶级革命家、战略家和理论家。

分词:
毛泽东|，|字润之|，|笔名|子任|，|湖南|湘潭人|。|中国|人民|的|领袖|，|伟大|的|马克思主义者|，|无产阶级|革命家|、|战略家|和|理论家|。
断句:
毛泽东，字润之，笔名子任，湖南湘潭人。
中国人民的领袖，伟大的马克思主义者，无产阶级革命家、战略家和理论家。
```

图 6.5

(3) 英文分词断句

```
#首先导入spacy和en_core_web_sm
import spacy
import en_core_web_sm
#加载一个`NLP`模型对象
spacy_nlp = en_core_web_sm.load()
s = "Next week I'll be in Shanghai."
print('英文原文: ', s)
doc = spacy_nlp(s)
#切词
```

```
print('切词： ')
for t in doc:
    print(t.text, end=' | ')
print()
```

输出结果为

```
英文原文： Next week I'll be in Shanghai.
切词：
Next | week | I | 'll | be | in | Shanghai | .
```

图 6.6

(4) 词性标注

```
import spacy
import pandas as pd
#设置指定选项的值
#无论列数多少，最后显示的是`1000`行
pd.set_option('display.max_columns', 1000)
pd.set_option('display.max_rows', 1000)
#每一行列中显示出来的数据宽度
pd.set_option('display.width', 1000)

#nlp = spacy.load('zh_core_web_sm')
nlp = spacy.load('en_core_web_sm')

# 每个`token`对象有着非常丰富的属性，如下的方式可以取出其中的部分属性。
doc = nlp("Next week I'll be in Shanghai.")
for token in doc:
    print("{0}\t{1}\t{2}\t{3}\t{4}\t{5}\t{6}\t{7}\t{8}\t{9}".format(
        token.text,      #单词
        token.idx,       #单词起始索引
        token.lemma_,    #单词的原型
        token.is_punct,  #是否为标点
        token.is_space,  #是否为空格
        token.shape_,    #形式 Xxx(如第一个字母大写，其余小写)
        token.pos_,      #词性标注
        token.tag_,      #标签
        token.is_stop,   #是否为停用词
        token.dep_       #标记单词是主语，谓语，宾语还是连接词
    ))

print()
#以下是一篇英文文章的词性标注
```

```
# col_names = ['词', '起始位置', '词形还原', '是否为标点', '是否为空格', '形式',
    '词性', 'TAG', '是否为停用词', '依存关系']
# result = pd.DataFrame(columns=col_names)
# f = open('novel.txt', encoding='GBK')
# data = f.read()
# f.close()
# doc = nlp(data)
# for i, token in enumerate(doc):
#     result.loc[i] = [token.text, token.idx, token.lemma_, token.is_punct,
    token.is_space, token.shape_, token.pos_, token.tag_, token.is_stop, token.
    dep_]
# result.to_excel('result.xls', encoding='utf-8')
# print(result)
```

```
Next    0   next    False   False   Xxxx    ADJ JJ  True    amod
week    5   week    False   False   xxxx    NOUN    NN  False   npadvmod
I   10  I   False   False   X   PRON    PRP True    nsubj
'll 11  'll False   False   'xx AUX MD  True    aux
be  15  be  False   False   xx  VERB    VB  True    ROOT
in  18  in  False   False   xx  ADP IN  True    prep
Shanghai    21  Shanghai    False   False   Xxxxx   PROPN   NNP False   pobj
.   29  .   True    False   .   PUNCT   .   False   punct
```

图 6.7

例如：输出结果中的第一行表示 Next 的起始索引是 0，原型是 next，不是标点，不是空格，形式为 Xxxx，是形容词，标签是 JJ，是停用词，是连接词。

(5) 组块分析

```
#加载spacy和pandas
import spacy
import pandas as pd
nlp = spacy.load('en_core_web_sm')  #加载小型英文训练模型
#对括号中的内容进行nlp处理
doc = nlp("Wall Street Journal just published an interesting piece on crypto
    currencies")
#组块分析
for chunk in doc.noun_chunks:
    print(chunk.text, '|', chunk.label_, '|', chunk.root.text)
#以下是对一篇较长文章的组块分析
#输出结果的行名
col_names = ['词', '标签', '词根']
result = pd.DataFrame(columns=col_names)
#打开`Harry Potter.txt`文件
f = open('Harry Potter.txt')
data = f.read()
f.close()
```

```
#对文章内容进行`nlp`处理
doc = nlp(data)
#组块分析
for i, chunk in enumerate(doc.noun_chunks):
    result.loc[i] = [chunk.text, chunk.label_, chunk.root.text]
print(result)
```

```
Wall Street Journal | NP | Journal
an interesting piece | NP | piece
crypto currencies | NP | currencies
                  词  标签         词根
0                Mr  NP         Mr
1       Mrs Dursley  NP    Dursley
2            number  NP     number
3    , Privet Drive  NP      Drive
4              they  NP       they
5               you  NP        you
6              They  NP       They
7   the last people  NP     people
8               you  NP        you
9          anything  NP  anything
10             they  NP       they
11    such nonsense  NP  nonsense
```

图 6.8

从图中可以看出，输出结果是进行组块分析的，例如 the last people 是在一起的，并没有分开，这个块的标签是 NP，词根是 people。

(6) 命名实体识别

```
#加载spacy和displacy
import spacy
from spacy import displacy
#加载小型中文训练模型
nlp = spacy.load('zh_core_web_sm')
#给data赋值
data = '`2011年4月11日17点16分`，`日本东北部的福岛和茨城地区发生里氏7.0级强烈地震`（`震中北纬36.9 度`、`东经140.7度`，`即福岛西南30公里左右的地方`，`震源深度10 公里`，属于浅层地震）`当局已经发布海啸预警震后约30分钟后在日本海地区发生巨型海啸`，同时造成福岛核电站出现核泄漏震后第十天，国际原子能机构对于日本政府反应迟钝进行了谴责'
#对data进行nlp处理
doc = nlp(data)
result = set()
#实体识别
for ent in doc.ents:
```

```
    print(ent.text, ent.label_)
    if ent.label_ == 'PERSON':
        result.add(ent.text)
print(len(result), result)
# PERSON 人名;
# NORP是国籍或宗教团体;
# GPE标识位置（城市、国家等等）;
# DATE 标识特定的日期或日期范围,
# ORDINAL标识一个表示某种类型的顺序的单词或数字
```

```
2011年4月11日17点16分 DATE
日本东北部 ORG
茨城地区 LOC
北纬 LOC
9 度 QUANTITY
7度 QUANTITY
30公里 QUANTITY
10公里 QUANTITY
30分 DATE
日本海地区 LOC
第 十天 DATE
日本 GPE
0 set()
```

图 6.9

输出结果为如图6.9, 从结果中可以得知 2011 年 4 月 11 日 17 点 16 分为 DATE，日本为 GPE。

(7) 句法依存分析

```
#加载spacy和displacy
import spacy
from spacy import displacy
#加载小型中文训练模型
nlp = spacy.load('zh_core_web_sm')
#对括号中的句子进行nlp处理
doc = nlp('1930年，胡🅵青正在北京师范大学念书。')
#对doc进行句法依存分析
html = displacy.render(doc, style='dep', jupyter=False, options={'distance':
    80})
f = open('result.html', mode='w', encoding='ansi')
f.write(html)
f.close()
print(html)
```

输出结果如图6.10

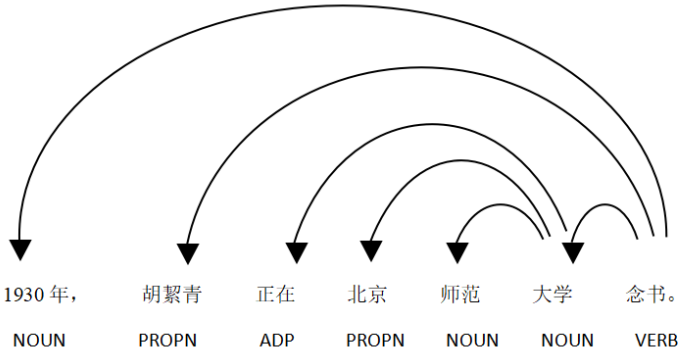


图 6.10

6.3 知识推理

6.3.1 知识推理的定义与任务

推理是逻辑学、哲学、心理学、人工智能等学科中的重要概念，早在古希腊时期，著名哲学家亚里斯多德就提出三段论作为现代演绎推理的基础，在计算机科学及人工智能领域，推理是一个按照某种策略从已知事实出发去推出结论的过程，知识推理亦是如此。

结构化知识图谱的一大优势就是能够支撑高效的推理，典型的推理任务包括知识库补全和知识库问答。知识库补全也称知识图谱上的链接预测，其本质是根据知识库中已有的知识推断出新的、未知的知识，知识库补全可以用来建立更全面的知识库，是知识库构建的重要手段之一；知识库问答即基于知识库的问答主要是通过对自然问句的解析，再从知识库中寻找答案的过程，但由于知识库中知识不完备等原因，知识库问答也需要推理技术的支撑。

6.3.2 知识推理的分类

归纳推理和演绎推理

归纳推理

归纳推理是根据部分对象所具有的性质，推出一类事物中所有对象都具有这类性质的推理方式，可分为三个步骤：（1）对部分资料进行观察、分析和归纳整理；（2）得出规律性的结论，即猜想；（3）检验猜想。例如：我们看到第一只天鹅是白色的，第二只天鹅也是白色的，第三只天鹅也是白色的，看了几千几万只天鹅，发现这些天鹅都是白色的，所以我们下了一个结论，认为所有的天鹅都是白色的。归纳推理是从特殊到一般的过程，即使推理的前提是正确的，也不能保证推理的结论是正确的，但是有些时候他是可能能得到结论的。

演绎推理

演绎推理是从一般性的前提出发，通过推导即“演绎”，得出具体陈述或个别结论的过程。亚里士多德提出的著名三段论：（1）一个一般性的原则（大前提）；（2）大前提的特殊化陈述（小前提）；（3）引申出的特殊化陈述符合一般性原则的结论。例如：老马是认识道路的（大前提），

这几匹马是老马（小前提），可以演绎出这几匹马是认识道路的（结论）。演绎推理是从一般到特殊的过程，只要演绎的前提是正确的，一般来讲，演绎的结果是不会出错的。但演绎推理不仅仅局限于三段论，也不只是从一般到特殊的过程，它有着强烈的“演绎”特性，重在通过利用每一个证据，逐步地推导到目标或意外的结论。例如：A 盒子说“礼物不在此盒中”，B 盒子说“礼物在 C 盒中”，C 盒子说“礼物不在此盒中”，上述三句话中，最多只有一句话是真的，这个推理便可以用演绎法逐步地进行。

确定性推理与不确定性推理

确定性推理

确定性推理大多指确定性逻辑推理，它具有完备的推理过程和充分的表达能力，可以严格地按照专家预先定义好的规则准确地推导出最终结论。但是确定性推理很难应对真实世界中，尤其是存在于网络大规模知识图谱中的不确定甚至不正确的事实和知识，例如“一个人和其父亲拥有同样的国籍”这条规则在现实生活中大部分情况下都是正确的，但也不排除移民、母方国籍等因素使得少量事实不满足（不确定的知识）；大规模知识图谱 YAGO 根据抽样统计宣布其中含有 5% 左右的错误事实（不正确的事实）。

不确定性推理

不确定性推理也被称为概率推理，是统计机器学习中一个重要的议题。它并不是严格地按照规则进行推理，而是根据以往的经验和分析，结合专家先验知识构建概率模型，并利用统计计数、最大化后验概率等统计学习的手段对推理假设进行验证或推测。不确定性推理可以有效建模真实世界中的不确定性。

符号推理与数值推理

符号推理

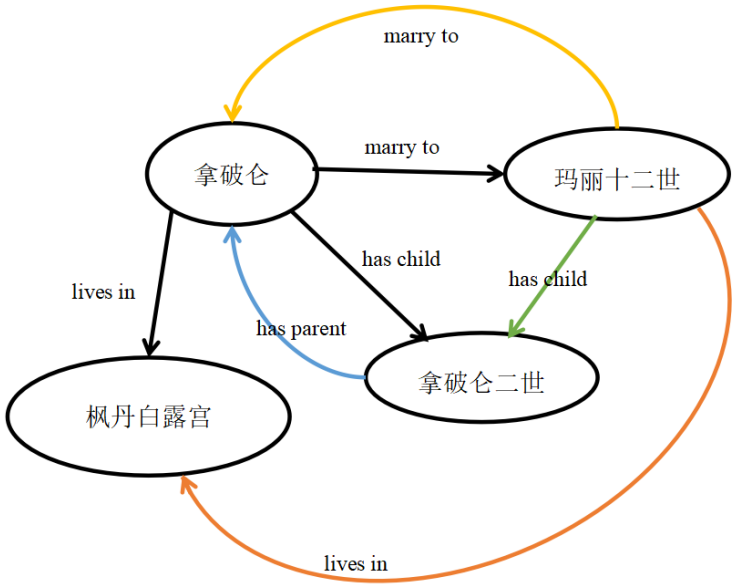


图 6.11

符号推理又可以看成是传统的逻辑推理，它的特点就是在知识图谱中的实体和关系符号上直接进行推理操作。符号推理的学习推理规则是从现实的大规模网络知识图谱中自动地总结出逻辑规则，这一任务也被称为逻辑规则挖掘。例如 ∀ x,y CapitalOf(x,y)⟹LocatedIn(x,y)，便是一个谓词逻辑规则挖掘的实例，其中，谓词是 CapitalOf，LocatedIn；个体变量是 x,y，逻辑蕴含 ⟹ 表示“若 . . . ，则 . . . ”的语义；CapitalOf(x,y) 表示该规则的前提；LocatedIn(x,y) 表示该规则的结论。若我们有了以下推理规则:

hasChild(A,B)⟹hasParent(B,A)，

marryTo(A,B)⟹marryTo(B,A)，

marryTo(A,B)∩hasChildren(A,C)⟹hasChild(B,C)，

marryTo(A,B)∩livesIn(A,C)⟹livesIn(B,C)

我们从图6.11便能推导出玛丽十二世住在枫丹白露宫的概率极大。

符号演算中比较特殊的一类挖掘是频繁子图挖掘，频繁子图挖掘是搜集知识图谱的规则实例，再将规则实例中的实体替换成变量，同时设定一些约束，以快速地确定规则的实用性，并根据知识图谱中规则实例的支撑来快速评价挖掘到的规则。例如若知识图谱中存在以下这种大量的规则实例，我们便能得到这样一条规则：一个人的父亲的父亲是这个人的爷爷，用符号表示是:

父亲（x，y）∩ 父亲（y，z）⟹ 爷爷（x，z）

父亲（向芷, 向佐）∩ 父亲（向佐, 向华强）⟹ 爷爷（向芷, 向华强）

父亲（姚沁蕾, 姚明）∩ 父亲（姚明, 姚志源）⟹ 爷爷（姚沁蕾, 姚志源）

父亲（霍中曦, 霍启刚）∩ 父亲（霍启刚, 霍震霆）⟹ 爷爷（霍中曦, 霍震霆）

父亲（步步, 吴奇隆）∩ 父亲（吴奇隆, 吴飞元）⟹ 爷爷（步步, 吴飞元）

数值推理

与符号推理相对应的就是数值推理，数值推理就是使用数值，尤其是向量矩阵等数值计算的方法，捕捉知识图谱上隐式的关联，模拟推理的过程，基于分布式知识表示的推理就是典型的数值推理方法。数值推理主要用于捕捉实体和关系之间的隐式关联，相对于符号推理，数值推理可以直接使用数据参与运算且计算速度非常快。

分布式知识表示 (KnowledgeGraphEmbedding) 的核心思想是将符号化的实体和关系在低维连续向量空间进行表示，在简化计算的同时最大程度保留原始的图结构，分布式空间映射对特征间的复杂关系进行了解耦，减少了维数灾难问题 (curse of dimensionality), 基本步骤：

(1) 实体关系表示：定义实体和关系在向量空间中的表示形式。

(2) 打分函数定义：定义打分函数，衡量每个三元组成立的可能性。

(3) 表示学习：构造优化问题，学习实体和关系的低维连续向量表示。

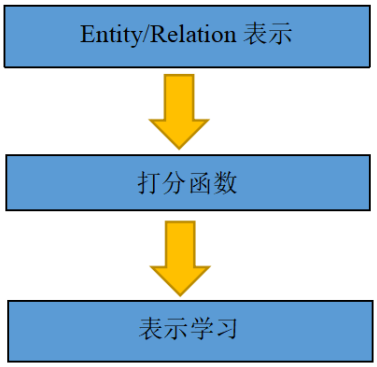


图 6.12

基于分布式知识表示的推理主要是知识图谱上的链接预测即预测一个三元组的头实体或者尾实体。一个正确的三元组，移除它的头实体或者尾实体，检验模型能否预测出正确的头实体或者尾实体。可用于知识补全时的检验，在补全知识图谱缺失关系时，可以选一条边连接任意两个实体，构成新的三元组，再判断该三元组是否正确。分布式知识表示方法可以分为翻译模型 (translationmodels) 和匹配模型 (semanticmatching models)，翻译模型是采用基于距离的打分函数来衡量三元组成立的可能性，语义匹配模型是采用基于相似度的打分函数来衡量三元组成立的可能性。

6.3.3 频繁项集的一个案例

```
# -*- coding: utf-8 -*-
# 分析某导演的电影的 频繁项集和关联规则
from efficient_apriori import apriori
import csv
```

```
director = u'宁浩'
file_name = './' + director + '.csv'
lists = csv.reader(open(file_name, 'r', encoding='utf-8-sig'))
# 数据加载
data = []
for names in lists:
        name_new = []
        for name in names:
                # 去掉演员数据中的空格
                name_new.append(name.strip())
        data.append(name_new[1:])

# 挖掘频繁项集和关联规则
itemsets, rules = apriori(data, min_support=0.4, min_confidence=0.3)
print('频繁项集：', itemsets)
print('关联规则：', rules)
```

输出结果为

```
频繁项集：  {1: {('黄渤',): 7, ('徐峥',): 6}, 2: {('徐峥', '黄渤'): 6}}
关联规则：  [{黄渤} -> {徐峥}, {徐峥} -> {黄渤}]
```

图 6.13

从结果中可以得知在宁浩导演的电影中一共有两个频繁项集，第一个是黄渤单独出现了 7 次，徐峥单独出现了 6 次，第二个是黄渤和徐峥共同出现了 6 次，故能得到如下关联规则：在黄渤出现的情况下，徐峥大概率会出现，反之亦然。

第 7 章　强化学习（一）

7.1　强化学习概述

7.1.1　什么是强化学习

强化学习的学习思路和人比较类似，是在实践中学习，比如学习走路，如果摔倒了，那么我们大脑后面会给一个负面的奖励值，说明走的姿势不好。然后我们从摔倒状态中爬起来，如果后面正常走了一步，那么大脑会给一个正面的奖励值，我们会知道这是一个好的走路姿势。(用一个简单的例子说明：想想训练一条小狗，你不会告诉他该做什么，因为它听不懂。他做对了你给他奖励，做错了给惩罚。慢慢它就知道你啥意思了。强化学习也是一样。而你把这个小狗看成一个机器人，用这个相同的思想来训练的话，那就是强化学习了。) 在给出强化学习的定义之前先看强化学习的一些要素定义。

1. **智能体**：自主采取行动以完成任务的强化学习系统；指强化学习需要优化的部分，我们能精确控制。

2. **环境**：智能体的交互对象，我们不能直接控制。比如我和小明下棋，我就是智能体，小明是智能体交互的对象是环境，我可以控制我的行为但我不能控制小明的行为。智能体做的事为根据当前的状态选择采取一个什么样的动作；环境做的事为根据当前的状态与行为给出反馈值，进一步影响智能体对其下一步的动作调整。

   - 注：注意区分智能体和环境与物理界限不同。如：扫地机器人电动机和机械结构，以及它的传感硬件是环境，基于强化学习的路径规划算法被认为是智能体。

3. **状态**：记为 S。t 时刻环境的状态 S_t 是它的环境状态集中某一个状态。状态是历史的一个函数，历史是一个状态、动作和回报的序列。表示为 $H_t = (s_1, a_1, r_1, s_2, a_2, r_2, \ldots, s_t, a_t, r_t)$ 是指智能体 t 时刻以前与环境的所有交互变量 s_1 a_1 r_1 … 组成的序列。本质上，状态是历史的一个函数 $S_t = (f(H_t))$。如下棋时，棋盘现在的布局可以被认为是状态 S_t 。如在状态 s_1 时智能体采取动作 a_1，此时环境给出一个反馈值和下一步的状态 s_2。

4. **行为（动作）**：记为 A。t 时刻个体采取的动作 a_t 是它的动作集中某一个动作。动作是智能体主动和环境交互的媒介，动作必须对环境起到一定的控制作用（尤其是对回报），如：打砖块时上/下动作不能改变环境，所以此动作不在强化学习的考虑范围，打砖块游戏只有左右这两个动作。

5. **回报**：记为 R。t 时刻个体在状态 S_t 采取的动作 a_t 对应的奖励 r_{t+1} 会在 $t+1$ 时刻得到。回报衡量了智能体在时间 t 上做的有多好，智能体的目标就是最大化累计回报。回报包括立即回报和长期回报。

   - 立即回报：当智能体在时间 t 做出动作 a_t 时，收到回报 R_t.
   - 长期回报：智能体与环境不断交互，会收到回报序列 R_t，R_{t+1} ,R_{t+2}…。一种通用的累计回报的方式是将这些回报值进行加权求和:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma R_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

   其中 γ 是折扣因子/衰减因子，衡量了未来时刻的回报在当前时刻的价值。在 $[0,1]$ 之间。如果为 0，则是贪婪法，即价值只由当前延时奖励决定，如果是 1，则所有的后续状态奖励和当前奖励一视同仁。大多数时候，我们会取一个 0 到 1 之间的数字，即当前延时奖励的权重比后续奖励的权重大。

6. **策略**：记为 π, 它代表个体采取动作的依据，即个体会依据策略 π 来选择动作。最常见的策略表达方式是一个条件概率分布 $\pi(a|s)$, 这个概率分布表示了选择每个动作的概率，也就是在状态 s 时采取动作 a 的概率。即 $\pi(a|s) = p(A_t = a|S_t = s)$。

7. **值函数**：包括状态值函数与动作值函数。其是用来衡量一个智能体在某个状态有多好或者在某个状态选择某个动作有多好。某个策略 π 下选择某个状态 S 的值函数一般用 $v_\pi(S)$ 表示，定义为从 S 开始根据策略 π 选择行为能都得到的期望回报；行为值函数一般用 $Q_\pi(S,a)$ 表示，定义为在状态 S 选择行为 a 后智能体再根据策略 π 选择行为能够得到的期望回报。值函数一般是一个期望函数。虽然当前动作会给一个延时奖励 R_{t+1}, 但是光看这个延时奖励是不行的，因为当前的延时奖励高，不代表到了 $t+1$,$t+2$,… 时刻的后续奖励也高。比如下象棋，我们可以某个动作可以吃掉对方的车，这个延时奖励是很高，但是接着后面我们输棋了。此时吃车的动作奖励值高但是价值并不高。因此我们的价值要综合考虑当前的延时奖励和后续的延时奖励。价值函数 $v_\pi(s)$ 和 $Q_\pi(S,a)$ 一般可以表示为下式，不同的算法会有对应的一些价值函数变种，但思路相同:

$$v_\pi(s) = E_\pi(R_{t+1} + \gamma R_{t+2} + \gamma R_{t+3} + \ldots | S_t = s)$$

$$Q_\pi(s) = E_\pi(R_{t+1} + \gamma R_{t+2} + \gamma R_{t+3} + \ldots | S_t = s, A_t = a)$$

8. **状态转化模型**：表示为 $P_s^a s^{'}$，可以理解为一个概率状态机，它可以表示为一个概率模型，即在状态 s 下采取动作 a, 转到下一个状态 $s^{'}$ 的概率。

9. **探索率**：记为 ϵ, 这个比率主要用在强化学习训练迭代过程中，由于我们一般会选择使当前轮迭代价值最大的动作，但是这会导致一些较好的但我们没有执行过的动作被错过。因此我们在训练选择最优动作时，会有一定的概率不选择使当前轮迭代价值最大的动作，而选择其他的动作。

以上就是强化学习的基本要素了。当然，在不同的强化学习模型中，会考虑一些其他的要素，或者不考虑上述要素的某几个，但是以上是大多数强化学习模型的基本要素。

7.2 强化学习整体结构

下面给出强化学习的定义：

定义：强化学习是智能体与环境进行交互，通过环境给出的的回报来调整自身行为，目的为使得到的累积期望回报最大。是一种以环境反馈作为输入的机器学习方法。

比如下棋，每走一步棋我的选择是通过计划——预测可能的回报——以及对特定位置和动作的期望判断来作出的。可能我走一个马被对方吃掉了一个车，立即回报是负值，但是我因为这一个马最后吃掉了对方的帅那我这一步的期望回报是大的。强化学习不拘泥于当前的回报，目标是使得累计期望回报最大。

7.2.1 强化学习的基本思路

智能体采取一个动作环境去接受这个动作并给出一个反馈值和下一步的状态。上面的智能体代表我们的算法执行智能体，我们可以操作智能体来做决策，即选择一个合适的动作 a_t。下面的环境是智能体交互的对象，成为环境，它有自己的状态模型，我们选择了动作 a_t 后，环境的状态会变成 s_{t+1}，同时环境给出了我们采取动作 a_t 的延时奖励 r_{t+1}。然后智能体可以继续选择下一个合适的动作 a_{t+1}，然后环境的状态又会变成 s_{t+2}，又有新的奖励值 r_{t+1}，…，这就是强化学习的思路。

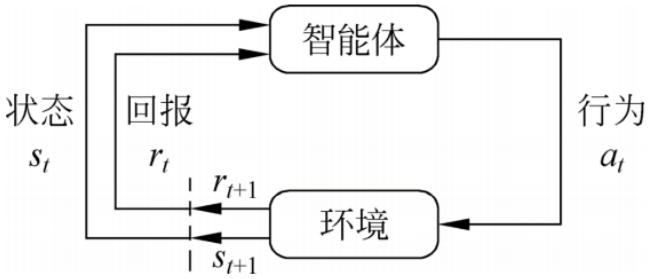


图 7.1: 智能体与环境交互过程

7.2.2 强化学习和其他机器学习的关系

先来看机器学习的分类情况：

$$\text{机器学习}\begin{cases}\text{监督学习;}\\ \text{非监督学习;}\\ \text{强化学习.}\end{cases}$$

可知强化学习是和监督学习、非监督学习并列的机器学习方法。下面解释强化学习与其它两种机器学习的区别:

监督学习：带有标记数据，预测未知数据标记。是静态数据，数据之间相互独立。

非监督学习：无标记数据，挖掘数据潜在结构，如聚类。是静态数据，数据之间相互独立。

强化学习:

- 没有标记，但有延迟的回报；
- 序贯决策过程；
- 数据是智能体与环境不断交互产生，是动态数据；
- 数据之间高度相关。

总的来说强化学习和监督学习最大的区别是它是没有监督学习已经准备好的训练数据输出值的。强化学习只有奖励值，但是这个奖励值和监督学习的输出值不一样，它不是事先给出的，而是延后给出的，比如上面的例子里走路摔倒了才得到大脑的奖励值。同时，强化学习的每一步与时间顺序前后关系紧密。而监督学习的训练数据之间一般都是独立的，没有这种前后的依赖关系。再来看看强化学习和非监督学习的区别。也还是在奖励值这个地方。非监督学习是没有输出值也没有奖励值的，它只有数据特征。同时和监督学习一样，数据之间也都是独立的，没有强化学习这样的前后依赖关系。

7.3 强化学习的环境

gym 包提供了强化学习中与 agent 交互的 environments，里面有很多的环境。gym 包安装问题说明：

要求 lstlisting 版本大于 3.5，使 pip 可以简单安装。安装的方式就是 pip install gym, 如果用 conda install gym 会有安装渠道的问题，根据提示进行安装。此时只是安装了 gym 中的 algorithms,toy text,classic control 这三种类型的环境，如果需要其它的环境可以单独安装，比如想使用 box2D 就使用命令 pip install box2D 进行安装该环境。

7.3.1 使用函数讲解

1. gym.make() 创建环境，可选择不同的环境.
2. reset() 为重新初始化函数。在强化学习算法中，智能体需要一次次地尝试，累积经验，然后从经验中学到好的动作。一次尝试我们称之为一条轨迹或一个 episode. 每次尝试都要到达终止状态. 每次尝试结束后，智能体需要从头开始，这就需要智能体具有重新初始化的功能，函数 reset() 就是这个作用，每次调用之后会有一个初始状态。
3. render() 该函数在这里扮演图像引擎的角色。一个仿真环境必不可少的两部分是物理引擎和图像引擎。物理引擎模拟环境中物体的运动规律；图像引擎来显示环境中的物体图像。其实，对于强化学习算法，该函数可以没有。但是，为了便于直观显示当前环境中物体的状态，图像引擎还是有必要的。另外，加入图像引擎可以方便我们调试代码
4. step() 该函数在仿真器中扮演物理引擎的角色。其输入是动作 a，输出是：下一步状态，立即回报，是否终止，调试项。该函数描述了智能体与环境交互的所有信息，是环境文件中最重要的函数。在该函数中，一般利用智能体的运动学模型和动力学模型计算下一步的状态和立即回报，并判断是否达到终止状态。

7.3.2 例子

官网例子

```
#加载gym包
import gym
#选择CartPole-v0的环境，也可以换成其他的环境 CartPole-v0、MountainCarContinuous-
    v0
env = gym.make('MountainCarContinuous-v0')
#进行智能体与环境的交互
env.reset()#环境初始化
for _ in range(1000):
    env.render()
    env.step(env.action_space.sample()) # take a random action
env.close()
```

例子中相当于做了 1000 次的尝试，每次尝试的步数不确定，根据是否结束即 done 是否为 true 来决定每条轨迹的步数。例如，在状态 s_1 下执行动作 a_1 done 的返回值为 true, 此时结束该次尝试，环境重启，进入下一次尝试，否则继续在状态 s_2 下执行动作 a_2.....。运行之后可以看到，当随机给动作时小车要么向前要么向后要么不动，如果开启上帝视角的话可以让小车向左滑动到达一个足够高的位置，然后释放使其到达终点，希望机器学到这个本事，就是目前要做的事。上述的例子其实是让大家理解环境，当前状态 s_1 有一个 a_1 环境会告诉我 s_2,r_1。这里都假定环境能给出 reward 进行后续的操作。

例子

```
import gym
#对环境进行输出
if __name__ == '__main__':
    env_names = list(gym.envs.registry.env_specs)
    print(env_names)
#选择一个环境，也可选择BipedalWalker-v3/CliffWalking-v0/CarRacing-v0
    env = gym.make('CarRacing-v0')
    print('env.action_space = ', env.action_space)  #输出行为空间
    observation = env.reset()  #重置初始状态
    action_number = 0
    #智能体和环境进行交互
    for _ in range(100):
        env.render()
        action = env.action_space.sample()  # 随机选择一个动作
        next_observation, reward, done, info = env.step(action)
        action_number += 1
        print(action, next_observation, reward, done, action_number, info)
        #如果结束一次尝试执行下方命令进行下一次尝试
        if done:
```

```
            action_number = 0
            observation = env.reset()
    env.close()
```

上述例子所使用的 CarRacing-v0 环境中动作空间包含三个量，打方向盘、踩油门、刹车，三个量的取值范围分别为：打方向盘（-1，+1），踩油门（0，+1），刹车（0，+1）。状态空间是 96*96 的像素。使用该环境是赛车与跑道的交互，赛车随机选择一个动作如踩油门，做完该动作之后赛车到达下一个状态即下一个像素点，并得到一个奖励，环境给予智能体的奖励分为两个部分：一个是随着时间的流逝，会一直付出代价，-0.1/frame，如果按照 30frams/s 来算的话，就是-3/s；第二部分是经过赛道上铺的瓦片所获得的奖励，每经过一个瓦片都将给予智能体 1000/N 的奖励，N 为轨道上的总瓦片数，赛车碰到边界就结束。

7.4 解决强化学习问题

7.4.1 强化学习问题

强化学习的两个基本问题：

1. 预测问题。求解给定策略的状态价值函数的问题。这个问题的求解过程我们通常叫做策略评估。

2. 控制问题。求解最优的价值函数和策略。

下面介绍求解强化学习基本问题的方法，分为基于模型的方法和不基于模型的方法。其中基于模型的方法有动态规划，不基于模型的方法有蒙特卡罗法。

7.4.2 马尔可夫决策过程

首先看定义：

马尔可夫性： 某一状态蕴涵了所有相关的历史信息，当前状态已知时所有的历史信息就不再需要，即当前状态可以决策未来，则该状态具有马尔可夫性。如下棋时只要知道当前的棋盘状态就不需要知道之前的每一步是怎么下的。即

$$p(s_{t+1} \mid s_t) = p(s_{t+1} \mid s_t, s_{t-1}, ..., s_2, s_1) \tag{7.1}$$

马尔可夫过程： 具有马尔可夫性的随机过程。它是一个无记忆的随机过程，是一个二元组用 (S, P) 表示。

马尔可夫决策过程： 在强化学习中马尔可夫决策过程是一个五元组，用 $< S, A, P, R, \gamma >$ 表示。其中 S 表示环境的状态集合；A 表示智能体的动作集合；P 表示状态转移概率；R 表示回报函数；γ 表示折扣因子取值为 (0 1)。

引入马尔可夫决策过程有两个目的：

首先是对模型的简化。比如环境的状态转移概率模型，如果按照真实的环境转化过程看，由当前的状态 s 转化到下一个状态 s' 的概率既与上一个状态 s 有关，还与上上个状态，以及上上上个状态有关。这一会导致我们的环境转化模型非常复杂，复杂到难以建模。因此我们需要对强化学习的环境转化模型进行简化。简化的方法就是假设状态转化的马尔科夫性，也就是假设转化到下一个状态 s' 的概率仅与上一个状态 s 有关，与之前的状态无关；其次是马尔可夫决策过程是对强化学习问题的数学描述，几乎所有的强化学习问题都可以用马尔可夫决策过程描述。

7.4.3 贝尔曼方程

为求解值函数，引入贝尔曼方程。下面介绍表示值函数与后继值函数迭代关系的贝尔曼方程。在贝尔曼方程中值函数的表达式可以分解为立即回报和下一刻值函数的折扣期望。
对于状态值函数：

$$\begin{aligned} V_\pi(s) &= E(G_t|S_t = s) \\ &= E(R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots |S_t = s) \\ &= E(R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \ldots)|S_t = s) \\ &= E(R_{t+1} + \gamma G_{t+1}|S_t = s) \\ &= E(R_{t+1} + \gamma V_\pi(S+1)|S_t = s) \end{aligned} \tag{7.2}$$

同样对于动作值函数：

$$\begin{aligned} Q_\pi(s,a) &= E(G_t|S_t = s, A_t = a) \\ &= E(R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots |S_t = s, A_t = a) \\ &= E(R_{t+1} + \gamma Q_\pi(S+1)|S_t = s, A_t = a) \end{aligned} \tag{7.3}$$

贝尔曼期望方程

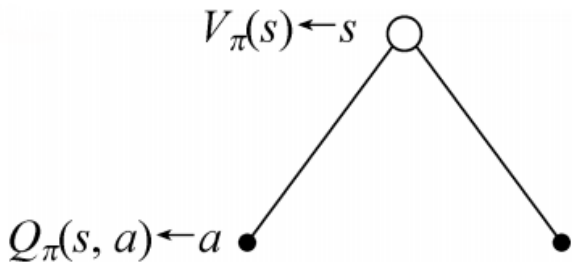


图 7.2

图7.2中空心圆圈代表状态，实心圆点代表动作，状态指向动作表示在该状态下采取某种动作，动作指向状态表示在状态行为对下采取某动作发生了状态转变。其中在状态 s 时对应一个

状态值函数, 在该状态下采取动作 $a \in A$ 对应一个行为值函数，可以得到状态值函数与行为值函数的关系式：

$$V_\pi(s) = \Sigma\pi(a|s)Q_\pi(s,a) \tag{7.4}$$

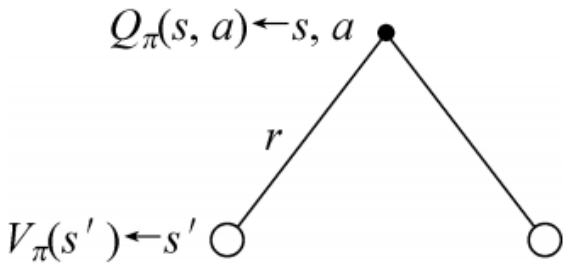


图 7.3

图7.3中状态行为对 (s,a) 对应一个行为值函数，在该状态 s 时采取动作 a, 到达下一个状态 $s^{'} \in S$ 和一个立即回报 R_s^a。由此可得到行为值函数和后继状态值函数的关系式：

$$Q_\pi(s,a) = R_s^a + \gamma\Sigma P_s^a s^{'} V_\pi(s^{'}) \tag{7.5}$$

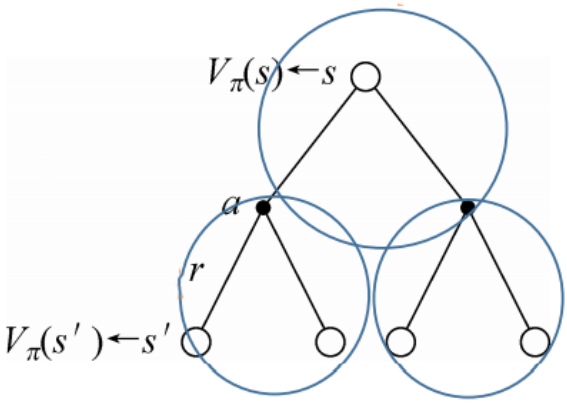


图 7.4

将两个图7.3置于图7.2之下可得到状态值函数与后继状态值函数的关系式：

$$V_\pi(s) = \Sigma\pi(a|s)(R_s^a + \gamma\Sigma P_s^a s^{'} V_\pi(s^{'})) \tag{7.6}$$

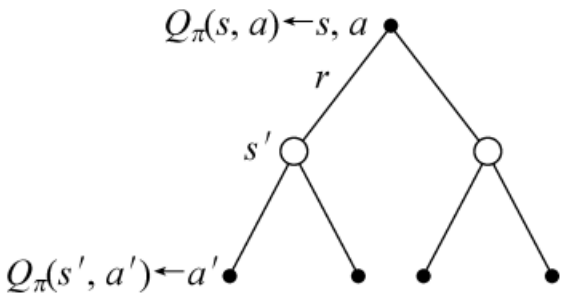


图 7.5

将两个图7.2置于图7.3之下可得到行为值函数与后继行为值函数的关系式：

$$Q_\pi(s,a) = R_s^a + \gamma \Sigma P_s^a s^{'} \Sigma \pi(a^{'}|s^{'}) Q_\pi(s^{'}, a^{'}) \tag{7.7}$$

下面看一个例子，来看看如何用贝尔曼期望方程来求解值函数。

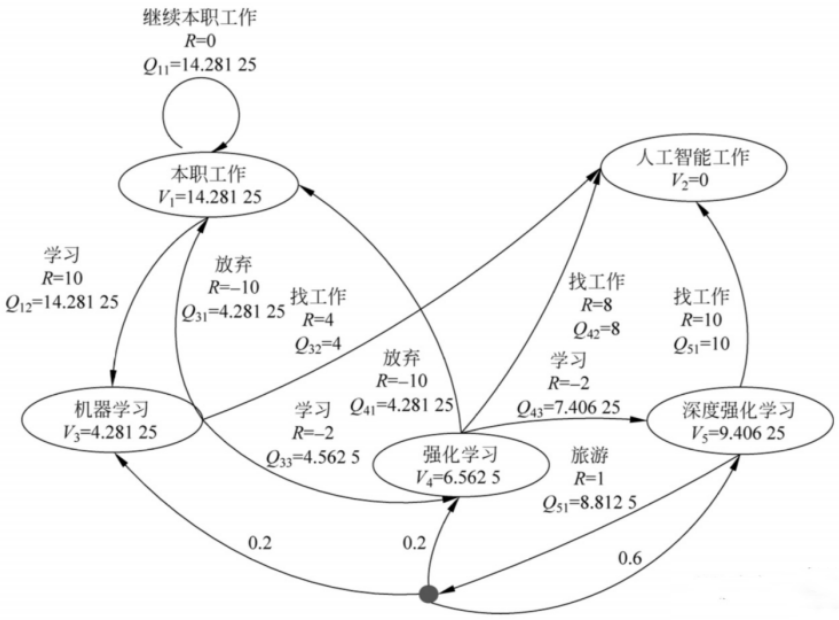


图 7.6: 工作学习的 MDP

图7.6共有五个状态，分别为本职工作 s_1、人工智能工作 s_2、机器学习 s_3、强化学习 s_4 和深度强化学习 s_5；动作空间包括：继续本职工作 a_1、学习 a_2、放弃学习 a_3、找工作 a_4、旅游 a_5；其中小黑点处表示在深度强化学习状态时做了一个随机行为，有 0.2 的概率会到达机器学习这一状态即 $P_{s_5}^{a_5} s_3 = 0.2$、0.2 的概率会到达强化学习状态即 $P_{s_5}^{a_5} s_4 = 0.2$、0.6 的概率

会继续留在深度强化学习这一状态，即 $P_{s_5}^{a_5} s_5 = 0.6$。除了小黑点处其余状态转移概率都是 1。假设折扣因子 $\gamma = 1$, 策略为随机策略，则根据状态值函数与后继状态的值函数表达式可知:

$$\begin{cases} V_1 = \frac{1}{2}[(0+1*1*V_1)+(10+1*1*V_3)], & ①; \\ V_2 = 0, & ②; \\ V_3 = \frac{1}{3}[(4+1*1*V_2)+(-10+1*1*V_1)+(-2+1*1*V_4), & ③; \\ V_4 = \frac{1}{3}[(-10+1*1*V_1)+(8+1*1*V_2)+(-2+1*1*V_5)], & ④; \\ V_5 = \frac{1}{2}[(10+1*1*V_2)+(1+(0.2*1*V_3)+(0.2*1*V_4)+(0.6*1*V_5))], & ⑤. \end{cases}$$

上面五个方程联立即可求出各个状态的值函数。

贝尔曼最优方程

解决强化学习问题意味着要寻找一个最优的策略，在该策略下让智能体在与环境交互过程中获得始终比其它策略都要多的收获，这个最优策略我们可以用 π^* 表示。一旦找到这个最优策略 π^*，那么我们就解决了这个强化学习问题。一般来说，比较难去找到一个最优策略，但是可以通过比较若干不同策略的优劣来确定一个较好的策略，也就是局部最优解。

如何比较策略的优劣呢? 一般是通过对应的价值函数来比较的，也就是说，寻找较优策略可以通过寻找较优的值函数来完成。最优值函数定义为在所有策略中最大的值函数。
可以定义最优状态值函数是所有策略下产生的众多状态值函数中的最大者，即:

$$V^*(s) = \max V_\pi(s), s \in S$$

比如玩游戏时，当处在某个状态时，有两种策略可以选择，一种是激进策略 π_1 （动作空间包括开火、快速前进等），一种是保守策略 π_2（动作空间包括躲闪、下蹲前进等），若在策略 π_1 下的状态值函数 $V_{\pi_1}(s)$ 大于在策略 π_2 下的状态值函数 $V_{\pi_2}(s)$，则认为 $V^*(s) = V_{\pi_1}(s)$。同理，最优的行为值函数是所有策略下产生的众多行为值函数中的最大者，即:

$$Q^*(s,a) = \max Q_\pi(s,a), s \in S$$

对比贝尔曼期望方程可以得到贝尔曼最优方程的四种形式:

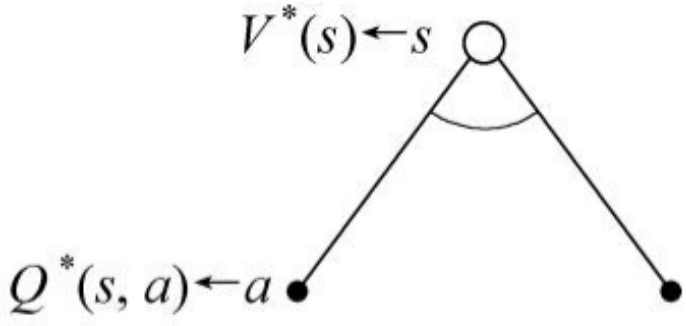


图 7.7

同样空心圆圈代表状态，实心圆点代表动作，状态指向动作表示在该状态下采取某种动作，动作指向状态表示在状态行为对下采取某动作发生了状态转变。图7.7表示在状态 s 时采取动作

$a \in A$，那么状态 s 对应的最优状态值函数与最优行为值函数的关系式为：

$$V^*(s) = \max Q^*(s, a) \tag{7.8}$$

这是因为智能体是可以选择动作的，在状态 s 时可以采取动作 a_1，也可以采取动作 a_2，.....，在每个动作下都有一个对应的最优行为值函数（比如采取动作 a_1，后继采取不同的策略得到不同的行为值函数，最大的行为值函数则是采取动作 a_1 最优的行为值函数），对比贝尔曼期望方程中的式7.1可知道上述等式成立。

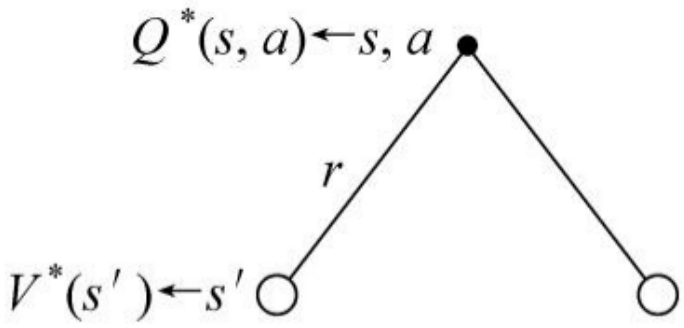


图 7.8

图7.8表示在该状态 s 时采取动作 a, 到达下一个状态 $s^{'} \in S$。对比贝尔曼期望方程式（2）可知在状态 s 时采取动作 a 的最优行为值函数与后继最优状态值函数的关系式为：

$$Q^*(s, a) = R_s^a + \gamma \Sigma P_s^a s^{'} V^*(s^{'}) \tag{7.9}$$

这是因为状态是环境给的，智能体不能自己选择状态，所以在贝尔曼期望方程的基础上取最优。

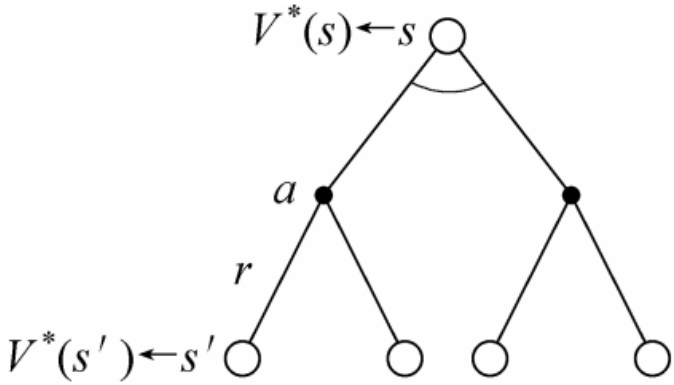


图 7.9

图7.9为两个图7.8加到图7.7下，得到最优值函数与后继最优值函数的关系式为：

$$V^*(s) = \max(R_s^a + \gamma \Sigma P_s^a s^{'} V^*(s^{'})) \tag{7.10}$$

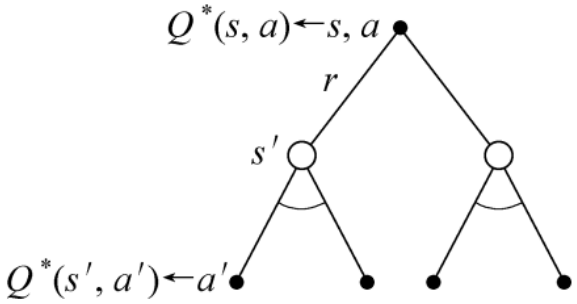


图 7.10

图7.10为两个图7.7加到图7.8下，得到最优行为值函数与后继最优行为值函数的关系式为：

$$Q^*(s,a) = R_s^a + \gamma \Sigma P_s^a s^{'} \max Q^*(s^{'}, a^{'}) \tag{7.11}$$

最优策略

定义：
对于任何状态 s，当且仅当遵循策略 π 的价值不小于遵循策略 $\pi^{'}$ 的价值时，则称策略 π 优于遵循策略 $\pi^{'}$，即：

$$\pi \geq \pi^{'} \; V_\pi(s) \geq V_{\pi'}(s) \; \forall s$$

对于任何 MDP，存在一个最优策略，即满足：$\pi^* \geq \pi \; \forall \pi$。每个策略对应一个值函数，最优策略对应最优值函数，找到最优值函数就找到了最优策略。
求解最优策略：

- 基于最优行为值函数 $\pi^*(a|s)$=$\begin{cases} 1, & a = \text{argmaxQ}^*(\text{s},\text{a}); \\ 0, & \text{其他}. \end{cases}$

- 基于最优状态值函数 $\pi^*(a|s) = \text{argmax}(\text{R}_\text{s}^\text{a} + \gamma \; \text{P}_\text{s}^\text{a}\text{s}^{'}\text{V}^*(\text{s}^{'}))$

即在得到最优行为值函数时直接 argmax 求最优策略，在得到最优状态值函数时先转换为最优行为值函数再 argmax 求最优策略。下面给出基于最优行为值函数求最优策略的例子：

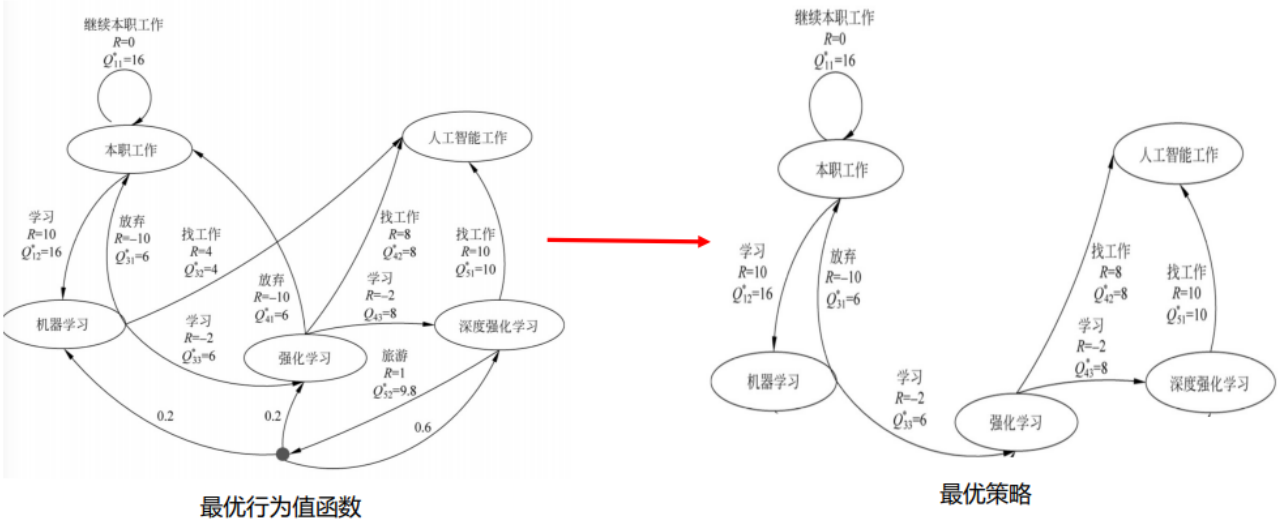


图 7.11: 最优策略

若在当前状态 s 下有 m 个行为值函数相等且取值最大，则对应的行为概率均为 $1/m$。如在本职工作状态下可采取两个动作，采取继续本职工作这一动作的最优行为值函数 $Q^*_{11}=16$，采取学习这一动作的最优行为值函数 $Q^*_{12}=16$，则在本职工作状态下采取继续本职工作和学习的动作为 1/2，此时将这两个动作都保留。在深度强化学习状态下采取找工作这一动作的最优行为值函数 $Q^*_{51}=10$，采取旅游这一动作的最优行为值函数 $Q^*_{52}=9.8$，所以保留学习这一动作，其他状态类似，通过比较最优行为值函数留下一个最优值函数最大所对应的动作，从而得到最优策略。

7.5 动态规划

虽然 MDP 可以直接用方程组来直接求解简单的问题，但是更复杂的问题却没有办法求解，因此我们还需要寻找其他有效的求解强化学习的方法。从本节开始介绍强化学习的三个基本算法：动态规划、蒙特卡洛、时间差分。

7.5.1 动态规划简介

是运筹学分支，求解决策过程最优化的决策方法，由贝尔曼提出。基本思想：就是将待求解问题分解成若干子问题，通过解决子问题，将子问题的解进行累积，来解决原问题。

问题特征：

- **最优子结构**：复杂问题的最优解由数个子问题的最优解构成，通过寻找子问题的最优解得到复杂问题的最优解。（如最短路径：$A\to B\to C$，$A\to C$ 的最短路径由 $A\to B$ 的最短路径和 $B\to C$ 的最短路径结合得到）
- **子问题重叠**：子问题在复杂问题内重复出现，可将子问题的解存储起来，重复利用。

马尔可夫决策过程 (MDP) 具备上述两个特性：由贝尔曼方程可以看出

- 贝尔曼方程把问题递归为求解子问题
- 价值函数相当于存储了一些子问题的解

故马尔可夫决策过程可由动态规划求解。

7.5.2 动态规划解决强化学习问题

条件：需要知道 MDP 的所有元素，尤其是环境模型 (状态转移概率、回报)。
动态规划是一个有模型的方法，需要知道 MDP 模型即五元组 $< S, A, P, R, \gamma >$。
步骤：使用规划方法进行策略评估和策略改进

- **策略评估 (预测)** : 给定一个马尔可夫决策过程模型 MDP: $< S, A, P, R, \gamma >$ 和一个策略 π, 要求输出基于当前策略 π 的所有状态的值函数 V_π 。
- **策略改进 (控制)** : 给定一个马尔可夫决策过程模型 MDP: $< S, A, P, R, \gamma >$ 和一个策略 π, 要求确定最优值函数 V^* 和最优策略 π^* 。

策略评估求解预测问题

策略评估的基本思路是从任意一个状态价值函数开始，依据给定的策略，结合贝尔曼期望方程、状态转移概率和奖励同步迭代更新状态价值函数，直至其收敛，得到该策略下最终的状态价值函数。
方法: 贝尔曼期望方程可以通过后继状态 $s^{'}$ 更新状态 s。

$$V_\pi(s) = \Sigma\pi(a|s)(R_s^a + \gamma\Sigma P_s^a{}_{s^{'}} V_\pi(s^{'})) \tag{7.12}$$

由贝尔曼期望方程可知，初始时下一时刻的状态值函数并不知道，我们初始所有状态值函数全部为 0 。第 $k+1$ 次迭代求解时，使用第 k 次计算出来的值函数 $V_k(s^{'})$ 更新计算 $V_{k+1}(s)$，迭代公式为：

$$V_k(s) = \Sigma\pi(a|s)(R_s^a + \gamma\Sigma P_s^a{}_{s^{'}} V_{k+1}(s^{'})) \tag{7.13}$$

该式和式7.12唯一的区别是由于我们的策略 π 已经给定，我们不再写出，对应加上了迭代轮数的下标。我们每一轮可以对计算得到的新的状态价值函数再次进行迭代，直至状态价值的值改变很小 (收敛)，那么我们就得出了预测问题的解，即给定策略的状态价值函数。

策略改进求解控制问题

目的：利用已经求得的值函数 V_π，找到最优策略 $\pi^{'}$。

方法：基于求得的 V_π ，针对每个状态采用贪心方法对策略进行改进。所谓贪心方法即利用贪心算法选取行为, 直接将所选择的动作改变为当前最优的动作，也就是在当前状态下选取使得到达后继状态时值函数最大的动作。

$$\pi^{'} = \mathrm{greedyV}_\pi \text{ 等价于 } a = \mathrm{argmaxQ}_\pi(\mathrm{s,a})$$

策略评估与策略改进可以进行不同程度的组合（组合方式不同）让策略评估和策略改进交替运行，产生不同的算法：策略迭代和值迭代。

策略迭代

其中策略迭代分为两步：第一步是使用当前的策略，依据评估得到当前策略的最终价值函数，第二步是根据状态价值通过一定的方法（比如贪婪法）更新策略，接着回到第一步，一直迭代下去，最终得到收敛的策略和状态价值，具体的迭代过程如图7.12所示：

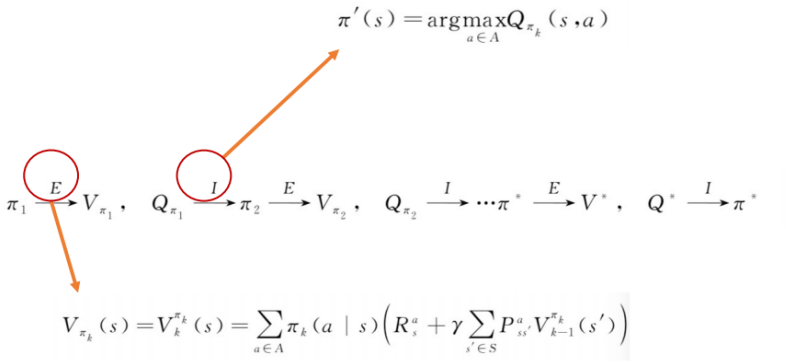


图 7.12: 策略迭代过程

策略迭代的算法如表7.1所示：

输入：MDP 五元组 $M=<S,A,P,R,\gamma>$
初始化值函数 $V(s)=0$，初始化策略 π_1 为随机策略
For $k=0,1,2,3...do$
$\forall s\in S^{'}: V_{k+1}(s)=\Sigma_{a\in A}\pi(a\|s)(R_s^a+\gamma\Sigma_{s'\in S}P_s^a s^{'}V_k(s^{'}))$
if $max(\|V_{k+1}-V_k\|<\theta)$
end if
end for
$\forall s\in S^{'}$:$\pi^{'}(s)=\text{argmax}_{\text{a}\in\text{A}}\text{Q(s,a)}$
or $\pi^{'}(s)=\text{argmax}_{\text{a}\in\text{A}}(\text{R}_\text{s}^\text{a}+\gamma_{\text{ s}\in\text{S}}\text{P}_\text{s}^\text{a}\text{s}^{'}\text{V(s}^{'}\text{)})$
if $\forall s:\pi^{'}(s)=\pi(s)$ then
break
else
$\pi=\pi^{'}$
end if
输出: 最优策略 π

表 7.1: 策略迭代的算法

一部分是策略评估，一部分是策略改进，策略评估到 V 收敛，策略改进用的是贪心算法，一次评估伴随着一次改进不断地循环至 π 收敛时（策略不发生变化）得到最优的策略 π。

例 1：网格世界寻宝

- 状态空间为 $S=0,1,...,24$, 状态 8 为宝藏区，动作空间 $A=$ UP, RIGHT, DOWN,LEFT , 宝藏区回报为 $r=0$, 其他位置回报为 $r=-1$ 。
- 状态转移概率 $P_a^s s^{'}=1$, 折扣因子 $\gamma=1$。
- 初始策略为随机策略即选择上下左右这四个动作的概率均为 1/4。

目的：找到一个最优策略 (动作集合)，随便一个位置（状态）我能找到宝藏的最优路径是什么。

0	1	2	3	4
5	6	7	宝藏	9
10	11	12	13	14
15	16	17	18	19
20	21	22	23	24

利用贝尔曼期望方程中的式7.10可进行值函数的迭代求解，代码如下所示：

```
import numpy as np
from lib.envs.gridworld import GridworldEnv
import copy
import gym
#策略评估
def policy_eval(policy,env,discount_factor=1.0,threshold=0.00001):
    #初始化各状态的状态值函数
    V = np.zeros(env.nS)
    V_pre = np.zeros(env.nS)
    cishu = 0
    while True:
        value_delta = 0
        #遍历各状态
        for s in range(env.nS):
            v = 0
            #遍历各行为的概率(上,右,下,左)
            for a, action_prob in enumerate(policy[s]):
            #对于每个行为确认下个状态
#参数:prob:概率,next_state:下一个状态的索引,reward:回报,done:是否是终止状态
             for prob, next_state, reward, done in env.P[s][a]:
             #使用贝尔曼期望方程进行状态值函数的求解
             v+=action_prob*(reward+discount_factor*prob*V_pre[next_state])
            #求出各状态和上一次求得状态的最大差值
            value_delta=max(value_delta,np.abs(v-V_pre[s]))
            V[s]=v
        V_pre=copy.deepcopy(V)
        cishu+=1
        print(V.reshape(5,5),cishu)
        #当前循环得出的各状态和上一次状态的最大差值小于阈值,则收敛停止运算
        if value_delta < threshold:
```

```
            break
    return np.array(V)
#策略改进
def policy_improvement(v, policy, discount_factor=1.0):
    def get_max_index(action_values):
        indexs = []
        policy_arr = np.zeros(len(action_values))
        action_max_value = np.max(action_values)
        for i in range(len(action_values)):
            action_value = action_values[i]
            if action_value == action_max_value:
                indexs.append(i)
                policy_arr[i] = 1
        return indexs, policy_arr
    #定义一个标识来证明本次是否有提升
    policy_stable = True
    #遍历各状态
    for s in range(env.nS):
        #取出当前状态下最优行为的索引值
        chosen_a = np.argmax(policy[s])
        #初始化行为数组[0,0,0,0]
        action_values = np.zeros(env.nA)
        for a in range(env.nA):
            #遍历各行为
            for prob, next_state, reward, done in env.P[s][a]:
            #根据各状态值函数求出行为值函数
            action_values[a]+=(reward+discount_factor*prob*v[next_state])
        best_a_arr, policy_arr = get_max_index(action_values)
       #如果求出的最大行为值函数的索引没有改变,则定义当前策略未改变，收敛输出
       #否则将当前的状态中将有最大行为值函数的方向置1,其余方向置0
        if chosen_a not in best_a_arr:
            policy_stable = False
        policy[s] = policy_arr
    return policy_stable, policy
#策略迭代
def policy_iteration(env):
    policy = np.ones([env.nS, env.nA])/env.nA #初始化一个随机策略
    #直至policy_stable=FALSE则结束迭代
    while True:
        #评估当前的策略,输出为各状态的当前的状态值函数
        v=policy_eval(policy, env)
        #进行策略改进
        policy_stable, policy = policy_improvement(v, policy)
        #如果当前策略没有发生改变,即已经到了最优策略,返回
        if policy_stable:
            return policy, v
```

```
if __name__ == "__main__":
    env = GridworldEnv() #对环境进行初始化
    v = policy_iteration(env)
    print("值函数:")
    print(v.reshape(5,5))
```

k 表示迭代次数，在 k 取不同值时对应的状态值函数如下所示：

$k=0$ 时随机策略下的值函数 $V_0^\pi(s)$ 为：

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

$k=1$ 时随机策略下的值函数 $V_1^\pi(s)$ 为：

-1	-1	-1	-0.75	-1
-1	-1	-0.75	0	-0.75
-1	-1	-1	-0.75	-1
-1	-1	-1	-1	-1
-1	-1	-1	-1	-1

$k=2$ 时随机策略下的值函数 $V_2^\pi(s)$ 为：

-2	-2	-1.88	-1.44	-1.88
-2	-1.94	-1.5	0	-1.44
-2	-2	-1.88	-1.5	-1.88
-2	-2	-2	-1.94	-2
-2	-2	-2	-2	-2

$k=528$ 时值函数收敛 $V_{528}^\pi(s)$ 为：

-46.12	-40.73	-30.24	-17.62	-19.628
-47.55	-41.80	-28.38	0	-17.62
-50.70	-46.56	-38.47	-28.38	-30.24
-53.98	-51.27	-46.56	-41.80	-40.73
-55.98	-53.98	-50.70	-47.55	-46.14

针对初始策略进行策略改进，得到 π_1，对收敛的值函数 $V_{528}^\pi(s)$ 使用贪心算法进行策略改进，求取 $V_{528}^\pi(s)$ 对应的策略改进后的 π_1，则有 π_1：

→	→	→	↓	←↓
→	→	→	宝藏	←
→	→	↑→	↑	↑
↑	↑→	↑	↑	↑
↑→	→	↑	↑	↑

对 π_1 进行策略评估，经过 6 轮值函数收敛，得到 $V_6^{\pi_1}(s)$ 为：

-3	-2	-1	0	-1
-2	-1	0	0	0
-3	-2	-1	0	-1
-4	-3	-2	-1	-2
-10	-4	-3	-2	-3

对 $V_6^{\pi_1}(s)$ 使用贪心算法进行策略改进，求取 $V_6^{\pi_1}(s)$ 对应的策略改进后的 π_2，对 π_2 进行策略评估，对收敛值函数进行策略改进得到 π_3，.....，直至策略收敛，最优策略 π^* 为：

↓→	↓→	↓→	↓	←↓
→	→	→	宝藏	←
↑→	↑→	↑→	↑	←↑
↑→	↑→	↑→	↑	←↑
↑→	↑→	↑→	↑	←↑

值迭代

值迭代：借助贝尔曼最优方程，直接使用行为回报的最大值更新原来的值。使用贝尔曼最优方程进行更新:

$$V_{k+1}(s) = \max(\mathrm{R_s^a} + \gamma\, \mathrm{P_s^a s^{'} V_k(s^{'})})$$

迭代流程为：

$$V_1 \to V_1 \to V_2 \to ... \to V^*$$

注意：迭代收敛过程中，没有遵循任何策略。但是策略改进是隐含在值迭代过程中执行的。

输入：MDP 五元组 $M =< S, A, P, R, \gamma >$
初始化值函数 $V(s) = 0$，收敛阈值 θ
For $k = 0, 1, 2, 3...do$
$\forall s \in S^{'}$: $V^{'}(s) = \max(\mathrm{R_s^a} + \gamma\ \mathrm{P_s^a s^{'} V(s^{'})})$
if $\max(\lvert \mathrm{V^{'}(s) - V(s^{'})} \rvert < \theta)$ then
break
else $V = V^{'}$
end if
end for
$\forall s \in S^{'}$:$\pi^{'}(s) = \mathrm{argmax_{a \in A} Q(s, a)}$
or $\pi^{'}(s) = \mathrm{argmax_{a \in A}(R_s^a + \gamma\ _{s \in S} P_s^a s^{'} V(s^{'}))}$
if $\forall s : \pi^{'}(s) = \pi(s)$ then
break
else
$\pi = \pi^{'}$
end if
输出: 最优策略 π

表 7.2: 值迭代的算法

例二：值迭代解决网格世界寻宝问题，仍是例一的环境，利用贝尔曼最优方程进行更新值函数，代码如下所示：

```
#值迭代
def value_iteration(env,discount_factor=1.0,threshold=0.00001):
    def get_max_index(action_values):
        indexs = []
        policy_arr = np.zeros(len(action_values))
        action_max_value = np.max(action_values)
        for i in range(len(action_values)):
            action_value = action_values[i]
            if action_value == action_max_value:
                indexs.append(i)
                policy_arr[i] = 1
        return indexs, policy_arr
    def one_step_lookahead(state, v):
        q = np.zeros(env.nA)
        for a in range(env.nA):
            for prob, next_state, reward, done in env.P[state][a]:
                q[a] += (reward + discount_factor * prob * v[next_state])
        return q
    v = np.zeros(env.nS)
    while True:
```

```
        #停止条件
        delta = 0
        #遍历每个状态
        for s in range(env.nS):
            #计算当前状态的值函数
            q = one_step_lookahead(s, v)
            #找到最大值函数
            best_action_value = np.max(q)
            # 值函数更新前后求差
            delta = max(delta, np.abs(best_action_value - v[s]))
#更新当前状态的值函数，即：将最大的值函数赋值给值当前状态，用以更新当前状态的值
    函数
            v[s] = best_action_value
        #如果当前状态的值函数更新前后相差小于阈值,则说明已经收敛,结束循环
        if delta <= threshold:
            break
    #初始化策略
    policy = np.zeros([env.nS, env.nA])
    #遍历各状态
    for s in range(env.nS):
        #根据已经计算出的V,计算当前状态的各值函数
        q = one_step_lookahead(s,v)
        #求出当前最大值函数对应的动作索引
        #将初始策略中的对应的状态上将最大值函数方向置1,其余方向保持不变,仍为0
        best_a_arr, policy_arr = get_max_index(q)
        #将当前所有最优值赋值给当前状态
        policy[s] = policy_arr
    return policy,v
if __name__ == "__main__":
    env = GridworldEnv()
    v = value_iteration(env)
    print("值函数:")
    print(v)
```

k 表示迭代次数，当 k 取不同值时，基于贝尔曼最优方程得到得个状态下的值函数如下：

$k=0$ 时随机策略下的值函数 $V_0^{\pi}(s)$ 为：

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

$k=1$ 时得 $V_1^{\pi}(s)$ 为：

-1	-1	-1	-1	-1
-1	-1	-1	0	-1
-1	-1	-1	-1	-1
-1	-1	-1	-1	-1
-1	-1	-1	-1	-1

$k = 2$ 时得 $V_2^{\pi}(s)$ 为:

-2	-2	-2	-1	-2
-2	-2	-1	0	-1
-2	-2	-2	-1	-2
-2	-2	-2	-2	-2
-2	-2	-2	-2	-2

$k = 3$ 时得 $V_3^{\pi}(s)$ 为:

-3	-3	-2	-1	-2
-3	-2	-1	0	-1
-3	-3	-2	-1	-2
-3	-3	-3	-2	-3
-3	-3	-3	-3	-3

第七次之后各状态得最优行为值函数已经收敛，V_3^{π} 为:

-4	-3	-2	-1	-2
-3	-2	-1	0	-1
-4	-3	-2	-1	-2
-5	-4	-3	-2	-3
-6	-5	-4	-3	-4

对应得最优策略为:

↓→	↓→	↓→	↓	←↓
→	→	→	宝藏	←
↑→	↑→	↑→	↑	←↑
↑→	↑→	↑→	↑	←↑
↑→	↑→	↑→	↑	←↑

7.6 蒙特卡罗

上面我们讨论了用动态规划来求解强化学习预测问题和控制问题的方法。但是由于动态规划法需要在每一次更新某一个状态的价值时，回溯到该状态的所有可能的后续状态，导致对于复杂问题计算量很大；同时很多时候，我们连环境的状态转化模型 P 都无法知道，这时动态规划

法没有办法使用。这时候我们如何求解强化学习问题呢？本文要讨论的蒙特卡罗就是一种可行的方法。蒙特卡罗是在环境未知时，即智能体不知道环境的工作机制：不知道环境的状态转移概率、不知道环境奖励，智能体通过和环境交互获得的轨迹进行学习，与环境交互本质上相当于从概率分布 $P_a^s s^{'}$ 和 R_s^a 中进行采样，采样足够充分时，可以使用样本分布良好地刻画总体分布。

7.6.1 蒙特卡罗简介

蒙特 · 卡罗方法（Monte Carlo method），也称统计模拟方法，是二十世纪四十年代中期由于科学技术的发展和电子计算机的发明，而被提出的一种以概率统计理论为指导的一类非常重要的数值计算方法。是指使用随机数（或更常见的伪随机数）来解决很多计算问题的方法。在强化学习问题中使用蒙特卡罗法的随机体现在采样上，也就是随机模拟模型的分布。蒙特卡罗法通过采样若干经历完整的状态序列 (*episode*) 来估计状态的真实价值。所谓的经历完整，就是这个序列必须是达到终点的。比如下棋问题分出输赢，驾车问题成功到达终点或者失败。有了很多组这样经历完整的状态序列，我们就可以来近似的估计状态价值，进而求解预测和控制问题了。
蒙特卡罗法和和动态规划比：一、它不需要依赖于模型状态转化概率; 二、完整的经历越多，学习效果越好。下面用一张图解释动态规划与蒙特卡罗的区别：
动态规划是基于贝尔曼方程进行计算当前状态的值函数，需要知道所有后继状态的值函数。

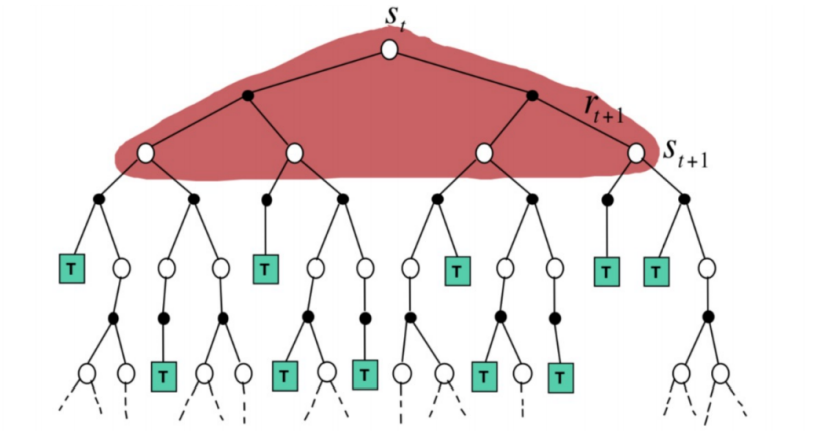


图 7.13: 动态规划求值函数

当不知道所有后继状态的值函数时，蒙特卡罗最简单的思路是通过不断的采样，然后统计平均回报值来估计值函数。

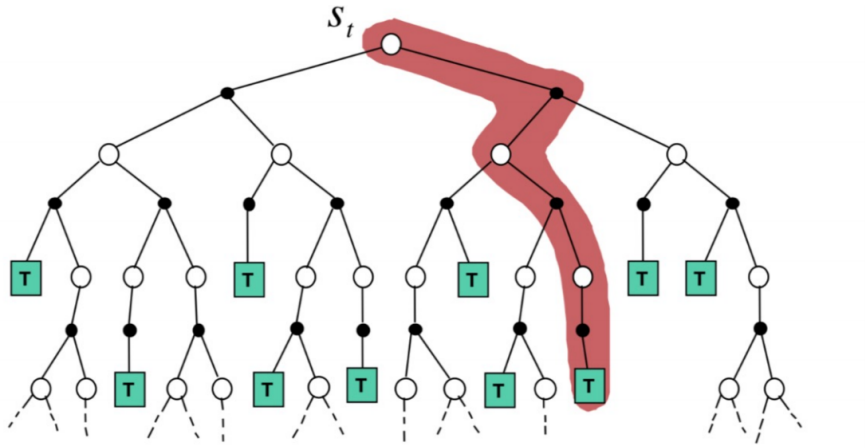


图 7.14: 蒙特卡罗求值函数

7.6.2 蒙特卡罗评估

从值函数的定义来看:

$$V_\pi(s) = E(G_t|s_t = s) = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots \tag{7.14}$$

每个状态的值函数等于所有该状态累积回报的期望。那么对于蒙特卡罗法来说，如果要求某一个状态的值函数，只需要求出所有的完整序列中该状态出现时候的收获累积回报再取平均值即可近似求解，也就是:

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots \tag{7.15}$$

$$V_\pi(s) \approx \text{average}(\text{G}_\text{t}) \tag{7.16}$$

蒙特卡罗法的计算思路就是这样，但是有两个问题需要说明:

1. 同样一个状态可能在一个完整的状态序列中重复出现，那么该状态的回报该如何计算？有两种解决方法。第一种是仅把状态序列中第一次出现该状态时的收获值纳入到收获平均值的计算中；另一种是针对一个状态序列中每次出现的该状态，都计算对应的收获值并纳入到收获平均值的计算中。两种方法对应的蒙特卡罗法分别称为：首访法和每访法。第二种方法比第一种的计算量要大一些，但是在完整的经历样本序列少的场景下第一种方法更适用。

   首访法: 每一条轨迹中，只记录状态 s 首次出现对应的回报.

   轨迹一: $< s_1, a_1, G_{11}, s_2, a_2, G_{12}, \dots, s_k, a_k, G_{1k}, \dots, s_t, a_t, G_{1t} >$

   轨迹二: $< s_3, a_1, G_{21}, s_1, a_2, G_{22}, \dots, s_1, a_k, G_{2k}, \dots, s_t, a_t, G_{2t} >$

   $$V(s_1) = (G_{11} + G_{22} + \dots)/N(s_1) \tag{7.17}$$

   每访法: 每一条轨迹，记录状态 s 出现每次出现对应的回报.

   仍以上面的两个轨迹为例:

   $$V(s_1) = (G_{11} + G_{22} + G_{2k} + \dots)/N(s_1) \tag{7.18}$$

2. 在上面预测问题的求解公式里，我们有一个平均值的公式，意味着要保存所有该状态的收获值之和最后取平均。这样浪费了太多的存储空间。一个较好的方法是在迭代计算收获均值，即每次保存上一轮迭代得到的收获均值，当计算得到当前轮的收获时，即可计算当前轮收获均值。对于状态 s_t ，基于 k 次采样数据估计其值函数:

$$\begin{aligned} V_k &= 1/k\sum_{j=1}^{k} G_j = 1/k(G_k + \sum_{j=1}^{k-1} G_j) \\ &= 1/k(G_k + (k-1)V_{k-1}) \\ &= V_{k-1} + 1/k(G_k - V_{k-1}) \end{aligned} \tag{7.19}$$

k 次采样数据估计值 $=k-1$ 次采样估计值 $+$ 一个增量, 上一时刻的平均值和这一时刻的平均值建立联系。

蒙特卡罗方法值函数估计的更新公式：

$$V_{s_t} = V_{s_t} + \alpha(G_t - V_{s_t}) \tag{7.20}$$

在更新公式中用 $Q_\pi(s,a)$ 代替 V_s，原因是：通过 $Q_\pi(s,a)$ 求 π 更直接, 还有就是在模型未知的情况下无法通过 $V_\Pi(S)$ 求 π.

替换之后的 MC Q 值迭代公式为:

$$Q(s_t,a_t) = Q(s_t,a_t) + \alpha(G_t - Q(s_t,a_t)) \tag{7.21}$$

在状态 s_t 时采取行为 a_t 得到的目标值（累积回报）与当前值（上一次采样得到的行为值函数）的差值，给一个学习率去更新当前值。需注意这里的累积回报 G_t 当用其他值代替时即为 TD（sarsa Q-learning）后面会讲。

7.6.3 蒙特卡罗改进

蒙特卡罗法求解控制问题的思路和动态规划价值迭代的的思路类似。回忆下动态规划价值迭代的的思路，每轮迭代先做策略评估，计算出价值 $V_k(s)$，然后基于据一定的方法（比如贪婪法）更新当前策略 π。最后得到最优价值函数 V^* 和最优策略 π^*。

和动态规划比，蒙特卡罗法不同之处体现在三点：一是预测问题策略评估的方法不同，这个上面已经讲了。第二是蒙特卡罗法一般是优化最优动作价值函数 Q。三是动态规划一般基于贪婪法更新策略, 而蒙特卡罗法一般采用 ϵ 贪婪法更新,ϵ 贪心法通过设置一个较小的 ϵ 值，使用 $1-\epsilon$ 的概率贪婪地选择目前认为是最大行为价值的行为，而用 ϵ 的概率随机的从所有 m 个可选行为中选择行为。这里相比于贪心算法增加了一个随机性，有可能那些没有使 Q 达到最大行为也会得到一个好的结果，ϵ 贪心法就是增加一个随机性有 ϵ 的概率随机选择. 所谓的 ϵ 贪心法是大概率选择使值函数最大的行为，小概率随机选择，结果证明该方法能改进任一策略，即使用该方法之后得到的策略对应的值函数比之前的值函数要大。

$$a=\begin{cases} \text{argmaxQ(s,a)}, & \text{1-}\epsilon; \\ random, & \epsilon. \end{cases} \quad \pi(a_t|s_t)=\begin{cases} \text{1-}\epsilon+\epsilon/\text{m}, & \text{a=}argmax_{a\in A}Q(s,a); \\ \epsilon/\text{m}, & a \neq argmax_{a\in A}Q(s,a). \end{cases}$$

第 8 章　强化学习（二）

8.1　时序差分

8.1.1　时序差分简介

上一章介绍的动态规划算法要求马尔可夫决策过程是已知的，即要求与智能体交互的环境是完全已知的。在此条件下，智能体其实并不需要和环境真正交互来采样数据，直接用动态规划算法就可以解出最优价值或策略。这就好比对于有监督学习任务，如果直接显式给出了数据的分布公式，那么也可以通过在期望层面上直接最小化模型的泛化误差来更新模型参数，并不需要采样任何数据点。但这在大部分场景下并不现实，机器学习的主要方法都是在数据分布未知的情况下针对具体的数据点来对模型做出更新的。对于大部分强化学习现实场景（例如电子游戏或者一些复杂物理环境），其马尔可夫决策过程的状态转移概率是无法写出来的，也就无法直接进行动态规划。在这种情况下，智能体只能和环境进行交互，通过采样到的数据来学习，这类学习方法统称为无模型的强化学习（model-free reinforcement learning）。

不同于动态规划算法，无模型的强化学习算法不需要事先知道环境的奖励函数和状态转移函数，而是直接使用和环境交互的过程中采样到的数据来学习，这使得它可以被应用到一些简单的实际场景中。本章将要讲解无模型的强化学习中的两大经典算法：Sarsa 和 Q-learning，它们都是基于时序差分（temporal difference，TD）的强化学习算法。

时序差分是一种用来估计一个策略的价值函数的方法，它结合了蒙特卡洛和动态规划算法的思想。时序差分方法和蒙特卡洛的相似之处在于可以从样本数据中学习，不需要事先知道环境；和动态规划的相似之处在于根据贝尔曼方程的思想，利用后续状态的价值估计来更新当前状态的价值估计。回顾一下蒙特卡洛方法对价值函数的增量更新方式:

$$V\left(s_t\right) \leftarrow V\left(s_t\right)+\alpha\left[G_t-V\left(s_t\right)\right]$$

这里 α 表示对价值估计更新的步长。可以将 α 取为一个常数，此时更新方式不再像蒙特卡洛方法那样严格地取期望。蒙特卡洛方法必须要等整个序列结束之后才能计算得到这一次的回报 G_t，而时序差分方法只需要当前步结束即可进行计算。具体来说，时序差分算法用当前获得的奖励加上下一个状态的价值估计来作为在当前状态会获得的回报，即：

$$V\left(s_t\right) \leftarrow V\left(s_t\right)+\alpha\left[r_t+\gamma V\left(s_{t+1}\right)-V\left(s_t\right)\right]$$

其中 $R_t+\gamma V\left(s_{t+1}\right)-V\left(s_t\right)$ 通常被称为时序差分 (temporal difference，TD) 误差（error），时序差分算法将其与步长的乘积作为状态价值的更新量。可以用 $r_t+\gamma V\left(s_{t+1}\right)$ 来代替 G_t 的原因

是:

$$
\begin{aligned}
V_\pi(s) &= \mathbb{E}_\pi\left[G_t \mid S_t = s\right] \\
&= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k} \mid S_t = s\right] \\
&= \mathbb{E}_\pi\left[R_t + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s\right] \\
&= \mathbb{E}_\pi\left[R_t + \gamma V_\pi\left(S_{t+1}\right) \mid S_t = s\right]
\end{aligned}
$$

因此蒙特卡洛方法将上式第一行作为更新的目标，而时序差分算法将上式最后一行作为更新的目标。于是，在用策略和环境交互时，每采样一步，我们就可以用时序差分算法来更新状态价值估计。时序差分算法用到了 $V\left(s_{t+1}\right)$ 的估计值，可以证明它最终收敛到策略 π 的价值函数，我们在此不对此进行展开说明。

蒙特卡罗强化学习：学习完整的采样轨迹，更新值函数和改进策略, 学习效率很低。

动态规划强化学习：采用自举 (bootstapping) 的方法，用后继状态的值函数估计当前状态值函数, 可以每一时间步更新一次，效率较高。

时间差分强化学习：充分结合动态规划的自举和蒙特卡罗的采样，通过学习后继状态的值函数来逼近当前状态值函数，实现对不完整轨迹的学习。要点：

- 时间差分和蒙特卡罗一样，它也需要采样，需要从轨迹中学习。
- 不需要了解模型本身，属于无模型方法。
- 可以学习不完整的轨迹。

8.1.2 三种方法的性质对比

接下来从值函数估计方式、偏差与方差、马尔可夫属性等方面对时序差分、动态规划和蒙特卡罗三种方法进行对比。

(1) 值函数估计

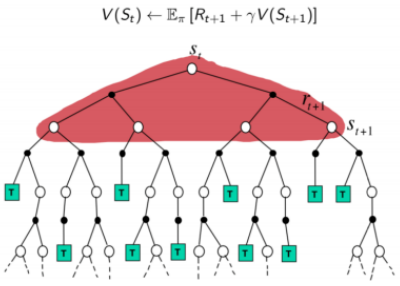


图 8.1: DP 方法

DP：没有采样。根据环境模型，获得状态 S 所有可能的转移状态 S'、转移概率、即时奖励来计算当前状态 S 的值函数。

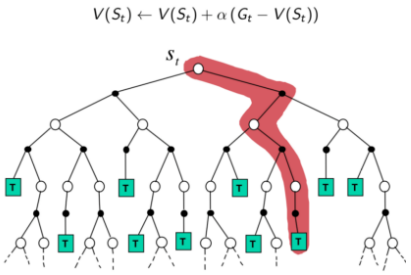


图 8.2: MC 方法

MC: 采样，获得完整轨迹。用实际回报更新当前状态值函数。

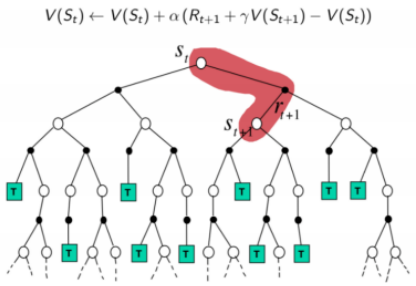


图 8.3: TD 方法

TD：采样，轨迹可不完整。用下一状态的值函数更新当前状态值函数。

表 8.1: 时间差分、动态规划和蒙特卡洛三种方法

方法	值函数估计	是否自举	是否采样
DP	$V_\pi\left(S_t\right) = E_\pi\left[R_{t+1} + \gamma V\left(S_{t+1}\right) \mid S_t = s\right]$	自举	无须采样
MC	$V_\pi\left(S_t\right) \approx G_t \mid S_t = s$	不自举	采样, 完整轨迹
TD	$V_\pi\left(S_t\right) \approx R_{t+1} + \gamma V\left(S_{t+1}\right) \mid S_t = s$	自举	采样, 不完整轨迹

(2) 偏差/方差

蒙特卡罗 (MC) 和时序差分 (TD) 均是利用样本去估计值函数，可以从统计学的角度来对比两种方法的期望和方差两个指标。

蒙特卡罗在估计值函数时，使用的是累积回报 G_t 的平均值。G_t 期望便是值函数的定义，因此蒙特卡罗方法是无偏估计。

蒙特卡罗在计算 G_t 值时，需要计算从当前状态到最终状态之间所有的回报，在这个过程中要经历很多随机的状态和动作，因此每次得到的随机性很大。所以尽管期望等于真值，但方差无穷大。

时序差分方法使用 $R_{t+1} + \gamma V(S_{t+1})$(也叫 TD 目标) 估计值函数，若 TD 目标采用真实值，是基于下一状态的实际价值对当前状态实际价值进行估计，则 TD 估计也是无偏估计。然而在实际中 TD 目标用的是估计值，即基于下一状态预估值函数计算当前预估值函数，因此时序差

分估计属于有偏估计。跟蒙特卡罗相比，时序差分只用到了一步随机状态和动作，因此 TD 目标的随机性比蒙特卡罗方法中的 G_t 要小，其方差也比蒙特卡罗方法的方差小。

动态规划方法利用模型计算所有后继状态，借助贝尔曼方程，利用后继状态得到当前状态的真实值函数，不存在偏差和方差，三种方法的偏差和方差对比见下表。

表 8.2: 三种方法的偏差和方差对比

方法	偏差	方差
DP	无偏差	无方差
MC	无偏差	高方差
TD	无偏 (真实 TD 目标)　有偏 (预估 TD 目标)	低方差

(3) 马尔可夫性

动态规划是基于模型的方法，基于现有的一个马尔可夫决策模型 MDP 的状态转移概率和回报，求解当前状态的值函数，因此该方法具有马尔可夫性。

蒙特卡罗和时序差分方法都是无模型方法，都需要通过学习采样轨迹估计当前状态值函数。所不同的是，应用时序差分 (TD) 算法时，时序差分算法试图利用现有的轨迹构建一个最大可能性的马尔可夫决策模型。即首先根据已有经验估计状态间的转移概率，同时估计一个状态的立即回报，最后计算该马尔可夫决策模型的状态值函数。蒙特卡罗算法并不试图构建马尔可夫决策模型，该算法试图最小化状态值函数与累计回报的均方误差。

通过比较可以看出，时序差分和动态规划均使用了马尔可夫决策模型问题的马尔可夫属性，在马尔可夫环境下更有效；但蒙特卡罗方法并不利用马尔可夫属性，通常在非马尔可夫环境下更有效。

表 8.3: 三种方法马林科夫性对比

方法	是否使用马尔可夫属性
DP	是
MC	否
TD	是

8.1.3 Sarsa：在线策略 TD

对于它的控制问题求解，和蒙特卡罗法类似，都是价值迭代，即通过价值函数的更新，来更新当前的策略，再通过新的策略，来产生新的状态和即时奖励，进而更新价值函数。一直进行下去，直到价值函数和策略都收敛。

时序差分法的控制问题，可以分为两类，一类是在线控制，即一直使用一个策略来更新价值函数和选择新的动作。而另一类是离线控制，会使用两个控制策略，一个策略用于选择新的动作，另一个策略用于更新价值函数。

我们的 Sarsa 算法，属于在线控制这一类，即一直使用一个策略来更新价值函数和选择新的动作，而这个策略是 ϵ 贪婪法，即通过设置一个较小的 ϵ 值，使用 ϵ 的概率贪婪地选择目前认

为是最大行为价值的行为，而用 ϵ 的概率随机的从所有 m 个可选行为中选择行为。用公式可以表示为：

$$\pi(a|s) = \begin{cases} \epsilon/m + 1 - \epsilon & \text{if} \quad a^* = argmax_{a \in A} Q(s,a) \\ \epsilon/m & \text{else} \end{cases} \tag{8.1}$$

作为 Sarsa 算法的名字本身来说，它实际上是由 S,A,R,S,A 几个字母组成的。而 S,A,R 分别代表状态（State），动作 (Action), 奖励 (Reward)，这也是我们前面一直在使用的符号。这个流程体现在下图：

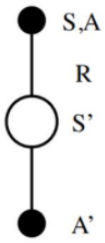


图 8.4: Sarsa 的名字来源及流程

在迭代的时候, 我们首先基于 ϵ 贪婪法在当前状态 S 选择一个动作 A, 这样系统会转到一个新的状态 S', 同时给我们一个即时奖励 R, 在新的状态 S', 我们会基于 ϵ 贪妥法在状态 S' 选择一个动作 A', 但是注意这时候我们并不执行这个动作 A', 只是用来更新的我们的行为值函数, 行为值函数的更新公式如下:

$$Q(S,A) = Q(S,A) + \alpha\left(R + \gamma Q\left(S', A'\right) - Q(S,A)\right) \tag{8.2}$$

其中, γ 是衰减因子, α 是迭代步长。这里和蒙特卡罗法求解在线控制问题的迭代公式的区别主要是, 收获 G_t 的表达式不同, 对于时序差分，收获 G_t 的表达式是 $R + \gamma Q\left(S', A'\right)$。

除了收获 G_t 的表达式不同, Sarsa 算法和蒙特卡罗在线控制算法基本类似。算法流程如下：

算法：Sarsa 算法

输入：环境 E，状态空间 S，动作空间 A，折扣回报 γ，初始化行为值函数 $Q(s,a)=0$，$\pi(a|s)=\frac{1}{|A|}$

For $k=0,1,\cdots,m$ do(针对每一条轨迹)
　初始化状态 s
　在 E 中通过 π 的 ε-贪心策略采取行为 a，得到第一个状态行为对(s,a)
　For $t=0,1,2,3\cdots$ do(针对轨迹中的每一步)
　　$r,s'=$在 E 中执行动作 a 产生的回报和转移的状态；
　　基于 s'，通过 π 的 ε-贪心策略采取行为 a'，得到第二个状态行为对(s',a')
　　更新(s,a)的 Q 值
　　$Q(s,a) \leftarrow Q(s,a)+\alpha(r+\gamma Q(s',a')-Q(s,a))$；
　　$s \leftarrow s', a \leftarrow a'$
　end for　s 是一个终止状态
$\forall s_t \in S'$：
　　$\pi(s_t) = \underset{a_t \in A}{\text{argmax}} Q(s_t, a_t)$
　end for

输出：最优策略 π

行为策略为 ε-贪心策略

目标策略均为ε-贪心策略

8.1.4 Q-learning：离线策略 TD 算法

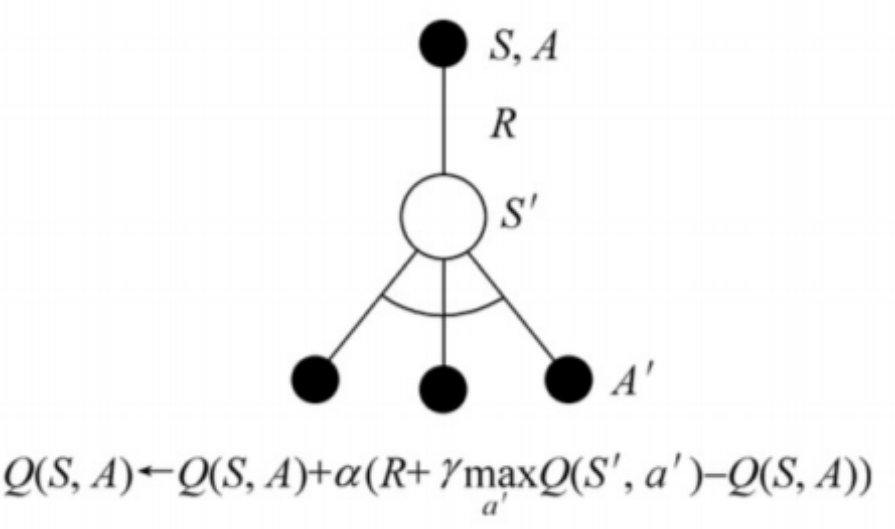


图 8.5: Q-learning 公式及图解

Q-learning 的问题求解不需要环境的状态转化模型，是不基于模型的强化学习问题求解方法。对于它的控制问题求解，和蒙特卡罗法类似，都是价值迭代，即通过价值函数的更新，来更新策略，通过策略来产生新的状态和即时奖励，进而更新价值函数。一直进行下去，直到价值函数和策略都收敛。

时序差分法的控制问题，可以分为两类，一类是在线控制，即一直使用一个策略来更新价值函数和选择新的动作，比如我们上面讲到的 Sarsa, 而另一类是离线控制，会使用两个控制策略，一个策略用于选择新的动作，另一个策略用于更新价值函数。这一类的经典算法就是 Q-learning。

对于 Q-learning，我们会使用 ϵ 贪婪法来选择新的动作，这部分和 Sarsa 完全相同。但是对于价值函数的更新，Q-learning 使用的是贪婪法，而不是 Sarsa 的 ϵ 贪婪法。这一点就是 Sarsa 和 Q-learning 本质的区别。

证明: 使用贪心策略可以改进任意一个给定的策略

$\pi'(s) = \arg\max_a Q^{\pi}(s,a)$

$V^{\pi'}(s) \geq V^{\pi}(s)$, for all state s

$V^{\pi}(s) = Q^{\pi}(s,\pi(s)) \quad \leq \max_a Q^{\pi}(s,a) = Q^{\pi}(s,\pi'(s))$

$$
\begin{aligned}
V^{\pi}(s) \quad & \leq Q^{\pi}(s,\pi'(s)) \\
& = E\left[r_{t+1} + V^{\pi}(s_{t+1}) \mid s_t = s, a_t = \pi'(s_t)\right] \\
& \leq E\left[r_{t+1} + Q^{\pi}(s_{t+1}, \pi'(s_{t+1})) \mid s_t = s, a_t = \pi'(s_t)\right] \\
& = E\left[r_{t+1} + r_{t+2} + V^{\pi}(s_{t+2}) \mid \ldots\right] \\
& \leq E\left[r_{t+1} + r_{t+2} + Q^{\pi}(s_{t+2}, \pi'(s_{t+2})) \mid \ldots\right] \ldots \leq V^{\pi'}(s)
\end{aligned}
$$

用待评估策略 (目标策略) 产生的下一个状态行为对的 Q 值, 来更新行为策略。值函数更新

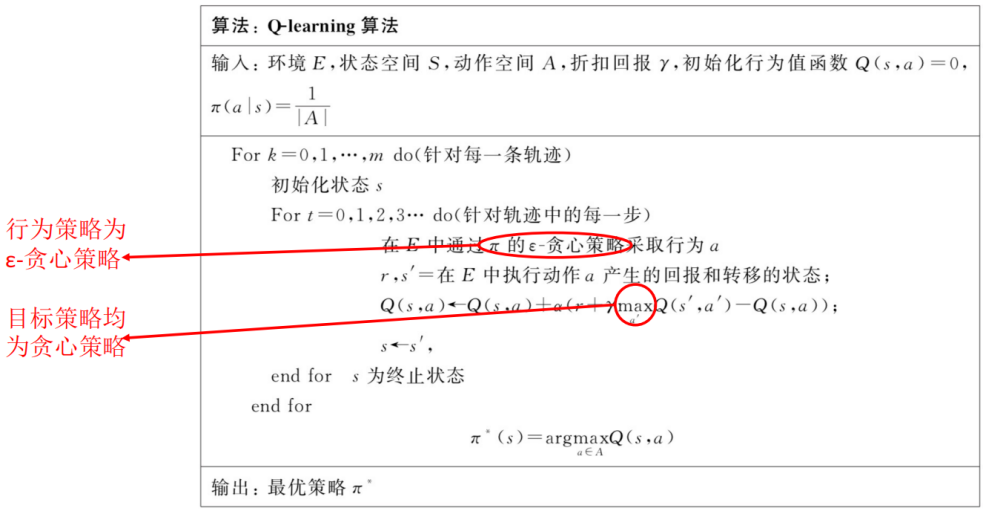
公式:

$$\pi\left(S_{t+1}\right)=\operatorname{argmax} Q\left(S_{t+1}, a'\right) \tag{8.3}$$

有

$$R_{t+1}+\gamma Q\left(S_{t+1}, A'\right)=R_{e+1}+\gamma Q\left(S_{Ht}, \operatorname{argmax} Q\left(S_{H+1}, a'\right)\right)=R_{t+1}+\max \gamma Q\left(S_{t+1}, a'\right) \tag{8.4}$$

算法流程如下：



算法：Q-learning 算法

输入：环境 E，状态空间 S，动作空间 A，折扣回报 γ，初始化行为值函数 $Q(s,a)=0$，$\pi(a|s)=\frac{1}{|A|}$

For $k=0,1,\cdots,m$ do(针对每一条轨迹)
　初始化状态 s
　For $t=0,1,2,3\cdots$ do(针对轨迹中的每一步)
　　在 E 中通过 π 的 ε-贪心策略采取行为 a
　　$r,s'=$在 E 中执行动作 a 产生的回报和转移的状态；
　　$Q(s,a)\leftarrow Q(s,a)+\alpha(r+\gamma\max_{a'}Q(s',a')-Q(s,a))$；
　　$s\leftarrow s'$，
　end for　s 为终止状态
end for

$$\pi^*(s)=\operatorname*{argmax}_{a\in A} Q(s,a)$$

输出：最优策略 π^*

8.2 资格迹

8.2.1 资格迹简介

蒙特卡罗方法和时序差分方法，这两种算法之间存在一个关键的不同点：更新当前状态的值函数时，基于当前状态往未来看的距离不同。在蒙特卡罗算法中，这个距离是整个轨迹的长度，记为 N；而在一步时序差分方法中，这个距离是 1(单位是时间步)。那么在 1 ~ N 中，就有很多可以选择的距离 d，使得 1≤d≤N。通过利用这些不同距离，构造出了新的算法类型——多步时序差分法 (也称资格迹法)。

关于多步时序差分法存在两种视角。

一种是前向视角，向前看，即由当前状态出发向还未访问的状态观察设计的一种算法。前向视角也叫理论视角，它认为资格迹是连接时序差分方法和蒙特卡罗方法的桥梁。当 n=1 步，资格迹法退化为一步时序差分法；当 n≥N 步，资格迹法发展为蒙特卡罗法；当 1 < n < N 步，则产生了一系列介于时序差分和蒙特卡罗两者中间的多步时序差分方法。我们可以采用不同 n 值的线性组合来对参数进行更新，只要它们的权重值和为 1。

另一种是后向视角，向后看，即由当前状态向已经访问过的状态观察设计的一种算法。本章引入资格迹 (Eligibility Traces) 来对两种视角进行解释。资格迹是进行资格分配 (信用分配) 的方法，它是强化学习的一项基本机制。TD(A) 算法中的 x 就是对资格迹的运用。几乎所有的 TD 算法，包括 Q-learning 方法、Sarsa 方法，都可以结合资格迹来提升效率。后向视角也叫工程视角，它认为资格迹是事件发生的临时记录，如访问某个状态或采取某个行动。它为轨迹中每个状

态 (或状态行为对) 附加一个属性，这个属性决定了该状态 (或状态行为对) 与当前正访问的状态 (或状态行为对) 的值函数更新量之间的关联程度，或者说影响程度。当目标误差 (TD 误差) 产生时，只有有资格的状态行为才能被分配回报或者惩罚。

虽然两种算法的表述不一样，但在本质上是统一的。前向视角告诉我们资格迹在理论层面是如何工作的；后向视角告诉我们资格迹在工程层面是如何实现的。实际中，因为前向算法计算量较大，一般都采用后向算法实现。

8.2.2 多步 TD 评估

使用蒙特卡罗算法来估计值函数:

$V(s_t) \leftarrow V(s_t) + \alpha(G_t - V(s_t))$

其中，更新目标 G_t 为累积回报:

$G_t = r_{t+1} + \gamma_{r_{t+2}} + \gamma^2 r_{t+3} + \cdots + \gamma^{T-t+1} r_T$

用一步 TD 估计值函数 V(s_t) 时，更新目标为一步回报:

$G_t^1 = r_{t+1} + \gamma V(s_{t+1})$

两步 TD 的更新目标为两步回报:

$G_t^2 = r_{t+1} + \gamma r_{t+2} + \gamma^2 V(s_{t+2})$

n 步 TD 的更新目标为 n 步回报

$G_t^n = r_{t+1} + \gamma_{r_{t+2}} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n V(s_{t+n})$

使用 n 步 TD 方法估计值函数时的更新公式为

$V(s_t) \leftarrow V(s_t) + \alpha(G_t^n - V(s_t))$

- $d = 1$：一步时序差分方法，TD(1) 如 sarsa，Q-learning
- $d \geq N$（N 为轨迹长度）：蒙特卡罗算法
- $1 < d < N$：多步时序差分法

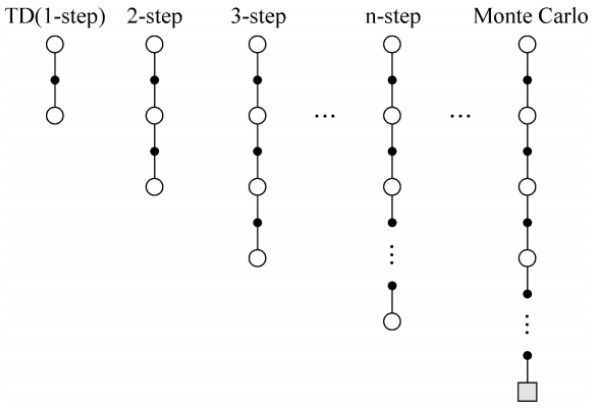


图 8.6: n 步预测估计值函数

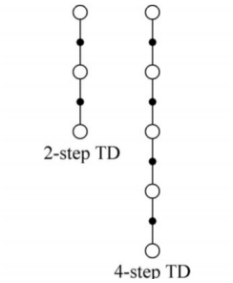
8.2.3 前向算法

问题: 既然存在 n-步时序差分方法，那么 $n=$? 时效果最好呢?

一种最简单的方法是通过平均多个不同的 n 步回报进行更新，即给每个回报赋予一定的权值，并确保这些权值的和为 1 。

比如选择 2 步 TD 和 4 步 TD 算法相结合。在每次状态更新时，2 步算法的回报占 1/2 权重，4 步算法的回报占 1/2 权重，然后求加权和。将 $\frac{1}{2}G_t^2+\frac{1}{2}G_4^t$ 作为最终回报.

实际操作: TD(λ) 直接平均了所有的 n 步回报，每个回报的权重是 $(1-\lambda)\lambda^{n-1}$ 。通过引入这个新的参数 λ, 综合考虑所有步数的回报，并且保证了最终所有权重之和为 1 。



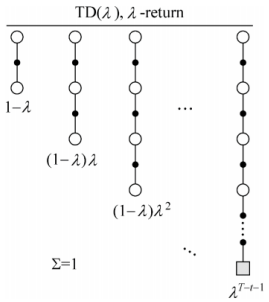


图 8.7: 2-step TD 和 4-step TD 算法结合　　TD(λ) 的向前视图

λ -回报:

$$G_t^\lambda=(1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_t^n$$

如果轨迹长度为 T:

$$G_i^\lambda=(1-\lambda)\sum_{n=1}^{T-t-1}\lambda^{n-1}G_t^n+\lambda^{T-t-1}G_t$$

值函数更新公式如下:

$$V(s_t)\leftarrow V(s_t)+\alpha\left(G_t^\lambda-V(s_t)\right)$$

如下前向视角示意图可知, 对于每个访问到的状态 s_t, 从它开始向前看所有的未来状态 s_{t+1}, s_{t+2}, ..., s_T, 并决定如何结合未来状态的回报来更新当前状态 s_t 的值函数 V (s_t) 。每更新完当前状态 s_t, 就转移至下一个状态 s_{t+1} ，不再回头关心已更新的状态 st 前向观点通过观看未来状态的回报估计当前状态的值函数。缺点: 必须要走完整个轨迹，获得每一个状态的即时奖励，才去更新，更新较低效。

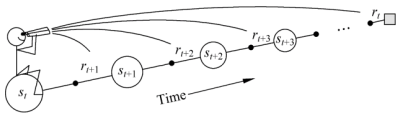


图 8.8: 向前算法示意图

8.2.4 后向算法

利用 TD(λ) 的前向观点估计值函数时，每一个时间步都需要用到很多步之后的信息，这在工程上很不高效。可以说，前向视角只提供了一个非常好但却无法直接实现的思路，这跟蒙特卡罗方法类似。实际中，我们需要一种无须等到实验结束就可以更新当前状态的值函数的更新方法。

这种增量式的更新方法需要利用多步时序差分的后向观点。它采用一种带有明确因果性的递增机制来实现值函数更新，恰恰解决了更新低效的问题。

后向视角在实现过程中，引入了一个和每个状态都相关的额外变量——资格迹。现在以一个“小狗死亡”的例子来解释资格迹的概念。如图下图所示，假设存在这样一个场景：一只小狗在连续接受了 3 次拳击和 1 次电击后死亡，那么在分析小狗的死亡原因时，到底是拳击的因素较重要还是电击的因素较重要呢？用资格迹表述就是哪个因素最有资格导致小狗死亡。

图 8.9: 小狗死亡示例

实际中进行资格分配时，有两种方式：一种是频率启发式，将资格分配给最频繁的状态，如上述例子里的拳击。另一种是最近启发式：将资格分配给最近的状态，如电击。而本节所介绍的资格迹同时结合了上述两种启发式。

在 t 时刻的状态 s 对应的资格迹, 标记为 $E_t(s)$:

$$
\begin{aligned}
E_0(s) &= 0 \\
E_t(s) &= \gamma\lambda E_{t-1}(s) + I\left(S_t = s\right)
\end{aligned}
$$

初始时刻, 每条轨迹中所有状态均有一个初始资格迹,$E_0(s) = 0$ 。下一时刻，被访问到的状态, 其资格迹为前一时刻该状态资格迹 $E_{t-1}(s)$ 乘以迹退化参数 λ 和衰减因子 γ, 然后加 1, 表示当前时刻该状态的资格迹变大。其他未被访问的状态, 其资格迹都只是在原有基础上乘以 λ 和 γ, 不用加 1, 表明它们的资格迹退化了。这种更新方式的资格迹为“累计型资格迹”。它在状态被访问的时候累计, 不被访问的时候退化, 如图所示。

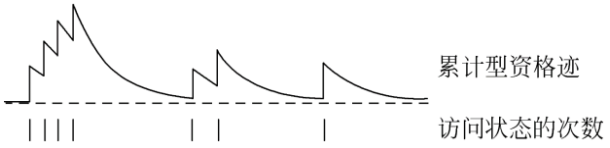


图 8.10: 累计型资格迹

以“小狗死亡”的例子来对资格迹定义进行说明。假设此例子一共涉及两个状态：拳击和电击，分别用 s1、s2 表示。λ=0.9，γ=0.8。初始时，令所有状态的资格迹为 0：

$$
E_0\left(s_1\right) = E_0\left(s_2\right) = 0
$$

当 $t=1$ 时, 有:

$$E_1(s_1)=\lambda\gamma E_0(s_1)+1=1$$
$$E_1(s_2)=\lambda\gamma E_0(s_2)=0$$

当 $t=2$ 时, 有:

$$E_2(s_1)=\lambda\gamma E_1(s_1)+1=1.72$$
$$E_2(s_2)=\lambda\gamma E_1(s_2)=0$$

当 $t=3$ 时, 有:

$$E_3(s_1)=\lambda\gamma E_2(s_1)+1=2.24$$
$$E_3(s_2)=\lambda\gamma E_2(s_2)=0$$

当 $t=4$ 时, 有:

$$E_4(s_1)=\lambda\gamma E_3(s_1)=1.61$$
$$E_4(s_2)=\lambda\gamma E_3(s_2)+1=1$$

因此在推测小狗的致死原因时，拳击所占的比重更大一些。同时，从计算拳击和电击两个状态的资格迹的过程可见，资格迹定义同时结合了频率启发式和最近启发式。其中，$\lambda\gamma E_{t-1}(s)$ 代表频率启发式，指示函数 $(s_t=s)$ 代表最近启发式。

可用资格迹 $\mathrm{E_t(s)}$ 来分配各值函数更新的资格。可以使用资格迹来衡量当 TD 误差发生时，各状态的值函数更新会受到多大程度的影响。

TD 误差公式如下：

$$\delta_t=R_{t+1}+\gamma V_t(S_{t+1})-V_t(S_t)$$

结合资格迹更新状态价值，则有：

$$V(s)\leftarrow V(s)+\alpha\delta_t E_t(s)$$

后向算法示意图，每次当前状态获得一个误差量 t 时，这个误差量都会根据之前各状态的资格迹来分配误差，进行值函数更新。此时之前各状态值函数更新的大小与距离当前访问状态的时间步相关。假设当前状态为 s_t，TD 偏差为 δ_t，那么 s_{t-1} 处的值函数更新资格乘以 $\lambda\gamma$，状态 s_{t-2} 处的值函数更新乘以 $(\lambda\gamma)^2$，以此类推。

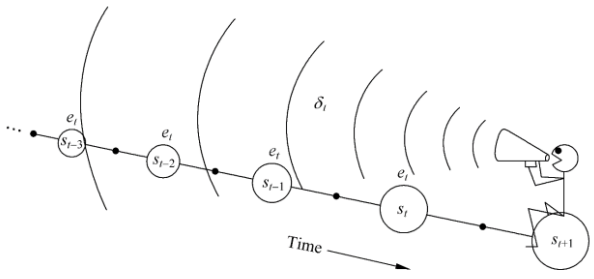


图 8.11: 后向算法示意图

TD(λ) 后向算法流程如下：

算法:TD(λ) 后向算法

输入: 环境 E, 状态空间 S, 动作空间 A, 折扣回报 γ, 初始化值函数 $V(s)=0$

For $k=0,1,\cdots,m$ do (针对每一条轨迹)

初始化资格迹 $Z(s)=0$, 对于所有的 $s\in S$

初始化状态 s

For $t=0,1,2,3\cdots$ do (针对轨迹中的每一步)

针对 s, 在 E 中通过 ϵ 贪心策略采取行为 a

$r,s'=$ 在 E 中执行动作 a 产生的回报和转移的状态;

$\delta\leftarrow R+\gamma V\left(s'\right)-V(s)$

$Z(s)\leftarrow Z(s)+1$

对于所有的 $s\in S$

$V(s)\leftarrow V(s)\leftarrow\alpha\delta Z(s)$

$Z(s)\leftarrow\gamma\lambda Z(s)$

$s^{\leftarrow}s'$

end for (s 为终止状态时，轨迹结束)

end for

$\pi^*(s)=\underset{a\in A}{\operatorname{argmax}}R_s^a+\gamma\sum_{s'\in S}P_{ss'}^aV\left(s'\right)$

8.2.5 Sarsa(λ) 方法

将多步时间差分方法和 Sarsa 算法结合，得到一种新的方法—Sarsa(λ) 。p 区别：不再去学习 $V_t(s)$，而是去学习 $Q_t(s\ a)$。同样的，Sarsa(λ) 也分为前向 Sarsa(λ) 方法和后向 Sarsa(λ) 方法。

前向 Sarsa(λ) 方法

下图为 Sarsa(λ) 的前向视图，其主要思想同 TD(λ) 一样, 是通过给每个回报哈子权值 $(1-\lambda)\lambda^{n-1}$ 来平均多个不同的 Q -回报进行更新。

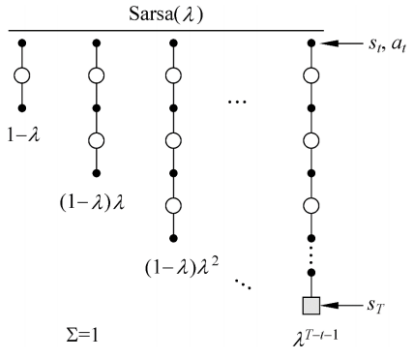


图 8.12: Sarsa(λ) 前向视图

这里的 Q -回报，相对于買积回报 G, 指的是状态行为对 (s,a) 从 t 时刻开始往后所有的回报的有衰减的总和。

其中,n-step Sarsa 的 Q-回报为:

$$Q_t^{\pi} = r_{i+1} + \gamma r_{t+2} + \gamma^{n-1} r_{t+n} + \gamma'' Q\left(s_{t+\infty}, a_{t+n}\right)$$

对 Q-回报加权求和得到 Q 的 λ -回报 Q_i^{λ} :

$$Q_t^{\lambda} = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} Q_i^{(n)}$$

结合 Sarsa 的更新公式, 得到 Sarsa(λ) 的更新公式:

$$Q\left(S_t, A_t\right) \leftarrow Q\left(S_t, A_t\right) + \alpha(G_t^{\lambda} - Q(s_t, A_t))$$

同普通 TD(λ) 算法一样，前向 Sarsa(λ) 估计值函数时，需要用到很多步以后的 Q-回报，这在工程应用中很不高效，因此实际中用得比较多的还是增量更新的后向算法。

后向 Sarsa(λ) 方法

后向 Sarsa(λ) 算法通过引人资格迹, 将当前行为值函数误差按比例抛给其他状态行为值函数, 作为其更新的依据。不同的是, 资格迹不再是 $E_1(s)$, 而是 $E_e(s,a)$, 即针对每一个状态行为对都有一个资格迹，公式如下:

$$\begin{aligned} E_0(s,a) &= 0 \\ E_r(s,a) &= \gamma\lambda E_{t-1}(s,a) + I\left(S_2 = s, A_i = a\right) \end{aligned}$$

其他的部分和后向 TD(λ) 一模一样。$Q_t(s,a)$ 的更新公式如下:

$$Q_{t+1}(s,a) < Q_t(s,a) + \alpha\delta_t E_t(s,a)$$

其中，$\delta_t = R_{t+1} + \gamma Q\left(s_{t+1}, a_{r+1}\right) - Q\left(s_t, a_t\right)$

$Sarsa(\lambda)$ 是在线策略算法, 也就是采样的策略和评估改进的策略是同一个策路。对于在线策略算法, 策略的更新方式有很多, 最简单的就是依据当前的行为值函数估计值采用 ϵ? 贪心算法进行更新。

$Sarsa(\lambda)$ 后向算法为单个轨迹内, 每进行一个时间步, 都会基于这个时间步的数据对行为值函数进行更新, 产生采样的策略和评估改进的策略都是 ϵ-贪心策略。

算法流程如下。

算法：Sarsa(λ) 后向算法
输人: 环境 E, 状态空间 S, 动作空间 A, 折扣回报 γ, 初始化行为值函数 $Q(s,a)=0$, $\pi(a \mid s) = \frac{1}{\lvert A\rvert}$
For $k = 0, 1, \cdots, m$ do (针对每一条轨迹) 初始化所有状态行为对的资格迹: $E(s,a) = 0$ 初始化状态 s 和行为 a, 得到第一个状态行为对 (s,a)

For $t = 0, 1, 2, 3 \cdots$ do(针对轨迹中的每一步)
$R, s' =$ 在 E 中执行动作 a 产生的回报和转移的状态;
基于 s', 通过 π 的 ε -贪心策略采取行为 a', 得到第二个状态行为对 (s', a')
求解 TD 误差 : $\delta \leftarrow R + \gamma Q(s', a') - Q(s, a)$
更新当前访问的状态行为对 (s, a) 的资格迹: $E(s, a) \leftarrow E(s, a) + 1$
对于所有的状态行为对 (s, a)
$Q(s, a) \leftarrow Q(s, a) + \alpha\delta E(s, a)$
$E(s, a) \leftarrow \gamma\lambda E(s, a)$
$s \leftarrow s', a \leftarrow a'$
end for s 是一个终止状态
$\forall s_t \in S'$:

$$\pi(s) == \text{argmax}_{a \in A} Q(s, a)$$

end for

输出: 最优策略 π

8.2.6 Q(λ) 方法

当我们把资格迹和 Sarsa 方法结合后，自然会想到是否资格迹也能和 Q-learning 方法结合。答案是可以，得到的方法就是 Q(λ) 方法。

前向 Watkins's Q(λ) 方法

常规的 Q-learning 方法属于离线策略算法，即产生采样的策略 (行为策略) 和评估改进的策略 (目标策略) 不是同一个策略。也就是说：Q-learning 方法需要依据带有 ϵ-贪心行为的轨迹来学习一个贪心策略。

在进行行为值函数估计的时候，Q-learning 采取的更新公式如下：

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left(R + \gamma \max Q\left(s', a'\right) - Q(s, a)\right)$$

对应的更新目标为：

$$Q_t = R + \gamma \max_{a'} Q\left(s', a'\right)$$

假设我们正在求解贪心策略在状态行为对 (s_t, a_t) 的行为值函数，前两个时间步选择的行为是贪婪行为，但是第三个时间步选择的行为是探索行为，那么 Watkins's Q(λ) 使用的有效轨迹长度，最长就到第二个时间步，从第三个时间步往后的序列都不再理会。也就是说，Watkins's Q(λ) 使用的有效轨迹长度最远到达第一个探索行为对应的时间步长。因为 Q(λ) 要学习的是贪心策略，而第三步采用的是探索行为，因此当 n 3 的 n 步回报已经和贪心策略没有联系了。

如何确定当前时间步选择的行为是不是贪婪行为呢? 很简单，把当前选择的行为和当前的 $argmax_{a'} Q(s'\ a')$ 对比，如果一致就是贪婪行为，不一致就是探索行为。

因此，不同于 TD(A) 或者 Sarsa(λ)，Watkins's Q(λ) 所使用的有效轨迹长度不是整个轨迹从开始到结束，它只考虑最近的探索行为，一旦探索行为发生，则轨迹结束。

Q 若 a_{t+n} 是第一个探索行为, 则轨迹以 s_{t+n} 为最后一个状态，则最长的 n 步 Q-回报为:

$$Q_t^n = r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n \max Q\left(s_{t+n}, a\right)$$

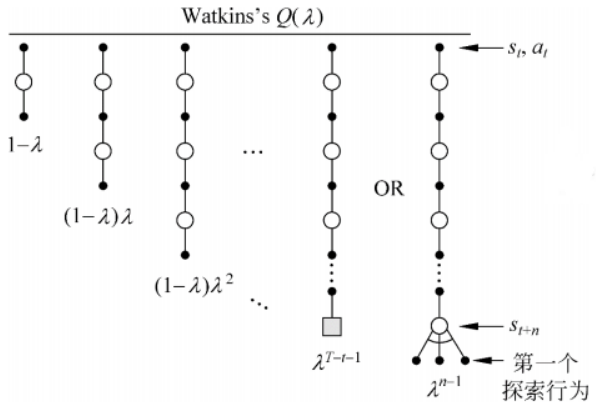


图 8.13: Watkins's Q(λ) 前向视图

可见，Watkins's Q(λ) 方法所使用的有效轨迹长度取决于第一个探索行为。如果第一个探索行为在轨迹结束前出现，有效轨迹长度最远到达此探索行为对应的时间步，否则等于整个轨迹长度。

对 Q-回报加权求和得到 Q 的 λ -回报 Q_t:

$$Q_t^\lambda = (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} Q_t^{(n)}$$

Watkins's Q(λ) 更新公式为:

$$Q\left(s_t, a_t\right) \leftarrow Q\left(s_t, a_t\right) + \alpha \left(Q_t^\lambda - Q\left(s_t, a_t\right)\right)$$

后向 Watkins's Q(λ) 方法

后向 Watkins's Q(λ) 算法的算法流程如下:

后向 Watkins's $Q(\lambda)$ 算法

输入: 环境 E, 状态空间 S, 动作空间 A, 折扣回报 γ, 初始化行为值函数 $Q(s,a)=0$, $\pi(a \mid s)=\frac{1}{|A|}$

For $k=0,1,\cdots,m$ do (针对每一条轨迹)

初始化所有状态行为对的资格迹: $E(s,a)=0$

初始化状态 s 和行为 a, 得到第一个状态行为对 (s,a)

For $t=0,1,2,3\cdots$do(针对轨迹中的每一步)

$R,s'=$ 在 E 中执行动作 a 产生的回报和转移的状态;

基于 s', 通过 π 的 ε -贪心策略采取行为 a', 得到第二个状态行为对 (s',a')

$a^* \leftarrow \arg\max_{b\in A} Q(s,b)$

$\delta^\star - R+\gamma Q\left(s',a^*\right)-Q(s,a)$

$E(s,a)\leftarrow E(s,a)+1$

对于所有的状态行为对 (s,a)

$Q(s,a)\leftarrow Q(s,a)+\alpha\delta E(s,a)$

如果 $a'=a^*$, 则有 $E(s,a)\leftarrow\gamma\lambda E(s,a)$

否则 $E(s,a)\leftarrow 0$

$s^\star s', a\leftarrow a'$

end for s 是一个终止状态

end for

$\forall s_t\in S': \pi^*(s)=\arg\max_{a\in A} Q(s,a)$

输出: 最优策略 $\pi*$

后向 Watkins's Q(λ) 的资格迹如何更新呢?

对于所有状态行为对 (s_t, a_t) 的资格迹, 更新分两部分: 首先, 如果当前选择行为 a_t 是贪妥行为, 资格迹乘以系数 $\gamma\lambda$, 否则资格迹变成 0 ; 其次, 对于当前正在访问的 (s_t, a_t), 其资格迹单独加 1。

资格迹的更新公式如下:

$$E_t(s,a)=I_{ss_t}\cdot I_{aa_t}+\begin{cases}\gamma\lambda E_{t-1}(s,a) & \text{if } Q_{t-1}\left(s_t,a_t\right)=\max_a Q_{t-1}\left(s_t,a\right)\\ 0 & \text{else}\end{cases} \tag{8.5}$$

其中, I_{ss_t} 和 I_{aa_t} 均为指示函数。I_{ss_t} 表示当 $s=s_t$ 时, 其值为 1。

$Q(s,a)$ 更新公式如下:

$$Q_{t+1}(s,a)=Q_t(s,a)+\alpha\delta_t E_t(s,a)$$

δt 计算公式如下:

$$\delta_t=r_{t+1}+\gamma\max Q_t\left(s_{t+1},a'\right)-Q_t\left(s_t,a_t\right)$$

8.3 值函数逼近

前面已经介绍了强化学习的基本方法: 基于动态规划的方法, 基于蒙特卡罗的方法和基于时序差分的方法。对于模型已知的系统, 可以利用动态规划的方法得到值函数; 对于模型未知的系

统，可以利用蒙特卡罗的方法或时序差分的方法得到值函数。应注意到使用这些方法的一个基本前提条件：状态空间和动作空间是离散的，而且状态空间和动作空间不能太大。具体来说，这里的值函数其实是一个表格。值函数的迭代更新实际上就是这张表的迭代更新。因此，之前所给的强化学习算法又称为表格型强化学习。对于状态值函数，其表格的维数为状态的个数 $|\mathcal{S}|$ ，其中 $\mathcal{S}$ 为状态空间。若状态空间的维数很大，或者状态空间为连续空间，此时值函数无法用一张表格来表示。这时，我们需要利用函数逼近的方法表示值函数。我们首先介绍了值函数逼近的思想。

在引入近似值函数后，强化学习中不管是预测问题还是控制问题，就转变成近似函数的设计以及求解近似函数这两个问题了。函数逼近方法可以分为参数逼近和非参数逼近。参数近似方法又可分为基于参数的线性函数近似和非线性近似两类，此时值函数可以由一组参数来近似表示，则值函数的更新也就等价于参数的更新。本节主要介绍参数近似方法，其中，通过建立目标函数，参数可使用梯度下降的办法进行训练求解。本节也就参数更新方法及常用的参数近似方法详细展开。

8.3.1 值函数逼近的思想

前面提到的基本方法中价值函数的更新是基于表格的迭代更新 (tabular solution)，这意味着每一个状态对应一个 $V(s)$ 或者每一个状态-行为对对应一个 $Q(s,a)$ 。而在现实中，状态空间一般都较大，就会产生“维数灾难”的问题。尤其是当状态空间是连续的时候，会有无穷的状态空间。比如，西洋双陆棋（Backgammon）所需要的状态空间是 10^{20}，围棋 AlphaGo 有 10^{170} 个状态空间，机器人控制以及无人机控制需要的是一个连续状态空间。对于类似的大规模问题，按查表的方式更新值函数需要太多的内存来存储，并且对于连续的状态空间无法用查表的方式更新值函数，而且有时候针对每一个状态学习得到值函数也是一个很慢的过程。此时，我们无法用之前基于表格的方法为每个状态或者状态动作对都求得一个值函数的特定值，因此需要有一种方法能够适应无限的状态集。值函数逼近可以将强化学习应用到这类大规模的问题中，进而进行预测和控制。

值函数逼近即建立一个函数 $\hat{V}$，这个函数由参数 w 描述，它直接接受表示状态特征的连续变量 s 作为输入，通过计算得到一个状态值函数，通过调整参数 w 的取值，使其符合基于某一策略 π 的最终状态值，那么这个函数就是状态值函数 $V^{\pi}(s)$ 的近似表示，

$$\hat{V}(s;w) \approx V^{\pi}(s).$$

类似的，如果由参数 w 构成的函数 $\hat{Q}$ 同时接受状态变量 s 和行为变量 a，然后输出一个行为值函数，通过调整参数 w 的取值，使其符合基于某一策略 π 的最终行为值函数，那么这个函数就是行为值函数 $Q^{\pi}(s,a)$ 的近似表示，

$$\hat{Q}(s,a;w) \approx Q^{\pi}(s,a).$$

此外，如果由参数构成的函数仅接受状态变量 s 作为输入，输出针对行为空间中的每一个离散行为值 $Q(s,a_j;w), j=1,\cdots,|\mathcal{A}|$，这给出另一种行为值函数的近似表示。在上面公式中，描述状态的 s 不再是一个字符串或者一个索引，而是由一系列的数据组成的向量，构成向量的每一项称为状态的一个特征，该项的数据值称为特征值；参数 w 需要通过求解确定，w 通常是一个向

量或矩阵等。构建了值函数的逼近形式后，强化学习中的预测和控制问题就转变为求解近似值中的参数 w 了。w 可以通过建立目标函数，然后使用梯度下降的方式进行求解。

注意到值函数逼近方法除节省存储空间之外，还能把从已知状态学到的函数通用化推广至那些未碰到的状态中 — 泛化性，即，假设离某个状态非常接近的另一状态，比如两状态之间的物理位置只相隔一毫米，则不必要单独存储其值函数。

概括来说，表格型强化学习和函数逼近方法的强化学习，值函数更新时的异同点可归结如下。(1) 表格型强化学习在更新值函数时，只有当前状态 s_t 处的值函数改变，其他地方的值函数不改变。(2) 值函数逼近方法更新值函数时，更新的是参数 w，而估计的值函数为 $\hat{V}(s;w)$，所以当参数 w 发生改变，任意状态处的值函数都会发生改变。

8.3.2 目标函数及梯度下降

我们现在对值函数逼近思想已有了解，即把用表格显式精确存储的值函数用近似函数来近似表示。也就是把原来的问题转移化为怎么使得近似变得更加准确的问题。接下来跟普遍机器学习的路数一样，我们要定义一个目标函数，并对这个目标函数进行优化迭代学习直到收敛，求得最优解。但需要注意强化学习里只有即时奖励，没有监督数据。所以我们要找到能替代 $V^{\pi}(s)$ 的目标值，以便使用监督学习的算法学习到近似函数的参数。

首先回顾一下表格型强化学习值函数更新的公式，以更好理解值函数逼近下的更新公式。

蒙特卡罗方法，值函数更新公式为

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left(G_t - Q(s,a)\right).$$

TD(0) 方法值函数更新公式为

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[R + \gamma Q\left(s',a'\right) - Q(s,a)\right].$$

TD(λ) 方法值函数更新公式为

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[G_t^{\lambda} - Q(s,a)\right].$$

从上式值函数的更新过程可以看出，无论是蒙特卡罗方法还是时序差分方法，值函数更新过程都是向着目标值函数靠近，这个目标值在蒙特卡罗方法中是 G_t，在时序差分方法中是 $r+\gamma Q\left(s',a'\right)$，在 TD($\lambda$) 中是 G_t^{λ}。

现把上式中的值函数换成近似函数，则相应的更新公式就变成了基于近似值函数的值更新方法。比如，TD(0) 方法值函数逼近下的更新公式为

$$\hat{Q}(s,a;w) \leftarrow \hat{Q}(s,a;w) + \alpha \left[r + \gamma \hat{Q}\left(s',a';w\right) - \hat{Q}(s,a;w)\right]$$

假设我们找到了参数 w 使得逼近值函数最终收敛不再更新，则意味着对任何状态或状态行为对，有形式：

$$\hat{Q}(s,a;w) = r + \gamma \hat{Q}\left(s',a';w\right).$$

事实上，很难找到参数 w 使得上式完全成立。同时由于算法是基于采样数据的，即使上式对于采样得到的状态成立，也很难对所有可能的状态成立。为衡量在采样产生的 T 个状态转换上

近似值函数的收敛情况, 定义目标函数 J_w 为:

$$J(w)=\frac{1}{2T}\sum_{t=1}^{T}\left[\left(R_t+\gamma\hat{Q}\left(s_t',a_t';w\right)\right)-\hat{Q}\left(s_t,a_t;w\right)\right]^2 \tag{8.6}$$

公式 (8.6) 中 T 为采样得到的状态转换的总数。若近似值函数 $\hat{Q}(s,a;w)$ 收敛，则意味着 $J(w)$ 逐渐减小。如果 $T=1$, 则通常称为损失 (loss)。定义损失 $\text{loss}(w)$ 为:

$$\text{loss}(w)=\frac{1}{2}\left[\left(R+\gamma\hat{Q}\left(s',a';w\right)\right)-\hat{Q}\left(s,a;w\right)\right]^2. \tag{8.7}$$

对于蒙特卡罗方法, 使用 G_t 代替目标值，目标函数为:

$$J(w)=\tfrac{1}{2T}\sum_{t=1}^{T}\left[G_t-\hat{V}\left(s_t;w\right)\right]^2,$$

$$J(w)=\tfrac{1}{2T}\sum_{t=1}^{T}\left[G_t-\hat{Q}\left(s_t,a_t;w\right)\right]^2.$$

如果事先存在对于预测问题最终基于某一策略最终价值函数 $V^{\pi}(s)$ 或 $Q^{\pi}(s,a)$, 或者存在对于控制问题的最优价值函数 $V^{*}(s)$ 或 $Q^{*}(s,a)$, 那么可以使用这些价值来代替上式中的目标值, 这里集中使用 V_{target} 或 Q_{target} 来代表目标值, 使用期望代替平均值的方式, 那么目标函数的表达公式为:

$$J(w)=\frac{1}{2}\mathbb{E}_{\pi}\left[V_{\text{target}}(s)-\hat{V}(s;w)\right]^2, \tag{8.8}$$

$$J(w)=\frac{1}{2}\mathbb{E}_{\pi}\left[Q_{\text{target}}(s,a)-\hat{Q}(s,a;w)\right]^2. \tag{8.9}$$

对于大规模强化学习问题来说，并不能保证对所有的 $J(w)$ 直接对其求梯度，并基于所求梯度等于 0 来得到最优参数。在这种情况下可以通过迭代、使用梯度下降法来求解。具体过程如下:

1. 设置初始参数值 $w=(w_1,w_2,\ldots,w_d)$;
2. 获取一个状态转换, 代入目标函数 $J(w)$, 并计算 $J(w)$ 对各参数 w 的梯度:

$$\nabla_w J(w)=\begin{pmatrix}\frac{\partial J(w)}{\partial w_1}\\ \vdots\\ \frac{\partial J(w)}{\partial w_d}\end{pmatrix}$$

3. 设置一个正的较小的步长 α, 将原参数 w 朝着梯度的反方向更新:

$$\Delta w=-\alpha\nabla_w J(w)$$
$$w\leftarrow w+\Delta w$$

对于式 (8.8), w 更新的具体表达式推导如下:

$$\begin{aligned}w_{t+1}&=w_t-\alpha_t\nabla J\left(w_t\right)\quad\text{(梯度下降准则)}\\&=w_t+\alpha_t\mathbb{E}_{\pi}\left[V_{\text{target}}(s_{t+1})-\hat{V}(s_t;w_t)\right]\nabla_w\hat{V}(s_t;w)|_{w=w_t}\\&\approx w_t+\alpha_t\left[V_{\text{target}}(s_{t+1})-\hat{V}(s_t;w_t)\right]\nabla_w\hat{V}(s_t;w)|_{w=w_t}\quad\text{(抽样)}\end{aligned} \tag{8.10}$$

使用不同的学习方法时，V_{target} 由不同的目标值代替，比如，对于蒙特卡罗方法，使用 G_t 代替 V_{target}，如表 8.1 所示。在时序差分方法中，用 $R_{t+1}+\hat{V}(s_{t+1};w)$ 代替 V_{target}。对于式 (8.9)，参数 w 的更新表达式类似，同样用不同的目标值代替 Q_{target}。

8.1 基于梯度的蒙特卡罗值函数评估算法

输入: 要评估的策略 π，一个可微的逼近函数 $\hat{V}:\mathcal{S}\times\mathbb{R}^d\to\mathbb{R}$
　　恰当地初始化值函数的权重 w （比如，$w=0$）
Repeat:
　　利用策略 π 产生一幕数据
　　For $t=0,1,\cdots,T-1$
　　$w\leftarrow w+\alpha_t[G_t-\hat{V}(s_t;w)]\nabla_w\hat{V}(s_t;w)$

概括来说，式 (8.8), (8.9) 是我们要求解的。在实际求解近似值函数中的参数 w 时, 式中的期望 $\mathbb{E}_\pi$ 并不好处理，为了去掉期望 $\mathbb{E}_\pi$, 最好的方法就是将 π 的概率分布求出来, 然后再加权求和。这样操作过于复杂, 实际操作的时候会采用蒙特卡罗的形式, 只用一个样本或者一批样本进行更新。即，我们基于近似值函数的目标值来代替 V_{target} 或 Q_{target}，考虑抽样后的参数更新表达式来得到极小值 w。

注意到在时序差分方法中，要更新的参数 w 不仅出现在要估计的值函数 $\hat{V}(s_t;w)$ 中，还出现在目标值 V_{target} 中。这里只考虑参数 w 对估计值函数 $\hat{V}(s_t;w)$ 的影响而忽略对目标值函数 V_{target} 的影响, 这种方法称为基于半梯度的 TD(0) 值函数评估算法，如表 8.2 所示。表 8.3 给出了基于半梯度的 Sarsa 算法。

8.2 基于半梯度的 TD(0) 值函数评估算法

输入: 要评估的策略 π，一个可微的逼近函数 $\hat{V}:\mathcal{S}\times\mathbb{R}^d\to\mathbb{R}$
　　恰当地初始化值函数的权重 w （比如，$w=0$）
Repeat:
　　初始化状态 s
　　Repeat （对一幕中的每一步）
　　选择动作 $a\sim\pi(\cdot|s)$
　　采用动作 a 观测到回报 R,s'
　　$w\leftarrow w+\alpha_t[R+\gamma\hat{V}(s';w)-\hat{V}(s_t;w)]\nabla_w\hat{V}(s_t;w)$
　　$s\leftarrow s'$
　　直到 s' 是终止状态

8.3 基于半梯度的 Sarsa 算法

输入: 一个可微的逼近函数 $\hat{Q}:\mathcal{S}\times\mathcal{A}\times\mathbb{R}^d\to\mathbb{R}$
　　任意初始化值函数的权重 w （比如，$w=0$）
Repeat (for each episode):
　　初始化状态行为对 s,a
　　Repeat （对一幕数据中的每一步）

采用动作 a 观测到回报 R, s'
如果 s' 是终止状态: $w \leftarrow w + \alpha_t[R - \hat{Q}(s,a;w)]\nabla_w \hat{Q}(s,a;w)$
　　进入下一幕
利用软策略选择一个动作 a'，以便估计动作值函数 $\hat{Q}(s',a';w)$
$w \leftarrow w + \alpha_t[R + \gamma\hat{Q}(s',a';w) - \hat{Q}(s,a;w)]\nabla_w \hat{Q}(s,a;w)$
$s \leftarrow s'$
$a \leftarrow a'$

8.3.3 线性逼近

到目前为止，我们已经知道了值函数逼近下的优化目标函数，优化策略一般用梯度下降来优化，接下来我们讨论要逼近的值函数的形式。理论上任何函数都可以被用作近似值函数，实际选择何种近似函数需根据问题的特点。比较常用的近似函数有线性模型、神经网络、决策树、最近邻法、傅里叶变换、小波变换等。总的来说就是基于一映射 $\phi : \mathcal{S} \to \mathbb{R}^d$ 构造特征集（基函数），希望每一个状态都能被恰当的表示。本小节主要介绍参数化的线性逼近和基于神经网络的非线性逼近。首先我们引入线性逼近。

相比于非线性逼近, 线性逼近的好处是只有一个最优值, 因此可以收敛到全局最优。线性逼近的表达形式为:

$$\hat{V}(s;w) = w^T\phi(s) = \sum_{j=1}^{d} w_j\phi_j(s)$$

其中 $\phi(s)$ 为状态 s 处的特征函数，或者称为基函数，w 为要求解的参数。常用的基函数的类型如下。

- 多项式基函数，如 $(1, s_1, s_2, s_1s_2, s_1^2, s_2^2, \cdots)$
- 傅里叶基函数: $\phi_j(s) = \cos(j\pi s), s \in [0,1]$
- 径向基函数: $\phi_j(s) = \exp\left(-\frac{\|s-c_j\|^2}{2\sigma_j^2}\right)$

对于由线性函数 $\hat{V}(s;w)$ 近似的值函数, 其相应的目标函数 $J(w)$ 为:

$$J(w) = \frac{1}{2}\mathbb{E}_\pi\left[V_{\text{target}}(s) - w^T\phi(s)\right]^2$$

相应的梯度 $\nabla_w J(w)$ 为:

$$\nabla_w J(w) = -\mathbb{E}_\pi\left(V_{\text{target}}(s) - w^T\phi(s)\right)\phi(s)$$

基于样本，参数的更新表达式为:

$$w_{t+1} \leftarrow w_t + \alpha_t\left(V_{\text{target}}(s_{t+1}) - w_t^T\phi(s_t)\right)\phi(s_t)$$

正如前文提到的，使用不同的学习方法时, $V_{\text{target}}(s_{t+1})$ 由不同的目标值代替。

- 蒙特卡罗方法值函数线性逼近参数更新公式:

$$w_{t+1} \leftarrow w_t + \alpha_t\left[G_t - w^T\phi(s_t)\right]\phi(s_t)$$

- TD(0) 线性逼近参数更新公式：

$$w_{t+1} \leftarrow w_t + \alpha_t \left[R_{t+1} + \gamma w^T \phi(s_{t+1}) - w^T \phi(s_t)\right] \phi(s_t)$$

- 前向视角的 TD(λ) 参数更新公式：

$$w_{t+1} \leftarrow w_t + \alpha_t \left(G_t^\lambda - w^T \phi(s_t)\right) \phi(s_t)$$

- 后向视角的 TD(λ) 参数更新公式：

$$\begin{aligned}
\delta_t &= R_{t+1} + \gamma w^T \phi(s_{t+1}) - w^T \phi(s_t) \\
E_t &= \gamma \lambda E_{t-1} + \phi(s_t) \\
w_{t+1} &\leftarrow w_t + \alpha_t \delta_t E_t
\end{aligned}$$

如式 (8.6) 所示，为衡量在采样产生的 T 个状态转换上近似值函数的收敛情况，希望充分利用样本数据 $D = \{\langle s_1, V_1^\pi \rangle, \langle s_2, V_2^\pi \rangle, \cdots, \langle s_T, V_T^\pi \rangle\}$，找到最好的拟合函数 $\hat{V}(s, w)$，使得目标函数，记为 $LS(w)$，最小，

$$LS(w) = \frac{1}{2T} \sum_{t=1}^{T} \left(V_t^\pi - w^T \phi(s_t)\right)^2.$$

基于 $LS(w)$ 可以直接找到参数使得 $LS(w)$ 为最小值，即线性最小二乘逼近，并且满足经验集中的各个状态，

$$\sum_{t=1}^{T} \left[V_t^\pi - w^T \phi\left(s_t\right)\right] \phi\left(s_t\right) = 0.$$

- 最小二乘蒙特卡罗方法参数为

$$w = \left(\sum_{t=1}^{T} \phi\left(s_t\right) \phi\left(s_t\right)^T\right)^{-1} \sum_{t=1}^{T} \phi\left(s_t\right) G_t$$

- 最小二乘 TD(0) 方法为

$$w = \left(\sum_{t=1}^{T} \phi\left(s_t\right) \left(\phi\left(s_t\right) - \gamma \phi\left(\mathrm{s}_{t+1}\right)\right)^T\right)^{-1} \sum_{t=1}^{T} \phi\left(s_t\right) R_{t+1}$$

- 最小二乘 TD(λ) 前向方法为

$$w = \left(\sum_{t=1}^{T} \phi\left(s_t\right) \phi\left(s_t\right)^T\right)^{-1} \sum_{t=1}^{T} \phi\left(s_t\right) G_t^\lambda$$

- 最小二乘 TD(λ) 后向方法为

$$w = \left(\sum_{t=1}^{T} E_t \left(\phi\left(s_t\right) - \gamma \phi\left(\mathrm{s}_{t+1}\right)\right)^T\right)^{-1} \sum_{t=1}^{T} E_t R_{t+1}$$

8.3.4 非线性逼近

线性逼近时，首先要选基函数，再根据目标函数训练得到基函数所对应的参数。这种线性逼近的方法表示能力非常有限，因为数量太少的基函数无法很好的逼近复杂的函数，而且基函数的形式是事先选定的，这也限制了函数的逼近能力。这里我们介绍一些常用的基于卷积神经网络（深度学习）的 Q-learning 算法及其变体：Deep Q-learning （DQN），Double DQN 和 Dueling DQN。由于卷积神经网络可以看成是基函数参数化的一种方法，因此本小节给出的逼近方法也称为基于参数的非线形逼近。关于卷积神经网络的具体内容详见第三章。

DQN 算法

在之前的章节中我们学过，Q-learning 是一种离线学习法，它能学习当前经历着的，也能学习过去经历过的，甚至是学习别人的经历。而 DQN，就是在传统 Q-learning 的基础上做了三个改进，使得其效果显著提升。

DQN 对 Q-learning 的修改主要体现在以下三个方面：

1. DQN 利用深度卷积神经网络逼近值函数；

2. DQN 利用了经验回放训练强化学习的学习过程；

3. DQN 独立设置了目标网络来单独处理时序差分算法中的 TD 偏差.

下面就三个方面进行具体介绍。

（1）DQN 利用深度卷积神经网络逼近值函数

我们已经知道线性逼近值函数的方法，即值函数由一组基函数和一组与之对应的参数相乘得到，值函数是参数的线性函数。而 DQN 的行为值函数利用神经网络逼近，而且用的是强大的深度卷积神经网络，也就是 CNN，属于非线性逼近。虽然逼近方法不同，但都属于参数逼近。此处的值函数对应着一组参数，在神经网络里参数是每层网络的权重。此时值函数的更新对应参数的更新。

（2）DQN 利用了经验回放训练强化学习的学习过程

当时学者们发现利用神经网络，尤其是深度神经网络逼近值函数不太靠谱，因为常常出现不稳定不收敛的情况。主要原因是，训练神经网络时，存在的假设是训练数据是独立同分布的，但是通过强化学习采集的数据之间存在着关联性，利用这些数据进行顺序训练，神经网络自然不稳定。DeepMind 团队的研究人员构造了一种神经网络的训练方法：经验回放。具体来说，DQN 有一个记忆库用于学习之前的经历，每次 DQN 更新的时候，都可以随机抽取一些之前的经历进行学习，这种随机抽取做法打乱了经历之间的相关性，使得神经网络更新更有效率。

（3）DQN 独立设置了目标网络来单独处理时序差分算法中的 TD 偏差

与前面提到的表格型 Q-learning 算法不同的是，利用神经网络对值函数进行逼近时，值函数的更新步更新的是参数，其更新方法是梯度下降法。也就是说 Q-learning 值函数更新实际上变成了监督学习的一次更新过程：

$$w_{t+1} \leftarrow w_t + \alpha_t \left[R_{t+1} + \gamma \max_{a'} Q\left(s_{t+1}, a'; w_t\right) - Q(s_t, a_t; w_t) \right] \nabla_w Q(s_t, a_t; w)|_{w=w_t}$$

其中, $R_{t+1}+\gamma \max_{a'} Q\left(s_{t+1}, a_{t+1}; w_t\right)$ 为 TD 目标值，在计算 $\max_{a'} Q\left(s_{t+1}, a_{t+1}; w_t\right)$ 值时，用到的网络参数为 w_t。我们称计算 TD 目标时所用的网络为 TD 网络。在 DQN 算法出现之前, 利用神经网络逼近值函数时, 计算 TD 目标值时所用的网络参数, 与梯度计算中要逼近的值函数 $Q(s_t, a_t; w_t)$ 所用的网络参数相同, 这样就容易导致数据间存在关联性, 从而使训练不稳定。为了解决此问题, DeepMind 团队提出使用两个结构相同但参数不同的神经网络，其中动作值函数逼近的网络具备最新的参数，而用于计算 TD 目标值的网络则是每隔固定的步数更新一次，这样的方式能让训练变得更加有效，更具有鲁棒性。值函数的更新变为，

$$w_{t+1} \leftarrow w_t+\alpha_t\left[R_{t+1}+\gamma \max_{a'} Q\left(s_{t+1}, a'; w^{-}\right)-Q(s_t, a_t; w_t)\right] \nabla_w Q(s_t, a_t; w)|_{w=w_t}.$$

最后我们给出 DQN 的伪代码, 如下所示。

DQN 算法

初始化回放记忆 $\mathcal{D}$，可容纳的数据条数为 N
利用随机权值 w 初始化动作-行为值函数 Q
令 $w^{-}=w$ 初始化，计算 TD 目标值 Q

1. For episode= $1, \cdots, M$ do
2. 初始化事件的第一个状态 $s_1=\{x_1\}$，通过预处理得到状态对应的特征输入 $\phi_1=\phi(s_1)$
3. For $t=1, \cdots, T$ do
4. 利用概率 ε 选一个随机动作 a_t
5. 若小概率事件没发生，则用贪婪策略选择当前值函数最大的那个动作 $a_t=\arg\max Q\left(\phi\left(s_t\right), a; w\right)$
6. 在仿真器中执行动作 a_t，观测回报 R_t 以及图像 x_{t+1}
7. 设置 $s_{t+1}=s_t, a_t, x_{t+1}$, 预处理 $\phi_{t+1}=\phi\left(s_{t+1}\right)$
8. 将转换 $\left(\phi_t, a_t, R_t, \phi_{t+1}\right)$ 储存在回放记忆 $\mathcal{D}$ 中
9. 从回放记忆 $\mathcal{D}$ 中均匀随机采样一个转换样本数据，用 $\left(\phi_j, a_j, R_j, \phi_{j+1}\right)$ 表示
10. 令 $y_j=\begin{cases} R_j, & \text{事件为终止状态;} \\ R_j+\gamma \max_{a'} Q\left(\phi_{j+1}, a'; w^{-}\right), & \text{其它} \end{cases}$
11. 关于参数 w 执行一次梯度下降，目标函数为 $\left(y_j-Q(\phi_j, a_j; w)\right)^2$
12. 每隔 C 步更新一次 TD 目标网络权值，即令 $w^{-}=w$;
13. end for
14. end for

DQN 算法实践

这里我们首先引入常用的库

```
import random
import gym
import numpy as np
import collections
from tqdm import tqdm
import torch
```

```
import torch.nn.functional as F
import matplotlib.pyplot as plt
```

首先定义经验回放池的类，用于储存数据和随机抽取一些之前的经历进。

```
class ReplayBuffer:      #定义一个类
    def __init__(self, capacity):
        #初始化这个类的属性，self.buffer是这个类的属性，赋予他一个默认值，之后会
            用这些属性。
        self.buffer = collections.deque(maxlen=capacity)
        # 相当于 list ，但这里运算更快。数据是队列先进先出，maxlen是指队列最大
            长。

    def add(self, state, action, reward, next_state, done):
        # 定义类 ReplayBuffer的名为add的方法，这一方法是对这个类的属性进行操作。(
            ReplayBuffer.add(参数)可以调用这一方法)
        self.buffer.append((state, action, reward, next_state, done))
        #这里state, action,reward,next_state,done等参数在调用的时候是要我们自己输
            入。 append方法的目的是将数据加入buffer这一属性

    def sample(self, batch_size):
        #从buffer中采样数据,数量为 batch_size。
        transitions = random.sample(self.buffer, batch_size)
        #random.sample()随机从self.buffer里面抽取数量为batch_size的数据储存在
            transitions里。
        state, action, reward, next_state, done = zip(*transitions)
        return np.array(state), action, reward, np.array(next_state), done

    def size(self):
        #目前buffer中数据的数量
        return len(self.buffer)
```

接下来定义一个只有一层隐藏层的 Q 网络。之后我们会用到这样一个网络。

```
class Qnet(torch.nn.Module):
    #torch.nn.Module是一个父类，torch.nn.Module中的属性还有方法都传递给子类 Qnet
        .
    # 首先，导入torch.nn模块。实际上，“nn”是neural #networks（神经网络）的缩
        写。顾名思义，该模块定义了大量神经网络的层。
    def __init__(self, state_dim, hidden_dim, action_dim):
        super(Qnet, self).__init__()
        #子类继承了父类的所有属性和方法，父类属性自然会用父类方法来进行初始化
        self.fc1 = torch.nn.Linear(state_dim, hidden_dim)
        self.fc2 = torch.nn.Linear(hidden_dim, action_dim)

    def forward(self, x):
        x = F.relu(self.fc1(x))  # 隐藏层使用ReLU激活函数
        return self.fc2(x)
```

有了这些基本组件之后，接来下开始实现 DQN 算法。

```
class DQN:
    ''' DQN算法 '''
    def __init__(self, state_dim, hidden_dim, action_dim, learning_rate, gamma,
                 epsilon, target_update, device):
        self.action_dim = action_dim
        self.q_net = Qnet(state_dim, hidden_dim,self.action_dim).to(device)
        # Q网络
        # 目标网络
        self.target_q_net = Qnet(state_dim,hidden_dim,self.action_dim).to(device)
        # 使用Adam优化器
        self.optimizer = torch.optim.Adam(self.q_net.parameters(),
                                          lr=learning_rate)
        self.gamma = gamma  # 折扣因子
        self.epsilon = epsilon  # epsilon-贪婪策略
        self.target_update = target_update  # 目标网络更新频率
        self.count = 0  # 计数器,记录更新次数
        self.device = device

    def take_action(self, state):  # epsilon-贪婪策略采取动作
        if np.random.random() < self.epsilon:
            action = np.random.randint(self.action_dim)
        else:
            state = torch.tensor([state], dtype=torch.float).to(self.device)
            action = self.q_net(state).argmax().item()      #.argmax()返回最大值的
                索引值,.item()将张量转化为标量。
        return action

    def update(self, transition_dict):
        #transition_dict是一个字典，例如含有键对  'actions':3
        states = torch.tensor(transition_dict['states'], dtype=torch.float).to(
            self.device)
        #返回states对应数值。
        actions = torch.tensor(transition_dict['actions']).view(-1, 1).to(self.
            device)
        #.view(-1, 1)按照列展平数组，-1表示展平，1表示按照列。例：view(3, 4)重塑
            形状为三行四列的张量。
        rewards = torch.tensor(transition_dict['rewards'],dtype=torch.float).view
            (-1, 1).to(self.device)
        next_states = torch.tensor(transition_dict['next_states'], dtype=torch.
            float).to(self.device)
        dones = torch.tensor(transition_dict['dones'],
        dtype=torch.float).view(-1, 1).to(self.device)
```

```
        q_values = self.q_net(states).gather(1, actions)
        # 返回Q值 .gather表示聚合，1表示按照列聚合，actions是每一列要聚会元素的索
            引
         # 下个状态的最大 Q值
        max_next_q_values = self.target_q_net(next_states).max(1)[0].view(
             -1, 1)
        q_targets = rewards + self.gamma * max_next_q_values * (1 - dones)
        # TD误差目标
        dqn_loss = torch.mean(F.mse_loss(q_values, q_targets))
        # 均方误差损失函数
        self.optimizer.zero_grad()
        # PyTorch中默认梯度会累积,这里需要显式将梯度置为0
        dqn_loss.backward()  # 反向传播更新参数
        self.optimizer.step()

        if self.count % self.target_update == 0:
            self.target_q_net.load_state_dict(
            self.q_net.state_dict())  # 更新目标网络
            self.count += 1
```

有了这些基本组件之后，接来下开始实现 DQN 算法。

```
lr = 2e-3
num_episodes = 500
hidden_dim = 128
gamma = 0.98
epsilon = 0.01
target_update = 10
buffer_size = 10000
minimal_size = 500
batch_size = 64
device = torch.device("cuda") if torch.cuda.is_available() else torch.device(
    "cpu")

env_name = 'CartPole-v0'
env = gym.make(env_name)
random.seed(0)
np.random.seed(0)
env.seed(0)
torch.manual_seed(0)
replay_buffer = ReplayBuffer(buffer_size)
state_dim = env.observation_space.shape[0]
action_dim = env.action_space.n
agent = DQN(state_dim, hidden_dim, action_dim, lr, gamma, epsilon,
            target_update, device)

return_list = []
```

```
for i in range(10):
    with tqdm(total=int(num_episodes / 10), desc='Iteration %d' % i) as pbar:
        for i_episode in range(int(num_episodes / 10)):
            env.render()
            episode_return = 0
            state = env.reset()
            done = False
            while not done:
                action = agent.take_action(state)
                next_state, reward, done, _ = env.step(action)
                replay_buffer.add(state, action, reward, next_state, done)
                state = next_state
                episode_return += reward
                # 当buffer数据的数量超过一定值后,才进行Q网络训练
                if replay_buffer.size() > minimal_size:
                    b_s, b_a, b_r, b_ns, b_d = replay_buffer.sample(batch_size)
                        # 调用上文中的类方法 replay_buffer.sample
                    transition_dict = {
                        'states': b_s,
                        'actions': b_a,
                        'next_states': b_ns,
                        'rewards': b_r,
                        'dones': b_d
                    }
                    agent.update(transition_dict)
            return_list.append(episode_return)
            if (i_episode + 1) % 10 == 0:
                pbar.set_postfix({
                    'episode':
                    '%d' % (num_episodes / 10 * i + i_episode + 1),
                    'return':
                    '%.3f' % np.mean(return_list[-10:])
                })
            pbar.update(1)
```

Double DQN

现在我们已了解到 DQN 利用了卷积神经网络表示行为值函数，并利用了经验回放和单独设立目标网络这两个技巧。但是 DQN 无法克服 Q-learning 本身所固有的缺点：过估计，即，估计的值函数比真实值函数要大。Q-learning 之所以存在过估计的问题，主要源于 Q-learning 中的最大化操作。具体来说，不论是表格型行为值函数的更新公式：

$$Q\left(s_t, a_t\right) \leftarrow Q\left(s_t, a_t\right)+\alpha\left[R_t+\gamma \max_{a'} Q\left(s_{t+1}, a'\right)-Q\left(s_t, a_t\right)\right],$$

还是行为值函数逼近下的更新公式：

$$w_{t+1} \leftarrow w_t + \alpha_t \left(R_{t+1} + \gamma \max_{a'} Q\left(s_{t+1}, a'; w_t\right) - Q\left(s_t, a_t; w_t\right) \right) \nabla_w Q\left(s_t, a_t; w\right) | w = w_t,$$

其中都有 max 操作。max 操作使得估计的值函数比真实值函数大。如果值函数中每一点的值都被过估计了相同的幅度，即过估计量是均匀的，那么基于最大的值函数找到的最优策略保持不变。也就是说, 在这种情况下, 即使值函数被过估计了, 也不影响最优的策略。然而, 在实际情况中, 过估计量并非是均匀的, 因此值函数的过估计会影响最终的策略决定, 从而导致最终的策略并非最优。为了解决值函数过估计的问题, Hasselt 提出了 Double Q-learning 的方法：将动作的选择和动作的评估分别用不同的值函数来实现。我们先介绍一般 Q-learning 的动作选择和评估动作。

- 动作选择：在 Q-learning 的值函数更新中, TD 目标为

  $$Y_t^Q = R_{t+1} + \gamma \max_{a'} Q\left(s_{t+1}, a'; w_t\right)$$

  在求 Y_t^Q 时, 我们首先需要选择一个动作 a^*, 该动作 a^* 使得在状态 s_{t+1} 处 $Q\left(s_{t+1}, a\right)$ 最大。

- 动作评估：选出 a^* 后, 利用 a^* 处的行为值函数构造 TD 目标。

可以看出一般的 Q-learning, 其动作选择和评估动作在同一个参数 w_t 下进行。Double Q-learning 考虑用不同的行为值函数选择和评估动作，其参数并不一样。动作选择所用的动作值函数为

$$\arg\max_{a'} Q\left(s_{t+1}, a'; w_t\right),$$

此时动作值函数网络的参数为 w_t。当选出最大的动作 a^* 后, 动作评估公式为

$$Y_t^{\text{DoubleQ}} = R_{t+1} + \gamma Q\left(s_{t+1}, a^*; w_t'\right)$$

此时动作评估所用的动作值函数网络参数为 w_t'。对应的 Double Q-learning 的 TD 目标为

$$Y_t^{\text{DoubleQ}} = R_{t+1} + \gamma Q\left(s_{t+1}, \underset{a'}{\text{argmax}} Q\left(s_{t+1}, a'; w_t\right); w_t'\right).$$

将 Double Q-learning 的思想应用到 DQN 中, 则得到 Double DQN 算法, 其伪代码与 DQN 一致，仅将 DQN 算法中的 TD 目标修改为

$$Y_t^{\text{DoubleQ}} = R_{t+1} + \gamma Q\left(s_{t+1}, \underset{a'}{\text{argmax}} Q\left(s_{t+1}, a'; w_t\right); w_t^-\right).$$

优先回放（Prioritized Replay）

DQN 的成功归结于经验回放和独立的目标网络。Double DQN 改进了 Q-learning 中的 max 操作，但经验回放仍然采用均匀分布。考虑到智能体的经验即经历过的数据, 对于智能体的学习并非具有同等重要的意义，比如，智能体在某些状态的学习效率比其他状态的学习效率高。所以经验回放利用均匀分布采样并没有高效的利用数据。优先回放的基本思想就是打破均匀采样, 赋予学习效率高的状态以更大的采样权重。

如何选择权重呢? 值函数更新中 TD 误差有着重要作用，我们的目标就是让 TD 误差尽可能小, 如果 TD 误差比较大, 意味着我们当前的行为函数离目标函数差距还很大, 应多进行更新, 因此基于 TD 误差来设置权重。我们设样本 i 处的 TD 误差为 δ_i，令样本处的采样概率为

$$P(i)=\frac{p_i^\alpha}{\sum_k p_k^\alpha}$$

其中 p_i 是样本 i 的优先级，由 TD 误差 δ_i 决定，指数 $\alpha\in[0,1]$ 是决定优先级化的程度，当 $\alpha=0$ 时，退化成均匀采样。p_i 的设置一般有两种方法, 第一种方法是 $p_i=|\delta_i|+\epsilon$，ϵ 是一个很小的正常数为了使 TD 误差为 0 的特殊边缘样本也能够被抽取; 第二种方法是 $p_i=\frac{1}{\mathrm{rank}(i)}$，其中 $\mathrm{rank}(i)$ 根据 $|\delta_i|$ 的排序得到。

当我们采用优先回放的概率分布采样时, 动作值函数的估计值是一个有偏估计。因为采样分布与动作值函数的分布是两个完全不同的分布, 为了矫正这个偏差, 引入重要性采样系数 $\theta_i=\left(\frac{1}{N}\cdot\frac{1}{P(i)}\right)^\beta$，其中 N 是经验池容量。如果 $\beta=1$，则可以完全补偿优先采样带来的偏差。在实际操作中，其实重要的是最终收敛的值函数无偏，但在训练过程中本来就是不稳定的，所以有点偏差可以容忍，所以一种温和的校正偏差方式是，β 从一个 $\beta_0<1$ 的超参逐渐增大，到训练结束时 $\beta=1$。带有优先回放的 Double DQN 算法伪代码如下。

带有优先回放的 Double DQN 算法

1. 输入: 确定 minibatch 的大小 k, 步长 η，回放周期 K，存储数据的总大小 N，
   常数 α,β, 总时间 T;
2. 初始化回放记忆库 $\mathcal{D}=\varnothing,\triangle=0,p_1=0$;
3. 观测初试状态, 选择动作 $a_0\sim\pi_w(s_0)$;
4. For $t=1,\cdots,T$ do;
5. 利用动作 A 作用于环境，环境返回观测 s_t,R_t,γ_t;
6. 将数据 $(s_{t-1},a_{t-1},R_t,\gamma_t,s_t)$ 存储到记忆库 $\mathcal{D}$ 中，且令其优先级为 $p_t=\max_{i<t}p_i$，
   采用该优先级初始化的目的是保证每个样本至少被利用一次;
7. If $t=0\bmod K$ then （每隔 K 步回放一次）
8. for $j=1,\cdots,k$ do （依次采集 k 个样本）;
9. 根据概率分布 $j\sim P(j)=p_j^\alpha/\Sigma_i p_i^\alpha$ 采样一个样本点;
10. 计算样本点的重要性权重 $\theta_j=(N\cdot P(j))^{-\beta}/\max_i\theta_i$;
11. 计算该样本点处的 TD 偏差，
    $\delta_j=R_j+\gamma_j Q_{w_{\mathrm{target}}}(s_j,\arg\max_a Q(s_j,a))-Q(s_{j-1},a_{j-1})$;
12. 更新该样本的优先级 $p_j\leftarrow|\delta_j|$;
13. 累计权重的改变量 $\Delta\leftarrow\Delta+\theta_j\cdot\delta_j\cdot\nabla_w Q(s_{j-1},a_{j-1})$;
14. end for
15. 处理完 k 个样本后更新参数值 $w\leftarrow w+\eta\cdot\Delta$，重新设置 $\Delta=0$;
16. 偶尔地复制新权重到目标网络中，即 $w_{\mathrm{target}}\leftarrow w$;
17. end if
18. 根据新的策略选择下一个动作，$a_t\sim\pi_w(s_t)$;
19. end for

Dueling DQN

不管是最初的 DQN，还是 DQN 的变体 Double DQN，在值函数逼近时所用的神经网络都是卷积神经网络（图 8.14）。Dueling DQN 则从网络结构上改进了 DQN （图 8.15）。

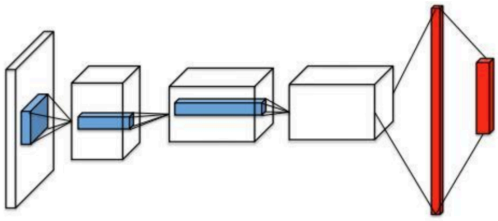


图 8.14: DQN 网络结构

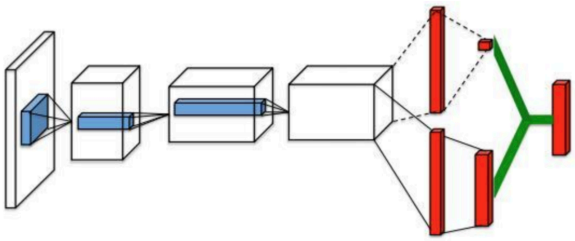


图 8.15: Dueling DQN 网络结构

Dueling DQN 考虑改进网络结构的原因在于：已知 $Q(s,a)$ 表示状态 s 下动作 a 的价值。因为有状态这个条件, $Q(s,a)$ 并不能完全代表状态 a 的价值, 因为有时在某种状态下, 无论做什么动作, 对下一个状态都没有很大的影响：在一个好的状态下, 无论做什么动作, 都能得到很高价值; 在一个很差的状态下, 采取任何动作, 都得到一个较低的价值。因此提出了 Dueling DQN 方法, 希望衡量状态 s 的价值 $V(s)$ 和动作 a 的价值 $A(s,a)$。将状态 s 的价值 $V(s)$ 和动作的价值 $A(s,a)$ 相加得到状态 s 下动作 a 的价值 $Q(s,a)$，即

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\alpha) + A(s,a;\theta,\beta),$$

其中，$V(s;\theta,\alpha)$ 称为状态值函数，$A(s,a;\theta,\beta)$ 称为优势函数, θ 是公共部分的网络参数, α 是价值函数独有的部分网络参数, β 是优势函数独有的部分网络参数。但是上述公式并不能辨识最终输出中 $V(s;\theta,\alpha)$ 和 $A(s,a;\theta,\beta)$ 各自的作用, 为了可以体现这种可辨识性 (identifiability), 实际使用的组合公式如下，

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\alpha) + \left(A(s,a;\theta,\beta) - \frac{1}{|\mathcal{A}|} \sum_{a'} A\left(s,a';\theta,\beta\right) \right).$$

8.4 策略梯度方法

之前介绍的强化学习方法都是基于值函数的，通过近似值函数从而间接得到我们最终需要的策略。然而在强化学习中我们最终的目标是得到一个最优的策略，所以一个自然的问题是有没

有相应的算法可以直接近似策略函数，通过学习参数化的策略从而直接达到我们的目标。本章我们介绍这类重要的方法，我们叫做策略梯度方法。

策略梯度的基本思想是直接参数化策略函数，然后基于一些标量化指标 $J(\theta)$，通过极大化目标函数从而优化策略函数的参数。这些方法试图使性能最大化，因此它们的更新是对目标行数 J 执行梯度上升：

$$\theta_{t+1} = \theta_t + \alpha\widehat{\nabla J(\theta_t)}$$

其中 $\widehat{\nabla J(\theta_t)}$ 是目标函数梯度的一个随机估计，我们把遵循此类模式的方法都称为策略梯度方法，无论方法本身是否同时也近似学习了值函数。基于策略梯度的基本思想，我们推到得到常用的演员-评论家算法，其中“演员”所指学习策略，“评论家”所指学习值函数，进而我们也给出了一些主流策略梯度方法的简单介绍。

8.4.1 策略近似及其优势

在策略梯度方法中，我们只需要直接参数化策略函数，即 $\pi(a \mid s, \boldsymbol{\theta})$, 我们重点关注的是离散动作空间的参数化方法，对于连续动作空间可以通过参数化正态分布输出连续动作空间来实现，感兴趣的作者可以查阅更多学习资料。

如果动作空间离散并且空间不是很大，一种常见的做法是对每个状态输出对应动作的概率，一种常见的参数化方法是基于 soft-max 方法：

$$\pi(a \mid s, \boldsymbol{\theta}) \doteq \frac{e^{h(s,a,\boldsymbol{\theta})}}{\sum_b e^{h(s,b,\boldsymbol{\theta})}}$$

其中输出函数 $h(s, a, \boldsymbol{\theta})$ 可以任意参数化，例如可以通过神经网络，或者通过对特征 $\mathbf{x}(s, a)$ 的线性化参数方法：

$$h(s, a, \boldsymbol{\theta}) = \boldsymbol{\theta}^\top \mathbf{x}(s, a)$$

相较于基于值函数的方法，策略梯度方法具有如下的几个优点：

1. 近似策略可以接近确定性策略，而基于动作选择的 ϵ 贪婪总是有 ϵ 可能性选择随机动作，而最优策略通常是确定性的，所有从逼近的有效性角度来看，策略梯度方法更容易得到一个最优的策略。

2. 很多强化学习的问题用策略梯度方法更容易优化，从而更容易收敛到较优的策略。另外在具有显著函数逼近的问题中，最佳近似策略可能是随机的，而策略梯度方法能够选择具有任意概率的动作。

3. 策略梯度方法能够处理连续的动作空间，基于值函数的方法不能处理。

8.4.2 策略梯度定理

强化学习的目标是为智能体找到一个最优的行为策略从而获取最大的回报，策略梯度方法主要特点在于直接对策略进行建模并且通过优化参数来实现最大化回报的目标，具体来说，回报函数的值收到该策略的直接影响，因此可以利用很多算法来对参数进行优化进而最大化目标函数。

目标函数的定义如下：

$$J(\theta) = \sum_{s\in\mathcal{S}} d^{\pi}(s)V^{\pi}(s) = \sum_{s\in\mathcal{S}} d^{\pi}(s) \sum_{a\in\mathcal{A}} \pi_{\theta}(a \mid s)Q^{\pi}(s,a)$$

其中 $d^{\pi}(s)$ 代表由 π_{θ} 引出的马尔科夫链的平稳分布。如果我们能够直接计算得到目标函数的梯度，就可以利用梯度上升来更新参数了。但是我们发现计算梯度 $\nabla_{\theta}J(\theta)$ 是一件很困难的事情，因为梯度值不仅依赖于动作的选择（由 π_{θ} 直接决定），还依赖于由选择的动作而产生的状态的平稳分布（由 π_{θ} 间接决定）。因为我们发现环境通常是未知的，很难去估计策略的更新对于状态分布造成的影响。

这个是时候我们需要借助策略梯度定理来方便梯度的计算，定理的基本原理是对梯度的形式进行变形使其不依赖于状态分布的导数，从而在很大的程度上简化了梯度 $\nabla_{\theta}J(\theta)$ 的计算，具体的证明可以参考 sutton 的书，最终得到的梯度计算公式如下：

首先要注意的是，状态值函数的梯度可以写成动作值函数

$$\begin{aligned}
\nabla v_{\pi}(s) &= \nabla\left[\sum_{\bar{a}} \pi(a \mid s) q_{\pi}(s,a)\right], \quad \text{for all } s \in \mathcal{S} \\
&= \sum_{\bar{a}} \left[\nabla\pi(a \mid s) q_{\pi}(s,a) + \pi(a \mid s)\nabla q_{\pi}(s,a)\right] \\
&= \sum_{\bar{a}} \left[\nabla\pi(a \mid s) q_{\pi}(s,a) + \pi(a \mid s)\nabla \sum_{s',r} p\left(s', r \mid s, a\right)\left(r + v_{\pi}\left(s'\right)\right)\right] \\
&= \sum_{\bar{a}} \left[\nabla\pi(a \mid s) q_{\pi}(s,a) + \pi(a \mid s) \sum_{s'} p\left(s' \mid s, a\right) \nabla v_{\pi}\left(s'\right)\right] \nabla v_{\pi}\left(s'\right)] \\
&= \sum_{\bar{a}} \left[\nabla\pi(a \mid s) q_{\pi}(s,a) + \pi(a \mid s) \sum_{s'} p\left(s' \mid s, a\right)\right.
\end{aligned}$$

$$\left.\sum_{a'}\left[\nabla\pi\left(a' \mid s'\right) q_{\pi}\left(s', a'\right) + \pi\left(a' \mid s'\right) \sum_{s''} p\left(s'' \mid s', a'\right) \nabla v_{\pi}\left(s''\right)\right]\right]$$

$$= \sum_{x\in S} \sum_{k=0}^{\infty} \Pr(s \to x, k, \pi) \sum_{\bar{a}} \nabla\pi(a \mid x) q_{\pi}(x,a),$$

重复展开后

$$
\begin{aligned}
\nabla J(\boldsymbol{\theta}) &= \nabla v_\pi\left(s_0\right) \\
&= \sum_s \left(\sum_{k=0}^{\infty} \Pr\left(s_0 \rightarrow s, k, \pi\right)\right) \sum_{\bar{a}} \nabla \pi(a \mid s) q_\pi(s, a) \\
&= \sum_s \eta(s) \sum_{\bar{a}} \nabla \pi(a \mid s) q_\pi(s, a) \\
&= \sum_{s'} \eta\left(s'\right) \sum_s \frac{\eta(s)}{\sum_{s'} \eta\left(s'\right)} \sum_{\bar{a}} \nabla \pi(a \mid s) q_\pi(s, a) \\
&= \sum_{s'} \eta\left(s'\right) \sum_s \mu(s) \sum_{\bar{a}} \nabla \pi(a \mid s) q_\pi(s, a) \\
&\propto \sum_s \mu(s) \sum_{\bar{a}} \nabla \pi(a \mid s) q_\pi(s, a)
\end{aligned}
$$

最终得到的梯度计算公式为

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta \sum_{s \in \mathcal{S}} d^\pi(s) \sum_{a \in \mathcal{A}} Q^\pi(s, a) \pi_\theta(a \mid s) \\
&\propto \sum_{s \in \mathcal{S}} d^\pi(s) \sum_{a \in \mathcal{A}} Q^\pi(s, a) \nabla_\theta \pi_\theta(a \mid s)
\end{aligned}
\tag{8.6}
$$

8.4.3 REINFORCE: 蒙特卡洛策略梯度算法

REINFORCE 算法依靠蒙特卡洛方法采样出的样本轨迹从而估计回报，进而通过梯度上升来更新策略的参数，具体的蒙特卡洛策略梯度公式推导如下：

$$
\begin{aligned}
\nabla J(\boldsymbol{\theta}) &\propto \mathbb{E}_\pi\left[\sum_a \pi\left(a \mid S_t, \boldsymbol{\theta}\right) q_\pi\left(S_t, a\right) \frac{\nabla \pi\left(a \mid S_t, \boldsymbol{\theta}\right)}{\pi\left(a \mid S_t, \boldsymbol{\theta}\right)}\right] \\
&= \mathbb{E}_\pi\left[q_\pi\left(S_t, A_t\right) \frac{\nabla \pi\left(A_t \mid S_t, \boldsymbol{\theta}\right)}{\pi\left(A_t \mid S_t, \boldsymbol{\theta}\right)}\right] \\
&= \mathbb{E}_\pi\left[G_t \frac{\nabla \pi\left(A_t \mid S_t, \boldsymbol{\theta}\right)}{\pi\left(A_t \mid S_t, \boldsymbol{\theta}\right)}\right]
\end{aligned}
\tag{8.7}
$$

其中 G_t 是通过采样轨迹的累计奖励得到的对回报的估计，通过计算回报以后就可以直接更新策略的参数, 参数更新规则如下：

$$
\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha G_t \frac{\nabla \pi\left(A_t \mid S_t, \boldsymbol{\theta}_t\right)}{\pi\left(A_t \mid S_t, \boldsymbol{\theta}_t\right)}
$$

请注意，REINFORCE 算法使用的是时间 t 的累积回报，包括直到轨迹结束的所有未来奖励。从这个意义上说，REINFORCE 算法就是蒙特卡洛算法，并且仅在情节的情况下定义良好，所有更新都在轨迹结束后进行回顾。整个的算法流程也十分简洁：

REINFORCE: 蒙特卡洛策略梯度算法
1. 随机初始化策略网络的参数 θ
2. 通过蒙特卡洛方法采样生成当前策略 π_θ 的一条完整的轨迹：$S_1, A_1, R_2, S_2, A_2, ..., S_T$
3. 对于每个时间步 $t = 1, \cdots, T$ ；
4. 计算累积回报 $G_t \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$

5. 通过梯度上升更新参数：$\theta \leftarrow \theta + \alpha\gamma^t G_t \nabla_\theta \ln \pi_\theta (A_t \mid S_t)$

8.4.4 带基线的 REINFORCE 算法

对于上述的 REINFORCE 算法一个直接的改进是从 G_t 中减去一个基准值 b_t 用来在保证偏差不变的情况下减少估计梯度产生的方差，一个重要的例子是我们用动作状态值函数减去状态值函数，这样实际中我们使用优势值：$A(s,a) = Q(s,a) - V(s)$ 进行梯度上升来更新参数。

我们引入一个基准函数到目标函数中，此时梯度可以表示为

$$\nabla J(\boldsymbol{\theta}) \propto \sum_s \mu(s) \sum_a \left(q_\pi(s,a) - b(s)\right) \nabla \pi(a \mid s, \boldsymbol{\theta})$$

基准函数可以是任何函数，甚至是一个随机变量，只要它不随动作 a 变化，这个方程仍然有效，因为减去的量是零：

$$\sum_{\bar{a}} b(s) \nabla \pi(a \mid s, \boldsymbol{\theta}) = b(s) \nabla \sum_{\bar{a}} \pi(a \mid s, \boldsymbol{\theta}) = b(s) \nabla 1 = 0$$

此时，新的参数更新规则如下：

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha \left(G_t - b\left(S_t\right)\right) \frac{\nabla \pi\left(A_t \mid S_t, \boldsymbol{\theta}_t\right)}{\pi\left(A_t \mid S_t, \boldsymbol{\theta}_t\right)}$$

我们最终得到的更新规则是一个包含一般基线的新版本的 REINFORCE。因为基线可以一致为零，所以这个更新是一个严格的一般化 REINFORCE。

实际中，基线的一个自然选择是对状态值的估计 $\hat{v}\left(S_t\right)$ $\mathrm{w} \in \mathbb{R}^m$ 是权值向量。因为 REINFORCE 是一种学习策略参数的蒙特卡罗方法，θ，很自然地使用蒙特卡罗方法学习状态值权重，w。

我们通常采用 $\hat{v}\left(S_t, w\right)$ 来作为 $b\left(S_t\right)$, 最后对应的算法如下：

带基线的 REINFORCE 算法

1. 随机初始化策略网络的参数 θ 和值函数网络参数 w
2. 通过蒙特卡洛方法采样生成当前策略 π_θ 的一条完整的轨迹：$S_1, A_1, R_2, S_2, A_2, ..., S_T$
3. 对于每个时间步 $t = 1, \cdots, T$ ；
    4. 计算累积回报 $G_t \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
    5. 计算带基线后的差值：$\delta \leftarrow G - \hat{v}\left(S_t, w\right)$
    6. 更新值函数网络的参数：$\mathrm{w} \leftarrow w + \alpha^w \delta \nabla \hat{v}\left(S_t, w\right)$
    7. 通过梯度上升更新策略参数：$\theta \leftarrow \theta + \alpha\gamma^t \delta \nabla_\theta \ln \pi_\theta \left(A_t \mid S_t\right)$

8.4.5 演员-评论家 Actor-Critic 及其改进算法

原始的带基线的 REINFORCE 算法需要借助蒙特卡洛方法估计回报 G_t，这对于具有在线增量学习的强化学习任务来说未必是适用的，很自然的改进方式是用时序差分方法代替蒙特卡洛方法进行在线学习，这诱导出了演员-评论家原始算法，本节给出了该重要框架的介绍，以及目前主流的基于演员-评论家的衍生算法的简单描述。

演员-评论家 Actor-Critic 算法

演员-评论家 Actor-Critic 算法可以看成带基线的 REINFORCE 算法的时序差分的版本。具体来说，我们这里用一步回报时序差分代替基于蒙特卡洛方法对整个轨迹的估计：

$$\begin{aligned}\boldsymbol{\theta}_{t+1} &\doteq \boldsymbol{\theta}_t + \alpha\left(G_{t:t+1} - \hat{v}\left(S_t, \mathbf{w}\right)\right) \frac{\nabla \pi\left(A_t \mid S_t, \boldsymbol{\theta}_t\right)}{\pi\left(A_t \mid S_t, \boldsymbol{\theta}_t\right)} \\ &= \boldsymbol{\theta}_t + \alpha\left(R_{t+1} + \gamma \hat{v}\left(S_{t+1}, \mathbf{w}\right) - \hat{v}\left(S_t, \mathbf{w}\right)\right) \frac{\nabla \pi\left(A_t \mid S_t, \boldsymbol{\theta}_t\right)}{\pi\left(A_t \mid S_t, \boldsymbol{\theta}_t\right)} \\ &= \boldsymbol{\theta}_t + \alpha \delta_t \frac{\nabla \pi\left(A_t \mid S_t, \boldsymbol{\theta}_t\right)}{\pi\left(A_t \mid S_t, \boldsymbol{\theta}_t\right)}\end{aligned} \tag{8.8}$$

一步时序差分方法的主要吸引力在于，它们完全在线且递增，同时避免了有效跟踪的复杂性。它们是有效跟踪方法的特殊情况，不像一般情况那样，但更容易理解。最终我们的演员-评论家 Actor-Critic 算法描述如下：

演员-评论家 Actor-Critic 算法
1. 随机初始化策略网络的参数 θ 和值函数网络参数 w
2. 初始状态 s 以及从初始策略中采样动作 $a \sim \pi_\theta(a\|s)$
3. 对于每个时间步 $t = 1, \cdots, T$ ；
4. 采样回报 $r_t \leftarrow R(s,a)$ 和下一步状态 $s' \leftarrow P(s'\|s,a)$
5. 采样下一个动作 $a' \sim \pi_\theta(a'\|s')$
6. 更新策略参数：$\theta \leftarrow \theta + \alpha_\theta Q_w(s,a) \nabla_\theta \ln \pi_\theta(a \mid s)$
7. 对于当前步计算 TD 误差：$\delta_t = r_t + \gamma Q_w\left(s', a'\right) - Q_w(s,a)$
6. 更新值函数网络的参数：$w \leftarrow w + \alpha_w \delta_t \nabla \hat{Q}_w\left(s,a\right)$
7. 更新当前动作 $a \leftarrow a'$ 和状态 $s \leftarrow s'$

其中是 α_θ 和 α_w 两个预先定义的分别用来更新策略以及值函数的更新步长。

DDPG 算法

DDPG（Lillicrap et al. 2015）是深度确定性策略梯度（Deep Deterministic Policy Gradient）的缩写，是一个结合了 DPG 以及 DQN 的无模型离线演员-评论家算法。从 DQN 这边，DDPG 借入了经验回放机制以及目标网络两个核心思想，不同之处是 DDPG 中演员-评论家的目标网络的参数实行的是软更新（“保守策略迭代”）：

$$\theta' = \tau\theta + (1-\tau)\theta'$$

其中 $\tau << 1$。：他们修改了 DQN 中最初使用的目标网络的更新频率。在 DQN 中，目标网络每隔几千步就会更新一次训练网络的参数。Lillicrap et al. 发现实际上更好的方法是让目标网络缓慢地跟踪训练过的网络，每次更新训练过的网络后，用行动者和批评家的滑动平均值更新目标网络的参数。使用该更新规则，目标网络总是比训练过的网络 “晚”，为 q 值的学习提供了更大的稳定性，从 DPG 借鉴的关键思想是行为人（Actor）的策略梯度，并通过常规的 Q-learning 和目标网络来学习：

$$J(\varphi) = \mathbb{E}_{s\sim\rho_\mu}\left[\left(r\left(s,a,s'\right) + \gamma Q_{\varphi'}\left(s'\mu_{\theta'}\left(s'\right)\right) - Q_\varphi(s,a)\right)^2\right]$$

另外 DDPG 对于确定的的策略网络的输出加入额外的噪音 ξ 来鼓励对环境的探索。

$$a_t = \mu_\theta(s_t) + \xi$$

PPO 算法

之前主流的 TRPO 算法的构造相对复杂，我们想要去实现一个类似的约束并且取得更好或者接近的结果，因此，近端策略优化（proximal policy optimization ，PPO）算法就被提出了，该算法在实现相似性能的同时通过使用一个截断的替代目标函数来简化 TRPO。

PPO 算法的设计首先是通过重新整理 TRPO 的替代损失函数为如下形式：

$$L^{\mathrm{CPI}}(\theta) = \mathbb{E}_t\left[\frac{\pi_\theta(s_t,a_t)}{\pi_{\theta_{\mathrm{old}}}(s_t,a_t)} A_{\pi_{\theta_{\mathrm{old}}}}(s_t,a_t)\right] = \mathbb{E}_t\left[\rho_t(\theta) A_{\pi_{\theta_{\mathrm{old}}}}(s_t,a_t)\right]$$

其中 A 为优势函数。如果不添加任何约束求解上述优化问题可能会导致训练过程中的不稳定，因此为了增加训练稳定性，我们需要对策略比值的波动进行乘法，使其在一定区间范围内波动，对此我们通过剪切函数将其约束在 $1-\epsilon$ 和 $1+\epsilon$ 之间：

$$L^{\mathrm{CLIP}}(\theta) = \mathbb{E}_t\left[\min\left(\rho_t(\theta) A_{\pi_{\theta_{\mathrm{old}}}}(s_t,a_t), \mathrm{clip}\left(\rho_t(\theta), 1-\epsilon, 1+\epsilon\right) A_{\pi_{\theta_{\mathrm{old}}}}(s_t,a_t)\right)\right]$$

与 TRPO 相比，PPO 的主要优势在于它的简单性: 通过使用随机梯度下降或 Adam 等一阶方法，可以直接使被裁剪目标最大化。PPO 算法在各类游戏中取得了非常优秀的结果，已经成为了优先考虑的强化学习算法之一。

SAC 算法

软演员-评论家算法（Soft Actor-Critic，SAC，Haarnoja et al. 2018）将策略的熵度量纳入回报函数中用以鼓励探索：我们希望学习到一种尽可能随机行动的策略，同时仍然能够在任务中完成目标。它是一个遵循最大熵强化学习框架的离线演员-评论家模型。

基于最大熵 RL 框架，Haarnoja 等通过扩展目标的定义，提出了软 Q-learning：

$$J(\theta) = \sum_{t=1}^{T} \mathbb{E}_{(s_t,a_t)\sim\rho_{\pi_\theta}}\left[r(s_t,a_t) + \alpha\mathcal{H}\left(\pi_\theta(.\mid s_t)\right)\right]$$

策略的训练目标是同时最大化期望累积回报以及策略的熵度量。

在这个基于轨迹的公式中，智能体寻求的策略是最大化完整轨迹的熵，而不是每个状态下策略的熵。这是一个非常重要的区别: 智能体不仅搜索一个具有高熵的策略，而且搜索一个具有高熵的状态的策略，即智能体最不确定的状态。这就允许了非常有效的探索策略，智能体将尝试着减少其对环境的不确定性，并收集比简单地寻找一个好的政策时更多的信息。

Haarnoja et al 提出了软演员-评论家算法 (SAC)，允许有随机行为者 (与 DDPG 相反)，同时比政策上的方法 (如 PPO) 更优，样本效率更高。

SAC 基于软 Q-learning 来实现这些改进。准确来说，SAC 旨在学习三个函数：

1. 由 θ 参数化的策略函数 π_θ。

2. 由 w 参数化的软 Q 值函数 Q_w。

3. 由 ψ 参数化的软 Q 值函数 V_ψ。

通过基于最大熵策略的原则下对三个网络进行交互训练，SAC 算法也取得了非常令人瞩目的效果。具体的训练过程可以参考相应的论文。

第 9 章　自然语言处理

9.1　自然语言处理概述

9.1.1　什么是自然语言处理

正如比尔盖茨所说，“语言理解是人工智能领域皇冠上的明珠”。传统机器学习模型解决的问题大部分是基于结构化数据，即由二维表结构来逻辑表达和实现的数据，并严格地遵循数据格式与长度规范。例如：常见的存储于 Excel 文档中的数据即是数据的一种结构化表达。然而，随着神经网络与深度学习的快速发展，模型的训练所需要的数据量亦在惊人的增加，传统的结构化数据已经难以支撑模型的学习。故而人们将目标转向了从蕴含大量信息的互联网中提取所需的数据，但将信息转换并储存为具有统一格式的数据表示并非易事。网络中绝大多数的数据都是非数据化结构，例如：视频、音频、图片、文本等，这些信息无法用数字或者统一的结构所表示。对于非结构化数据，文本的数量是最多的，它虽然没有图片和视频占用的空间大，但其所蕴含的信息量的确无可比拟。为了能够分析和利用这些文本信息，自然语言处理扮演着至关重要的角色，即通过自然语言处理技术，将文本信息转换成能让机器所能理解的语言并学习其中的信息来完成各种任务。

自然语言处理 (Natural Language Processing,NLP) 是以语言为对象，利用计算机技术来分析、理解和处理自然语言的一门学科，即把计算机作为语言研究的强大工具，在计算机的支持下对语言信息进行定量化的研究，并提供可供人与计算机之间能共同使用的语言描写。

自然语言处理主要包括两大任务：自然语言理解 (Natural Language Understanding, NLU) 和自然语言生成 (Natural Language Generation,NLG)。前者想实现人与计算机之间进行自然语言通信，即使得计算机能够明白人类自然语言含义，并完成相应任务。后者致力于赋予计算机以人类自然语言的形式，表达给定的意图、思想等的能力。在此背景下，衍生出大量的子任务，而这些子任务各自又构成了并非完全独立的研究领域。例如，在自然语言理解任务下，有命名体识别（Named Entity Recognition）问题、信息抽取（Information Extraction）问题、情感分析（Sentimental Analysis）问题等。在自然语言生成任务下，有机器翻译（Machine Translation）问题、自动问答（Question Answering）问题、自动摘要（Text Summarization）问题等。这些问题所涵盖的领域，共同构成了人类自然语言实际运用过程中所遇到的各种任务和情况。由此可见，自然语言处理是一个非常庞大的研究领域。

9.1.2 自然语言处理的发展阶段

自然语言处理主要历经了三个发展阶段：基于规则建模的方法、基于统计建模的方法和基于深度学习的方法。

20 世纪 90 年代之前，自然语言处理主要是基于规则的方法建立模型。语言学家总结出自然语言运用中蕴含的各种规律，并基于这些规律来分析新的句子。以机器翻译为例，如果输入的句子能够符合事先设定好的规律，且假设这些规律都是正确的，那么模型可能给出一定程度上合理的翻译结果。但是，由于自然语言的灵活多变，表达同一语义的句子通常具有多种不同的表达方式。面对这种情况，模型可能会输出不知所云的结果。事实上，通过有限的规则来刻画复杂的语言规律是不可行的办法。因此，基于规则的方法可能在某个特定的小领域能够取得不错的结果，一旦推广到更宽泛的领域，效果往往很不理想。

1990 年至 2012 年，自然语言处理迎来了其发展的第二阶段，即基于统计方法建立模型。在这一阶段中，人们基于统计的方法来处理自然语言，例如 Bag-of-words 模型、Term frequency inverse document frequency（TF-IDF）模型以及引入贝叶斯模型、隐马尔可夫模型、条件随机场模型等来处理自然语言的方法。在这些模型中，人们开始利用被标注的语料数据进行学习。相较于基于规则的模型，统计的方法允许模型有能力探索到隐藏在语言背后的更多规律，这使得统计方法下的模型具有更强的推广能力。

2012 年后，随着词向量这一奠基性概念的提出，自然语言处理正式进入第三阶段的发展，即基于深度学习的模型。2013 年，Word2Vec 模型与世人见面，该模型完全抛弃了人工设定规则和传统统计模型的方法，而是利用神经网络模型，直接学习出每一个单词在一个稠密空间中的向量表达形式。一年之后，GloVe 模型的发表更使得每一个单词的在向量空间中的位置具有了意义。词向量提出的重要意义在于免去了人工去找寻规律，而是通过大量的数据，让机器自己学习出有意义的词表达。有了这些词表达，就拥有了处理句子、段落、文章的能力。2017 年，google 团队提出了更先进的深度神经网络 Bert 模型，该模型在阅读理解的测试中，表现出了超过人类正确率的惊人成绩。

时至今日，基于深度学习的自然语言处理仍是人工智能领域中炙手可热的领域，每年都有层出不穷的模型在自然语言的各个领域中发表。在未来很长的一段时间里，随着网络的快速发展和计算机性能的不断提高，利用深度学习的方法处理自然语言仍将是备受瞩目的研究方向。

9.1.3 自然语言处理中的主要任务

自然语言处理中的主要任务有四类：句法分析，语义分析，信息抽取，文本生成。前三类任务属于自然语言理解，最后一类任务属于自然语言生成。这四类任务由浅及深，从最基本的分析句法结构到了解语句语义，抽取自然语言中蕴含的信息，最终到由计算机生成自然语言。文本生成是自然语言处理的最高层面，这一类任务所想实现的是机器真正的能做到和人通过自然语言进行无障碍对话，即能不仅理解语义，还能理解语境。也就是说，能明白话中的弦外之音，言外之意，能在不同语境中理解讽刺、玩笑等含义。如果能做到这一点，那么可以说这就实现了图灵所定义的人工智能的标准了。

句法分析

在了解句法分析任务前，我们先介绍最基础的词法分析阶段。词法分析阶段的主要任务有：分词（segmentation）、词性标注（Part-of-Speech Tagging）等。

分词任务，通俗的讲是指将句子或短语分为单词的一类任务。对于中文而言，分词是一大难点。例如："一位友好的哥谭市民" 这个短语，可以被分词为 "一位/友好/的哥/谭市民"，也可以被分词为 "一位/友好的/哥谭市民"。再例如，俗语 "一行行行行行，一行不行行行不行" 等，都是机器非常难以准确理解并加以合理区分的。

词性标注是指的是取一个词汇序列并为其中每个单词分配词性，如名词或动词的任务。以英文为例："I can open this can."，"I" 为人称代词，简记为 PRP，第一个 "can" 为情态动词，简记为 MD，"open" 为动词原形，简记为 VB，"this" 为限定词，简记为 DT，第二个 "can" 为名词，简记为 NN。英语中一个重要的词性标记集是 Penn Treebank 标记集，它有 45 个标记，如图9.1所示。

	Tag	Description	Example
Open Class	**ADJ**	Adjective: noun modifiers describing properties	*red*, *young*, *awesome*
	ADV	Adverb: verb modifiers of time, place, manner	*very*, *slowly*, *home*, *yesterday*
	NOUN	words for persons, places, things, etc.	*algorithm*, *cat*, *mango*, *beauty*
	VERB	words for actions and processes	*draw*, *provide*, *go*
	PROPN	Proper noun: name of a person, organization, place, etc..	*Regina*, *IBM*, *Colorado*
	INTJ	Interjection: exclamation, greeting, yes/no response, etc.	*oh*, *um*, *yes*, *hello*
Closed Class Words	**ADP**	Adposition (Preposition/Postposition): marks a noun's spacial, temporal, or other relation	*in, on, by under*
	AUX	Auxiliary: helping verb marking tense, aspect, mood, etc.,	*can, may, should, are*
	CCONJ	Coordinating Conjunction: joins two phrases/clauses	*and*, *or*, *but*
	DET	Determiner: marks noun phrase properties	*a, an, the, this*
	NUM	Numeral	*one, two, first, second*
	PART	Particle: a preposition-like form used together with a verb	*up, down, on, off, in, out, at, by*
	PRON	Pronoun: a shorthand for referring to an entity or event	*she, who, I, others*
	SCONJ	Subordinating Conjunction: joins a main clause with a subordinate clause such as a sentential complement	*that*, *which*
Other	**PUNCT**	Punctuation	; , ()
	SYM	Symbols like $ or emoji	$, %
	X	Other	asdf, qwfg

图 9.1: Penn Treebank 标记集

下面我们来看句子层面的句法分析任务。句法分析的主要任务是识别出句子所包含的句法成分以及这些成分之间的关系。语言学上的句法非常多，不同的句法从不同的方面表达句子结构，这里我们介绍两种句法分析中常用的语法形式：成分句法分析 (Constituent parsing) 与依存句法分析 (Dependency parsing)。

成分句法分析又称为短语结构句法，顾名思义，指的是通过层次化的短语结构来表达一句话，我们以 "Tim bought a fook for Mary." 为例，通过句法成分分析，如下图所示：

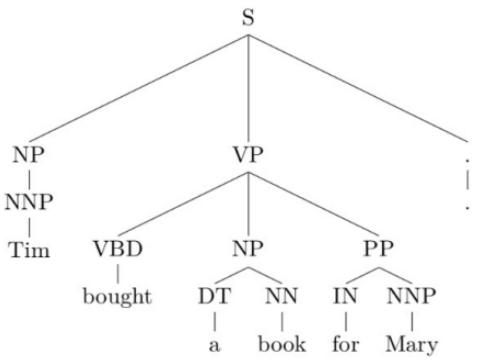


图 9.2: 成分句法分析 (Constituent parsing)

经过句法成分分析，我们可以看出该句子层化的短语结构：第一层为名词短语"a book" 由冠词与名词组成, 第二层为动词短语"bought a book for mary" 由动词加名词短语加介词短语构成，第三层即为整个句子，由名词短语加动词短语加句号构成。

依存句法分析则通过词与词之间的关系构成一句话的结构。在依存句法分析，每一句话有一个词称为根 (root)，除了根以外的其他单词均修饰着句子中除了它本身的唯一词。依存句法分析就通过这种词与词之间的修饰关系分析句子的结构。例如：“Tim bought a book.” 经过依存句法分析，如下图所示：

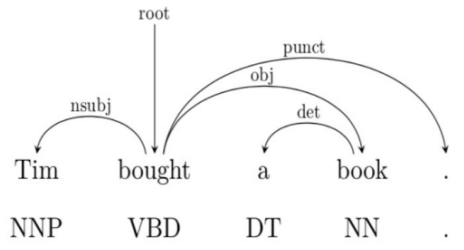


图 9.3: 依存句法分析 (Dependency parsing)

这句话中，“bought” 为根 (root), 其他词语或修饰 root，或修饰其他单词。这样的句法结构称为依存句法分析。

语义分析

与句法分析相同，我们先来看词层面上的任务，这类任务主要包括：词义消歧，隐喻识别，词义关联等。

词义消歧任务指的是如果一个单词具有多种含义，则计算机根据语句为单词赋予其符合当前语境的含义。例如这句话: “I saw a man saw a saw with a saw.”，在这句中，“saw” 出现了四次，但分别有不同的含义，第一个 “saw” 为 see 的过去式，第二个 “saw” 为动词，意指锯，第三个 “saw” 为名词，指的是被男人锯的锯子，第四个 “saw” 也为名词，但它指的是男人正在使

用的锯子。

隐喻识别任务是指计算机通过识别隐喻来理解句子表达的真实含义。如 “Love is a battlefield.” 这句话的 “battlefield” 就是一个隐喻，并不是指真正意义上的战场。

词义关联任务指的给定两个单词，计算机识别这两词词义之间的关联，比如识别同义词、反义词或者两词之间的从属关系等。

语义分析在句子层面上的任务主要为语义角色标注，也被称为对谓词论元结构的分析。谓词可以理解为动作，论元则指动作的参与者或属性。参与者又分为施事与受事，施事指的是动作的实施主体，受事指的是受动作支配的人或事物。例如分析下面这句话: “Tim bought this book for $1.”

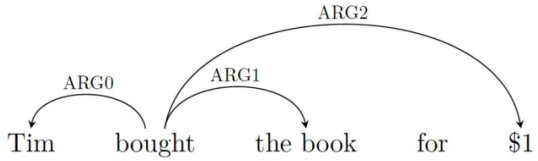


图 9.4: 谓词论元结构 (Predicate-argument relations)

“Bought” 为购买事件，“Tim” 为施事，“the book” 为受事，“$1” 为事件的价格属性。谓词论元结构通过分析事件结构来了解语义。

信息抽取

信息抽取（information extraction），即从自然语言文本中，抽取出特定的事件或事实信息，帮助我们将海量内容自动分类、提取和重构。这些信息通常包括实体 (entity)、关系 (relation)、事件 (event)、情绪 (Sentiment)。例如从新闻中抽取时间、地点、关键人物，或者从技术文档中抽取产品名称、开发时间、性能指标等。

基于抽取的信息不同，信息抽取的任务也分为四类:

1、命名题识别

2、关系抽取

3、事件抽取

4、情感分析

这里我们主要介绍前两种任务：命名体识别与关系抽取。

命名体识别任务指的是识别给定片段中的所有命名实体，并为其分配标签。词性标注可以告诉我们，像 $Janet, Harvard\ University$, 与 $Colorado$ 这样的词都是专有名词, 专有名词是这些词的语法属性。但从语义的角度来看，这些专有名词指的是不同种类的实体:$Janet$ 是一个人，$Harvard\ University$ 是一个组织，$Colorado$ 是一个地点。粗略地说，命名实体就是任何可以用命名实体固有名称引用的东西: 一个人、一个地点、一个组织。命名实体识别 (named entityrecognition, NER) 的任务是找到构成专有名称的文本范围，并标记命名实体识别的类型。四个实体标签是最常见的:PER(人)、LOC(地点)、ORG(组织) 或 GPE(地缘政治实体)。然而，命名实体这个术语通常被扩展为包含本质上不是实体的东西，包括日期、时间和其他种类的时间表达式，甚至像价格这样的数字表达式。以下是一个名命题识别输出的例子:

Citing high fuel prices, [ORG United Airlines] said [TIME Friday] it has increased fares by [MONEY $6] per round trip on flights to some cities also served by lower-cost carriers. [ORG American Airlines], a unit of [ORG AMR Corp.], immediately matched the move, spokesman [PER Tim Wagner] said. [ORG United], a unit of [ORG UAL Corp.], said the increase took effect [TIME Thursday] and applies to most routes where it competes against discount carriers, such as [LOC Chicago] to [LOC Dallas] and [LOC Denver] to [LOC San Francisco].

上面的片段中提到了 5 个组织、4 个地点、2 次时间、1 次人、1 次钱等 13 个实体。下图显示了典型的通用命名实体类型，许多应用程序还需使用特定的实体类型，如蛋白质、基因、商业产品或艺术品。

Type	Tag	Sample Categories	Example sentences
People	PER	people, characters	**Turing** is a giant of computer science.
Organization	ORG	companies, sports teams	The **IPCC** warned about the cyclone.
Location	LOC	regions, mountains, seas	**Mt. Sanitas** is in **Sunshine Canyon**.
Geo-Political Entity	GPE	countries, states	**Palo Alto** is raising the fees for parking.

图 9.5: 命名体分类与标签

关系抽取任务的工作主要为识别实体之间的关系，例如：包含关系，人与单位之间隶属关系，人与人之间的社会关系等等。

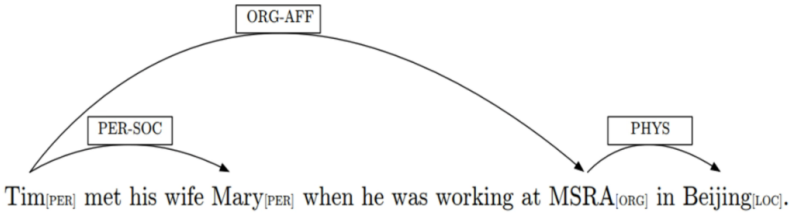


图 9.6: 关系抽取 (Relation extraction)

如上图所示，将这句话进行关系抽取后，我们可以得到 Tim 和 Mary 的社会关系，Tim 和其公司 MSRA 的隶属关系，MSRA 与 Beijing 的位置关系。进一步，实体之间的关系可以构成知识图谱进行利用。

文本生成

文本生成任务是自然语言处理的最高层面。这里列举目前最为常见的两种文本生成任务：机器翻译与句法纠错。

机器翻译相信绝大多数的读者均有所接触，这项任务主要指的是将一种语言的文本转换到另一种语言，是一种典型的文本生成任务。

句法纠错主要应用于如 Word 这类文本输入软件，计算机根据算法为人输入的句子进行句法纠错。还有一类应用情景为作文打分软件，根据句法错误的数量对作文进行评级。

9.2 基于相对频率计数的概率模型

本节我们将介绍基于相对频率计数的概率模型在自然语言处理中的应用。这类概率模型一般在自然语言处理发展的第二阶段中提出，这里我们介绍两种最经典的概率模型：N-gram 语言模型与朴素贝叶斯分类器。

9.2.1 N-gram 语言模型

N-gram 模型简介

在介绍 N-gram 语言模型前，我们不妨来看一句话："现在请把书翻....."。很明显这句话并没有说完，但我们很容易去猜测这句话后面紧跟着的很有可能是"到"或者"开"，但绝不可能是"空调"或者"冰箱"。以此为启发，我们将设计一种模型，为紧接着的下一个词语分配概率，同样也可以为整个句子分配概率。这种预测接下来的词语或者整个句子出现的可能性的情形在生活中非常常见，例如我们去百度搜索"山东"，百度可能会为我们联想"大学"，又或者拼写纠正或病句纠正这样的写作工具，都需要为词语序列或语句分配概率并以此为依据进行预测。

这种为词语序列分配概率的模型被称为语言模型 (language model)。在本章中，我们将介绍一个最简单的模型，它将概率分配给句子和词语序列，即 n-gram。n-gram 是由 n 个词语组成的序列: 2-gram(我们称之为 bigram) 是由两个词语组成的序列，比如"打开/电视"，或"你的/作业"，而 3-gram(a trigram) 是由三个词语组成的序列，比如"我/打开/电视"，或"检查/你的/作业"。我们将介绍如何使用 n-gram 模型来估计 n-gram 中最后一个词语的概率，并将概率分配给整个序列。我们通常使用术语 n-gram(和 bigram 等) 来表示词语序列本身或为其赋值概率的预测模型。

虽然 n-gram 模型比较简单，但它们是理解语言模型基本概念的重要基础工具，同样也是语言处理中最广泛使用的工具之一。

N-gram 理论

从上一节我们可以看出，对于中文而言，如果我们要使用 N-gram 模型，首先要对语句进行分词，这就涉及了一些其他问题，因此后文将使用英文做例，以避免我们在分词上耗费更多的精力。

我们的目的是计算 $P(w|h)$：在给定历史 h 的情况下，一个单词 w 出现的概率。假设历史 h 是 $its\ water\ is\ so\ transparent\ that$, 我们想知道下一个单词 the 出现的概率，即：

$$P(the|its\ water\ is\ so\ transparent\ that).$$

如何计算这个概率，我们有两种方法。第一种方法是通过频率计数: 取一个非常大的语料库，数一下 $its\ water\ is\ so\ transparent\ that$ 次数，然后数一下后面紧着是 the 的次数，即：

$$P(the|its\ water\ is\ so\ transparent\ that) =$$

$$\frac{C(its\ water\ is\ so\ transparent\ that\ the)}{C(its\ water\ is\ so\ transparent\ that)}$$

但这种计算方式算出的概率在很多情况下都不够准确，我们需要注意的是，语言具有创新性，它是在不断发展的。很多新颖的流行语在不断被创造出来，而这类流行语词语组合序列很有可能在语料库中出现的次数为 0。而且如果我们需要例如 *its water is so transparent* 此类整个句子的概率，我们需要把语料库中所有可能的五个单词组成的序列计数出来，这显然是很难实现的。

因此我们引入第二种方法，首先我们定义一些符号。我们使用 $P(the)$ 表示特定随机变量 X_i 取 *the* 即 $P(X_i = \text{“}the\text{”})$ 的概率，用 $w_1..w_n$ 或 $w_{1:N}$ 表示一个 N 个单词的序列 (因此表达式 $w_{1:N-1}$ 表示字符串 $w_1w_2...w_{n-1}$)。对于序列中每个单词具有特定值 $P(X = w_1, Y = w_2, Z = w_3 \ldots W = w_n)$ 我们使用 $P(w_1w_2...w_n)$ 表示其概率。

为了计算 $P(w_1w_2...w_n)$，我们使用概率链式法则对其进行分解：

$$\begin{aligned} P(w_{1:n}) &= P(w_1)\,P(w_2 \mid w_1)\,P(w_3 \mid w_{1:2}) \ldots P(w_n \mid w_{1:n-1}) \\ &= \prod_{k=1}^{n} P(w_k \mid w_{1:k-1}) \end{aligned}$$

但仅此还不够，我们需要知道如何具体计算上式的条件概率。为此我们引入 n-gram 模型，n-gram 模型的思路是仅用前几个单词代替整个句子来估计下一个单词的概率。例如，如果我们使用 bigram 模型，将使用条件概率 $p(w_n|w_{n-1})$ 来近似 $P(w_n|w_{1:n-1})$.

一个词的概率只取决于前一个词的假设叫做马尔可夫假设。马尔可夫模型是一类概率模型，它假设我们可以预测未来某个单元的概率，而不用太过关注过去。我们可以将 bigram(把一个词看成过去) 推广为 trigram(把两个词看成过去)，进一步推广为 n-gram(把 n−1 个词看成过去)。因此，利用 bigram 模型假设，我们将对 $P(w_1, w_2 \ldots w_n)$ 进行化简：

$$P(w_{1:n}) \approx \prod_{k=1}^{n} P(w_k \mid w_{k-1})$$

至此我们只需要计算 bigram 或者 n-gram 概率即可得到整个句子的概率。估计 n-gram 概率的一种直观的方法为极大似然估计 (MLE)。我们通过语料库中获取计数，并将计数归一化，使它们位于 0 和 1 之间，从而得到 n-gram 模型概率的 MLE 估计。

例如，如果给定一个单词 x，要计算一个单词 y 的特定双单词组合概率，我们将计算双单词组合 $C(xy)$ 的计数，并通过共享同一个第一个单词 x 的所有 bigram 的和进行归一化:

$$P(w_n \mid w_{n-1}) = \frac{C(w_{n-1}w_n)}{\sum_w C(w_{n-1}w)}$$

所有以某一特定单词 w_{n-1} 开头的 bigram 之和，必须等于该单词 w_n−1 的 unigram 之和，因此我们得到：

$$P(w_n \mid w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})}$$

最后，需要注意的是如果使用 bigram 模型，为了给出句子起始第一个词语的前文，我们需要再语料库中的每个句子开头加上起始符 <s>，同理在每个句子结尾加上 </s> 以提供末尾单词的后文。

有了以上的公式，我们利用语料库将很容易的计算出整个句子出现的概率 $P(w_1, w_2 \ldots w_n)$. 下面我们看一个例子，我们使用伯克利餐厅项目 (Berkeley Restaurant Project)

的数据作为语料库，该项目是上个世纪的一个对话系统，用于回答有关加州伯克利餐厅数据库的问题 (Jurafsky et al.，1994)。下面是一些用户查询示例的规范化文本 (网站上有 9332 个句子的示例):

can you tell me about any good cantonese restaurants close by

mid priced thai food is what i' m looking for

tell me about chez panisse

can you give me a listing of the kinds of food that are available

i' m looking for a good place to eat breakfast

when is caffe venezia open during the day

下图显示了来自伯克利餐厅项目的一段 bigram 计数。尽管我们选择了一些高频词语来相互衔接，但其中仍有大部分值为 0，但如果使用随机 7 个单词组成一个集合，矩阵将会更加稀疏。

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

图 9.7: bigram 计数

下图为 unigram 计数与归一化后的 bigram 概率 (ungram 计数如下图):

i	**want**	**to**	**eat**	**chinese**	**food**	**lunch**	**spend**
2533	927	2417	746	158	1093	341	278

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

图 9.8: unigram 计数与 bigram 概率

我们再给出一些其他有用的概率:

$$P(i \mid <\text{s}>) = 0.25 \qquad P(\text{ english } \mid \text{ want }) = 0.0011$$
$$P(\text{ food } \mid \text{ english }) = 0.5 \qquad P(<\text{/s}> \mid \text{ food }) = 0.68$$

现在我们就可以通过简单地将相应的 bigram 概率相乘来计算出 $I\ want\ English\ food$ 出现的概率，如下所示:

$$P(\langle \text{s}\rangle i\ want\ english\ food\langle/\text{s}\rangle)$$

$$= P(i\ |<\text{s}>)P(want\mid i)P(english\mid want)$$

$$P(food\mid english)P(<\ /\text{s}>|\ food)$$

$$= 0.25\times 0.33\times 0.0011\times 0.5\times 0.68 = .000031$$

困惑度

为了对语言模型进行评估，我们使用困惑度 (perplexity，简称 PP) 作为度量标准。对于一个测试集 $W = w_1w_2...w_N$，我们定义困惑度:

$$\begin{aligned}\text{PP}(W) &= P\left(w_1w_2\dots w_N\right)^{-\frac{1}{N}}\\ &= \sqrt[N]{\frac{1}{P\left(w_1w_2\dots w_N\right)}}\\ &= \sqrt[N]{\prod_{i=1}^{N}\frac{1}{P\left(w_i\mid w_1\dots w_{i-1}\right)}}\end{aligned}$$

若我们使用 bigram 模型，则可进一步化简得:

$$\text{PP}(W) = \sqrt[N]{\prod_{i=1}^{N}\frac{1}{P\left(w_i\mid w_{i-1}\right)}}$$

显然，词序列的条件概率越高，困惑度越低。因此，根据语言模型，最小化困惑相当于最大化测试集概率。此外，还有另一种思考困惑度的方式: 加权平均分支因子 (weighted average branching factor)。分支因子 (branching factor) 指的是能放置在任何一个词后面的词语的个数。

我们通过一个例子来说明带权重的平均 branching factor 的意义: 若某个任务是识别数字，包括 0,1,...,9。在没有任何假设前提的情况下，每个数字出现的概率都是一样的，概率均为 0.1。此时该模型的困惑度 PP 为:

$$\begin{aligned}\text{PP}(W) &= P\left(w_1w_2\dots w_N\right)^{-\frac{1}{N}}\\ &= \left(\frac{1}{10}^{N}\right)^{-\frac{1}{N}}\\ &= \frac{1}{10}^{-1}\\ &= 10\end{aligned}$$

此时每个数字出现概率相同，权重相同。但如果假设数字 0 出现的非常频繁，在一个训练集中，0 出现了 91 词，其它数字各只出现了 1 次。现在有这样一个测试集: 0 0 0 0 0 3 0 0 0 0，它的困惑度肯定是比 10 低的，因为大部分时候下一个数字是 0。尽管分支因子 (branching factor) 仍然是 10，但加权求和的结果（困惑度）会更小。

下面我们看一个利用困惑度比较不同的 n-gram 语言模型的实例，我们利用《华尔街日报》上的 3800 万个单词 (包括句子开头的代号) 来训练 ungram、bigram 和 triggram 模型。然后计算其困惑度。下表显示了不同 n-gram 模型在一个 150 万词的 WSJ 测试集上的困惑度。

	Unigram	Bigram	Trigram
Perplexity	962	170	109

图 9.9: 三种 n-gram 模型在 WSJ 测试集上的困惑度

如图所示，n-gram 给予我们单词序列的信息越多，困惑度就越低。但在我们利用困惑度对语言模型进行评估时，我们需要注意只有利用完全相同的词汇库进行训练与测试，且测试集没有被语言模型训练过的情况下，困惑度的比较才有意义。

平滑处理

对于 N-gram 模型，我们仍然不可避免会遇到这样的问题：一个词汇序列从未在训练集中出现过，但在测试集中出现了，此时若按照以往的公式简单的将概率相乘进行计算，则会出现概率为 0 的情况。为了避免此类情况，我们需要将那些频繁出现的词汇概率分给那些我们从未见过的词汇组合。这里我们简单介绍一种处理方式：拉普拉斯平滑 (laplace smoothing)。拉普拉斯平滑是将所有的 n-gram 计数加 1，然后再将它们归一化为概率。所有过去出现次数是 0 的现在都是 1，1 的计数是 2，以此类推。若原词汇表中有 V 个单词，则经过拉普拉斯平滑处理后 unigram 概率为：

$$P_{\text{Laplace}}\left(w_i\right)=\frac{c_i+1}{N+V}$$

比起同时改变分子和分母，我们可以通过直接定义一个调整因子（adjusted count）c^* 来描述平滑算法对于分子的影响。调整后的因子更容易直接与 MLE 计数进行比较，并且可以通过对 N 进行标准化将其转化为概率（如 MLE 计数）。要定义这个因子，除了要将分子加 1 外，我们还需要乘以归一化因子 $\frac{N}{N+V}$：

$$c_i^*=\left(c_i+1\right)\frac{N}{N+V}$$

接着我们利用 N 进行标准化，来将 c^* 转化为概率值 P_i^* 。我们可以将平滑理解为，减少非零的计数（即打折，discounting），从而给零值数更大的概率。因此，比起用折扣数 c^*，我们一般用相对折扣 d_c（即折扣计数与原始计数之比）来描述平滑算法，如下：

$$d_c=\frac{c^*}{c}$$

在我们理解了 unigram 模型的平滑处理后，对于 bigram 概率，我们有：

$$P_{\text{Laplace}}^*\left(w_n \mid w_{n-1}\right)=\frac{C\left(w_{n-1}w_n\right)+1}{\sum_w\left(C\left(w_{n-1}w\right)+1\right)}=\frac{C\left(w_{n-1}w_n\right)+1}{C\left(w_{n-1}\right)+V}$$

现在我们尝试对伯克利餐厅项目的 bigram 模型进行平滑处理。下图展示了进行加一平滑后的结果：

	i	want	to	eat	chinese	food	lunch	spend
i	6	828	1	10	1	1	1	3
want	3	1	609	2	7	7	6	2
to	3	1	5	687	3	1	7	212
eat	1	1	3	1	17	3	43	1
chinese	2	1	1	1	1	83	2	1
food	16	1	16	1	2	5	1	1
lunch	3	1	1	1	1	2	1	1
spend	2	1	2	1	1	1	1	1

图 9.10: 加一后的 bigram 计数

在 unigram 计数进行词汇数量 V=1446 的扩充后，我们可以得出 bigram 平滑概率，如下图：

	i	want	to	eat	chinese	food	lunch	spend
i	0.0015	0.21	0.00025	0.0025	0.00025	0.00025	0.00025	0.00075
want	0.0013	0.00042	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00078	0.00026	0.0013	0.18	0.00078	0.00026	0.0018	0.055
eat	0.00046	0.00046	0.0014	0.00046	0.0078	0.0014	0.02	0.00046
chinese	0.0012	0.00062	0.00062	0.00062	0.00062	0.052	0.0012	0.00062
food	0.0063	0.00039	0.0063	0.00039	0.00079	0.002	0.00039	0.00039
lunch	0.0017	0.00056	0.00056	0.00056	0.00056	0.0011	0.00056	0.00056
spend	0.0012	0.00058	0.0012	0.00058	0.00058	0.00058	0.00058	0.00058

图 9.11: bigram 平滑概率

类似的，我们利用 c^* 来观察计数的变化，并计算 bigram 平滑中 c^* 的计数，计算公式如下。图9.12表示了重构后的计数。

$$c^*\left(w_{n-1}w_n\right) = \frac{\left[C\left(w_{n-1}w_n\right)+1\right] \times C\left(w_{n-1}\right)}{C\left(w_{n-1}\right)+V}$$

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

图 9.12: bigram 重构计数值

观察上图可知，拉普拉斯平滑成功令频率从高频词语序列向频率为 0 的词语序列转移，解决了我们最初的问题。但仔细观察可知，拉普拉斯平滑使 bigram 计数发生了非常大的变化，

$C(want\ to)$ 从 609 减少到了 238！我们也可以在 bigram 概率中看出：$P(to|want)$ 从未平滑情况下的 0.66 下降到平滑情况下的 0.26。计数和概率的急剧变化是因为有过多的概率被分配给了所有概率为 0 的词语序列。为了解决这个问题，在拉普拉斯平滑的基础上，又提出了回退与差值，Kneser-Ney 平滑等等处理方式。

9.2.2 朴素贝叶斯分类器

文本分类简介

能够完成分类任务是人工智能的核心功能之一。决定哪个字母、单词或图像传达给人类，识别人脸或声音或者给邮件分类，给作业打分; 这些都是人工智能给输入信息分类的情景。

在本章中，我们将介绍朴素贝叶斯算法并将其应用于文本分类，即为整个文本或文档分配标签或类别的任务。情感分析是分本分类中常见的任务，作者对某个对象表达积极或消极的倾向。网络上对电影、书籍或产品的评论表达了作者对产品的感情，而社论或政治文章表达了对候选人或政治行动的情感。因此，从市场营销到政治，提取消费者或公众情绪都属于情感分析领域。

现在我们来考虑一个最简单的二元分类情感分析，即将文本分类为积极 (positive) 与消极 (negative)。文本中包含的词语会提供很好的线索。例如，考虑以下从电影和餐馆的正面和负面评论中提取的短语。像 great，richly，awesome，和 pathetic，以及 awful 和 ridiculous 这样的词都具有丰富的感情色彩:

+ ...zany characters and richly applied satire, and some great plot twists

− It was pathetic. The worst part about it was the boxing scenes...

+ ...awesome caramel sauce and sweet toasty almonds. I love this place!

− ...awful pizza and ridiculously overpriced...

同样的，文本分类的应用情景还有很多，比如给邮件分类：判断是否为垃圾邮件；给语言分类：判断是英语、汉语等等；甚至作者归属（确定文本作者）都属于文本分类的范畴。

语言处理中的大多数分类都是通过有监督的机器学习来完成的，这将是本节剩余部分的主题。形式上，监督分类的任务是取一个输入 x 和一个固定的输出类集合 $Y = y_1, y_2 \ldots$，并返回一个预测类 $y \in Y$。对于文本分类，我们会用 c(表示 “类”）而不是 y 作为输出变量，用 d(表示 “文档”）而不是 x 作为输入变量。在有监督的情况下，我们有一个由 N 个文档组成的训练集，每个文档都手工标注了一个正确的类:$(d_1, c_1) \ldots. (d_N, c_N)$。我们的目标是学习一种分类器，它能够将一个新文档 d 映射到它的正确类 $c \in C$。概率分类器还会告诉我们观察结果在这个类中的概率。本节主要介绍利用朴素贝叶斯算法构建的分类器。

朴素贝叶斯分类器

对于一整片文档，我们若要对其进行分类，首先要先寻找文本的特征，也就是对文档的拆分与化简。词袋模型就是表示文本特征的一种方式。给定一篇文档，它会有很多特征，比如文档中每个单词出现的次数、某些单词出现的位置、单词的长度、单词出现的频率……而词袋模型只考虑一篇文档中单词出现的频率 (次数)，用每个单词出现的频率作为文档的特征（或者说用单词出现的频率来代表该文档)。词袋模型的示意图如下:

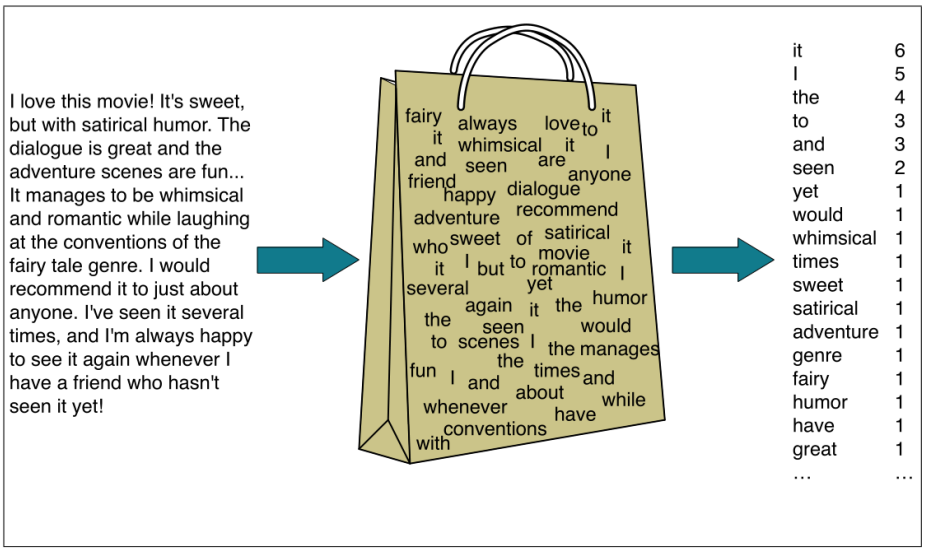


图 9.13: 词袋模型

朴素贝叶斯分类器是一个概率分类器。假设现有的类别 $C = \{c_1\ c_2\ \ldots\ldots c_m\}$。给定一篇文档 d，文档 d 最有可能属于哪个类呢？这个问题用数学公式表示如下：

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} P(c \mid d)$$

$\hat{c}$ 为在所有的类别 $C = \{c_1\ c_2\ \ldots\ldots c_m\}$ 中，能使得条件概率 $P(c|d)$ 值最大的类别。使用贝叶斯公式，将上式转换为：

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} P(c \mid d) = \underset{c \in C}{\operatorname{argmax}} \frac{P(d \mid c)P(c)}{P(d)}$$

我们在比较 $P(c|d)$ 的值时，分母项 $P(d)$ 并没有任何变化，因此我们仅需要使分子项最大即可，即：

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} P(c \mid d) = \underset{c \in C}{\operatorname{argmax}} P(d \mid c)P(c)$$

这个公式由两部分组成，前一部分 $P(d|c)$ 称为似然函数，后一部分 $P(c)$ 称为先验概率。我们利用前文提到的词袋模型来表示文档 d，则文档 d 的特征可以表示为：$d = \{f_1, f_2, f_3 \ldots\ldots f_n\}$，那么这里的特征 f_i 其实就是单词 w_i 出现的频率（次数），公式转化成如下形式：

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} \overbrace{P\left(f_1, f_2, \ldots, f_n \mid c\right)}^{\text{likelihood}} \overbrace{P(c)}^{\text{prior}}$$

为了计算上式的值，我们需要对文档做进一步假设，此假设也通常被称为朴素贝叶斯假设，即假设各个特征之间相互独立，则：

$$P\left(f_1, f_2, \ldots, f_n \mid c\right) = P\left(f_1 \mid c\right) \cdot P\left(f_2 \mid c\right) \cdot \ldots \cdot P\left(f_n \mid c\right)$$

因此公式最终转换为：

$$c_{NB} = \underset{c \in C}{\operatorname{argmax}} P(c) \prod_{f \in F} P(f \mid c)$$

下一步我们将计算先验概率与似然函数，这个过程也可以称为训练朴素贝叶斯过程。对于第一部分先验概率，我们只需使用数据中的频率即可计算。假设训练数据中一共有 N_{doc} 篇文档，只要数一下类别 c 的文档有多少就能计算 $p(c)$ 了，类别 c 的文档共有 N_c 篇，先验概率的计算公式如下：

$$\hat{P}(c) = \frac{N_c}{N_{doc}}$$

下面来考虑第二部分似然函数 $P(f_i|c)$。由于是用词袋模型表示一篇文档 d，对于文档 d 中的每个单词 w_i，找到训练数据集中所有类别为 c 的文档，计数单词 w_i 在这些文档（类别为 c）中出现的次数：$count(w_i, c)$，然后，再计数训练数据集中类别为 c 的文档一共有多少个单词，二者比值即为似然函数的值，公式如下：（其中 V 为词库，若有些单词在词库中，但是不属于类别 C，那么 $count(w, c) = 0$ ）

$$\hat{P}\left(w_i \mid c\right) = \frac{\text{count}\left(w_i, c\right)}{\sum_{w \in V} \text{count}(w, c)}$$

与上一节语言模型中遇到的问题相似，频率为“0”的计数会干扰我们的判断，例如考虑一个二元情感分析模型，在训练数据集中，类别为 positive 的所有文档都没有包含单词 $w_i = fantastic$（$fantastic$ 可能出现在类别为 negative 的文档中）那么 $count(w_i = fantastic, c_i = positive) = 0$ 即有：

$$\hat{P}(\text{ “fantastic” |positive) }) = \frac{\text{count }(\text{ “fantastic”, positive })}{\sum_{w \in V} \text{count}(w, \text{ positive })} = 0$$

则依据贝叶斯假设我们会简单的将所有概率相乘，其中一项为 0 则整体为 0。为此，我们同样需要进行拉普拉斯平滑处理，这里不再详细展开，此时似然公式变为（其中 |V| 是词库中所有单词的个数）：

$$\hat{P}\left(w_i \mid c\right) = \frac{\text{count}\left(w_i, c\right) + 1}{\sum_{w \in V}(\text{count}(w, c) + 1)} = \frac{\text{count}\left(w_i, c\right) + 1}{\left(\sum_{w \in V} \text{count}(w, c)\right) + |V|}$$

下面我们来看一个训练与测试朴素贝叶斯分类器的简单实例。假设训练数据集有五篇文档，其中 Negative 类别的文档有三篇，用符号’-’ 标识；Positive 类别的文档有二篇，用符号’+’ 标识，它们与测试集的内容如下：

	Cat	Documents
Training	-	just plain boring
	-	entirely predictable and lacks energy
	-	no surprises and very few laughs
	+	very powerful
	+	the most fun film of the summer
Test	?	predictable with no fun

图 9.14: 训练集与测试集内容

我们需要考虑的是分类器会将测试集 $predictable\ with\ no\ fun$ 分为 “+” 还是 “-”。为此我们需要计算先验概率与似然函数，先验概率如下：

$$P(-) = \frac{3}{5} \quad P(+) = \frac{2}{5}$$

with 这个词没有出现在训练集中,所以我们完全无视它。训练集中剩下的三个单词 *predictable*、*no* 和 *fun* 的概率由公式计算如下:

$$P(\text{ “predictable” } | -) = \frac{1+1}{14+20} \quad P(\text{ “predictable” } | +) = \frac{0+1}{9+20}$$
$$P(\text{ “no” } | -) = \frac{1+1}{14+20} \quad P(\text{ “no” } | +) = \frac{0+1}{9+20}$$
$$P(\text{ “fun” } | -) = \frac{0+1}{14+20} \quad P(\text{ “fun” } | +) = \frac{1+1}{9+20}$$

进一步可得:

$$P(-)P(S \mid -) = \frac{3}{5} \times \frac{2 \times 2 \times 1}{34^3} = 6.1 \times 10^{-5}$$
$$P(+)P(S \mid +) = \frac{2}{5} \times \frac{1 \times 1 \times 2}{29^3} = 3.2 \times 10^{-5}$$

比较上面两个概率的大小，就可以知道分类器会将 *predictable with no fun* 归为 “-” 类别。

文本分类的评估

在此节我们将介绍评估文本分类性能的几类度量标准。首先我们考虑一些简单的二元文本分类任务。例如，在垃圾邮件检测中，我们的目标是将每个文本标记为属于垃圾邮件类别 (positive “+”) 或不属于垃圾邮件类别 (negative “-”)。因此，对于每个项目 (电子邮件文档)，我们需要知道分类器是否将其称为垃圾邮件。我们还需要知道电子邮件是否真的是垃圾邮件，这就需要我们人为的将这些文本分类，我们将这些正确的人类分类标签称为黄金标签。

为了检验我们的垃圾邮件分类器的性能，我们构建了一个混淆矩阵以可视化算法对于黄金标签的执行情况，使用两个维度 (系统输出和金标签)，具体如下图9.15。表格的右下角是准确性的表达式，它表示了系统正确标记的所有观察结果的百分比。虽然准确性似乎是一个最符合我们直觉的度量标准，但我们通常不会在文本分类任务中使用它。这是因为当类别不平衡时，准确性有时无法准确表达分类器的性能。

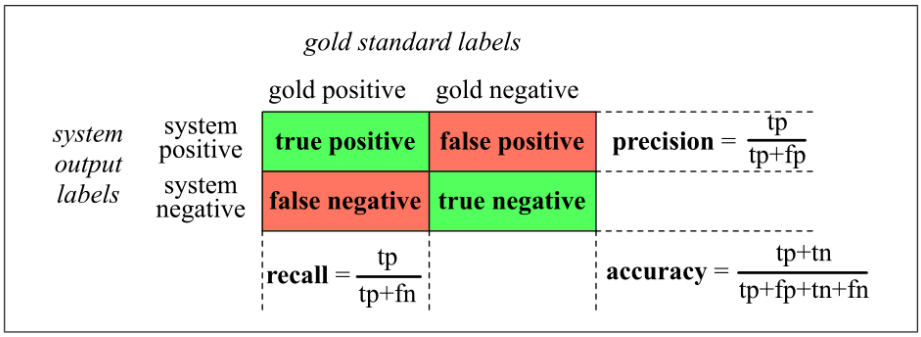


图 9.15: 混淆矩阵

比如，如果我们收到了 10000 封电子邮件，其中垃圾邮件仅有 10 封，而其余邮件均为正常邮件。若此时有一个傻瓜分类器，把所有电子邮件全部记为正常邮件，那么他的准确率能够高达 99.9%！但显而易见这是个毫无用处的分类器。

因此，我们通常使用的指标为图—上的另外两个数字，召回率与精度。精度表示的是被系统标记为阳性的文档中判断正确的百分比。召回率表示的是阳性文档中能被系统正确识别的百分

比。精度和召回率将有助于解决上文中无用的傻瓜分类器的问题。尽管这个傻瓜分类器的准确率高达 99.9%，但它的召回率与精度均为 0，这说明此分类器的性能很差。因此，精度和召回率与准确性不同，强调的因素是: 找到我们应该寻找的东西。

此外，还有许多单一度量标准可以同时包含精度与召回率，这里我们介绍一个最简单的指标，F 测度（F-measure），其定义如下：

$$F_\beta = \frac{(\beta^2 + 1)\, PR}{\beta^2 P + R}$$

参数 β 的设定值基于需要进行改变以区别地加权查全召回率和精度。$\beta > 1$ 时权重更偏向于召回率，$\beta < 1$ 时权重更偏向于精度。$\beta = 1$ 时，精度和召回率均衡看待，这是最常用的度量方式，称为 $F_\beta = 1$ 或 F_1，此时有:

$$\mathrm{F}_1 = \frac{2PR}{P + R}$$

在了解二元分类器的评估后，我们利用相同的思路去评估多元分类任务。多元分类任务在我们的日常生活中非常常见，如词类标注，情绪检测等等。我们仍然以邮件检测为例，但此时将分类拓展为三类，将邮件分为紧急邮件 (urgent)，普通邮件 (normal)，垃圾邮件 (spam)。此时混淆矩阵如下图：

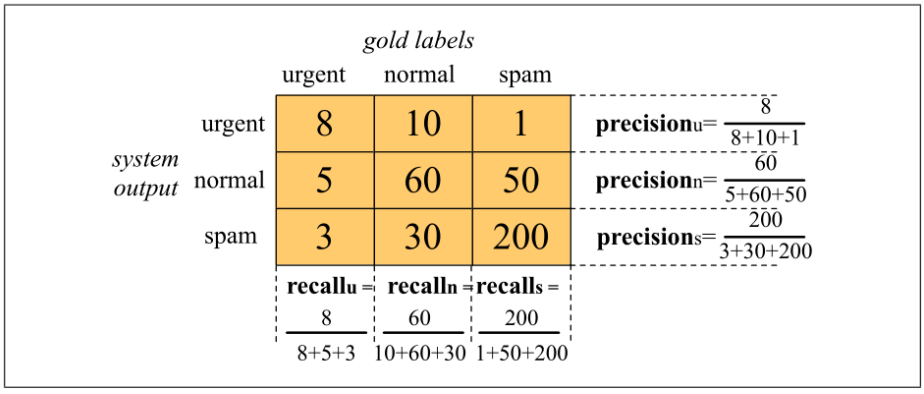


图 9.16: 三类别分类的混淆矩阵

此时我们需要稍微修改一下准确度和召回率的定义。该矩阵显示系统错误地将一个垃圾邮件文档标记为紧急，我们还展示了如何为每个类计算不同的精度和召回值。为了得出一个能评估系统运行情况的指标，我们可以用两种方式组合这些值。在宏观平均中，我们计算每个类的性能，然后对每个类平均。在微平均中，我们将所有类的决策收集到一个混淆矩阵中，然后从该表中计算精度和召回率。下图分别给出了每一类的混淆矩阵，并给出了微平均精度和宏平均精度的计算。

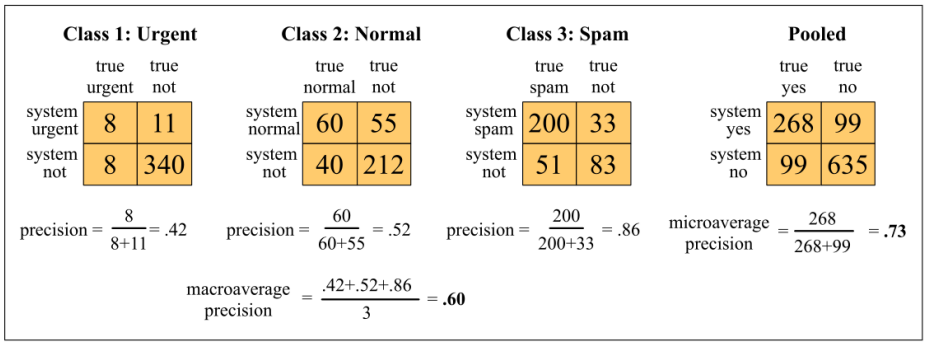


图 9.17: 分类混淆矩阵

如图所示，微平均值将由更频繁的类 (在本例中是垃圾邮件) 主导，因为这些计数被汇集在一起。宏平均更好地反映了较小类的统计数据，因此当所有类的性能同样重要时，宏观平均更合适。

9.3 深度学习下的自然语言处理

9.3.1 词向量

词是自然语言中能够独立运用的最小单位，是语言信息处理的基本单位。将自然语言转换为机器能理解的符号是语言模型的关键。在深度学习被应用于自然语言处理之前，传统的词表达采用的是 one-hot encoding 的方式。在自然语言处理中，若字典中或者语料库中有 N 个词，则每一个词可以被一个 N 维的向量所表示。并且在这个向量中，只有某一个维度上的数值为 1，其余 N-1 的维度上的数值皆为 0。这种表示就称为 one-hot 编码方式。通过 one-hot 编码方式，可以将不同的词转换成电脑可以理解的数字，并且由于这种编码方式将词语转换成具有固定的维度的向量，这使得机器学习演算时可以进行线性代数运算。

例如：如果我们的语料库中现在有三个省份，山东，河南，以及吉林。对于着三个省份，它们对应的 one-hot 向量可以为每一个省份对应的 one-hot 向量中，只有某一个维度上的数值为 1，其余维度上数值为 0。

但是，尽管 one-hot 编码方式成功的将字符转换为了数字，这种编码方式存在两个明显的缺点。第一，维数灾难。第二，相互独立。由于 one-hot 向量的维度取决于语料库中词汇的数量，当词汇表中每增加一个词，向量的维度也要增加一。在我们刚才的例子中，如果我们把语料库扩充到中国的全部省份，由于中国共有 34 个省级行政区，那么此时也需要把 one-hot 向量维度增加到 34 维。如果我们再想细化到市级行政区，那么向量的维数可能要增加到上百维。而实际上，如果想用 one-hot 编码来表示一个真实语料库中的词汇，需要几万甚至几十万维度，这不仅在运算时会带来维度灾难的问题，对于计算机内存也是难以支撑的。

另一个问题是 one-hot 向量之间相互独立，无法刻画相似词汇之间的关系。同样在之前的例子中，当我们计算两个向量的距离时，例如如果我们想计算山东与河南、山东与吉林余弦距离，我们发现两者的余弦距离都等于零，俩俩之间相互正交。但是我们都知道，由于山东与河南接壤，相对于吉林，山东省与河南省有更多的相似性。而在自然语言中，词与词之间的也具有多种

多样的关系，反义词、同义词、正向情感词、负向情感词等，如果我们的编码方式无法刻画出词汇之间的内部关系，那么就无法准确的理解自然语言。

为了解决这两个问题，2013 年 Word2Vec 模型的发表使词向量（Word Embedding）这一概念正式进入公众视野。这标志着自然语言处理进入了全新的阶段，引起了自然语言处理井喷式地发展。

词向量，也叫做词嵌入。它是指把一个维数为所有词的数量的高维空间嵌入到一个维数低得多的连续向量空间中，每个单词或词组被映射为实数域上的向量。举个例子，如果我们将 34 个省级行政区从 34 维的向量空间嵌入到 3 维的向量空间，那么此时代表山东省、河南省和吉林省的词向量可能分别变为：

$$V_{SD} = [0.7, 0.7, 0.3]$$

$$V_{HN} = [0.6, 0.7, 0.2]$$

$$V_{JL} = [0.9, 0.9, 0.7]$$

此时，山东与河南向量之间的距离为：

$$D_{V_{SD}V_{HN}} = \sqrt{(0.7-0.6)^2 + (0.7-0.7)^2 + (0.3-0.2)^2} \approx 0.14$$

$$D_{V_{SD}V_{JL}} = \sqrt{(0.7-0.9)^2 + (0.7-0.9)^2 + (0.3-0.7)^2} \approx 0.49$$

山东与河南的距离此时小于山东与吉林的距离。显然与通过词嵌入表达之后，向量之间的关系不再相互独立，它们之间的相似度也可以进行度量。如果我们再想添加别的省份，此时我们不需要再添加一个维度，而是可以直接利用模型获得其三位词向量的表达。这使得我们可以避免维数灾难的问题。

实际上，一个好的词向量空间不仅仅可以降低向量空间的维度，向量与向量之间在空间中差距也具有一定的意义。2014 年在斯坦福大学自然语言处理组发表的 GloVe 模型中，词向量在向量空间中的相对位置具有了意义，也使得词向量的加减具有了意义。

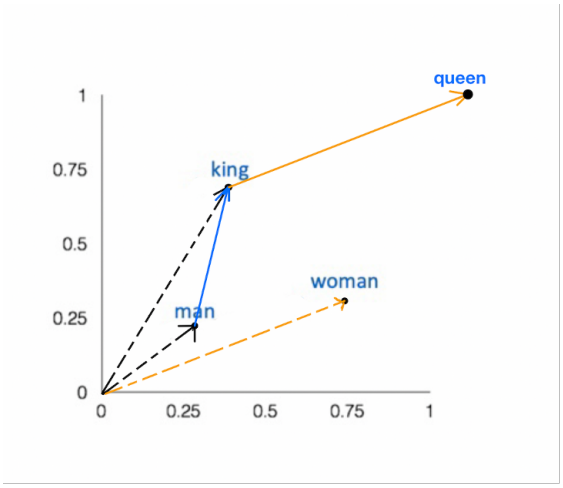


图 9.18: 词向量

在上图中，我们可以看到，queen=king-man+woman。图中蓝色的向量表示了 king 与 man 之间的差距，也就是说我们可以把蓝色向量理解成在向量空间中一种王权的表示。那么国王就是

由男人和王权所构成的，当我们将国王减去男人这个属性后，此时只剩下王权，但是在加上女性后，我们得到带有王权的女性，于是我们就得到了王后 queen。

实际上 GloVe 模型所得到的词向量空间维数有 50 维、100 维、200 维以及 300 维。为了可视化的方便，我们可以将其降维至 2 维，以观察其不同方向上的向量含义，例如下图中所示：

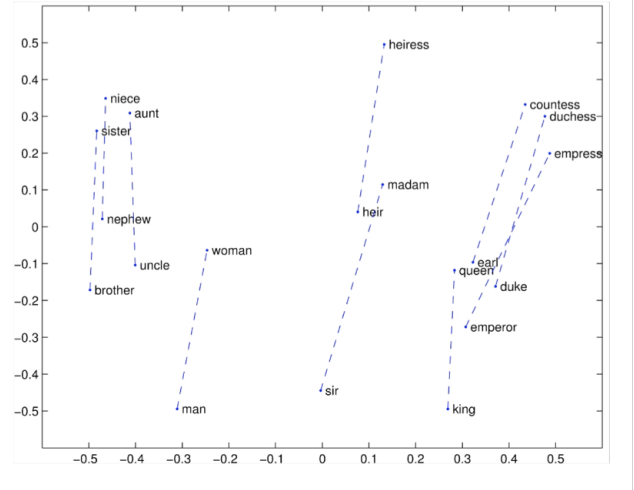


图 9.19: 词向量（性别）

在上图中，我们可以看到，从左下到右上这个方向的向量代表了性别的变化，当代表男性的词加上这个向量之后，就会转换成相对应地女性词汇。

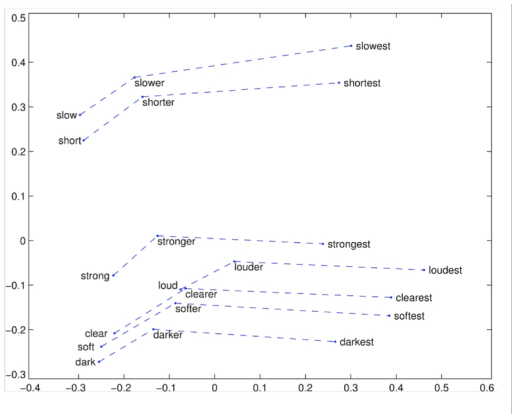


图 9.20: 词向量（形容词级别）

同样，上图中我们可以看到一组形容词从其原形到比较级，最后到最高级的变化趋势。下图是通过 GloVe 模型获得关于“banana”单词 100 维的词向量表示：

```
glove_100d.get_vector('banana')
```

```
array([-0.34028  ,  0.46436  , -0.083324 ,  0.20186  , -0.17831  ,
       -0.4663   ,  0.61793  ,  0.30129  ,  0.5728   , -0.34783  ,
       -0.9216   ,  0.30484  ,  0.30382  ,  0.58035  ,  0.12112  ,
        0.77288  ,  1.1547   , -0.576    ,  0.51471  ,  0.21552  ,
        0.21106  ,  0.67875  ,  1.1962   ,  0.11142  ,  0.50809  ,
        1.1873   ,  0.035288 , -0.88952  ,  0.042803 , -0.36714  ,
        0.37993  ,  0.61945  ,  1.0194   , -0.95084  , -0.0072258,
        0.69454  ,  0.38692  , -0.18544  ,  0.2885   , -0.81279  ,
       -0.46473  , -0.82623  ,  0.42778  , -0.14064  ,  0.30173  ,
        0.074418 , -0.40044  ,  0.33969  , -0.62917  , -0.054449 ,
       -0.78469  ,  0.2354   , -0.78359  ,  0.74708  , -0.31074  ,
       -0.07038  , -0.34623  ,  0.33849  ,  0.89621  ,  0.30288  ,
        0.012978 ,  0.020869 , -0.14436  , -0.40914  ,  0.16651  ,
       -0.88124  , -0.078419 ,  0.048156 ,  0.27032  , -0.81761  ,
        0.027778 ,  0.62487  ,  0.1549   , -0.15838  ,  0.088675 ,
        0.063411 , -0.14473  , -0.0066816, -0.18535  ,  1.5642   ,
        0.3726   , -0.81706  , -0.021685 ,  0.91209  , -0.35784  ,
       -0.98389  , -0.37103  , -0.10909  ,  0.18898  , -0.33884  ,
       -0.058326 ,  0.41438  , -1.0411   , -0.42643  , -0.50664  ,
       -0.75863  , -0.15815  , -0.1831   ,  0.7343   , -0.26852  ],
      dtype=float32)
```

图 9.21: “banana” 词向量（100 维）

GloVe 还允许我们找到与某一个词最相似的其他词，同样还是以 “banana” 为例，下面代码显示了 GloVe 模型中与其最相似的 10 个词，分别是 coconut，mango，bananas 等。

```
pprint(glove_100d.most_similar('banana'))
```

```
[('coconut', 0.7097253799438477),
 ('mango', 0.7054824233055115),
 ('bananas', 0.6887733936309814),
 ('potato', 0.6629636287689209),
 ('pineapple', 0.6534532904624939),
 ('fruit', 0.6519855260848999),
 ('peanut', 0.6420576572418213),
 ('pecan', 0.6349173188209534),
 ('cashew', 0.6294420957565308),
 ('papaya', 0.6246591210365295)]
```

图 9.22: “banana” 最相似词向量

词向量的发明不仅将语言转换成了机器能读懂的数字，并且还将这些数字赋予了意义，这对机器对自然语言的理解产生了重大的意义。实际上，近几年词向量表达的模型得到了质的飞跃。2018 年，由 google 团队开发的 BERT 模型，利用 transformer 神经网络结构在 33 亿语料上进行训练，获得了强大的词向量表达。该模型在阅读理解测试上的表现已经超越了人类水平。2020 年，Open AI 团队发表的 GPT-3 语言模型，甚至已经可以写出连贯富有激情的文章。总而言之，词向量是当前一切深度学习模型的基础，关于如何更近一步提升词向量的质量，也仍然是学术界研究的重点之一。

9.3.2 句向量嵌入模型

在得到词向量之后，自然想到如何获得句子向量。实际上获得通过词向量获得句子向量的模型有很多，下面举几个例子，简要看一下这些模型是如何获得句子向量的。

1、Bag-of-Words（Kalchbrenner et al.,2014）

该模型将一个句子中的 n 个词的词向量对应维数上的数值直接相加起来，获得一个句子的词向量。

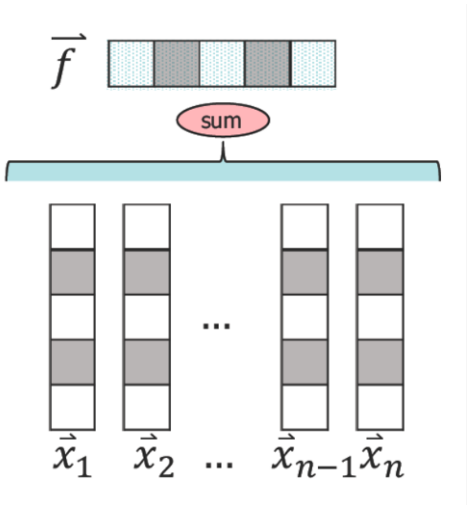


图 9.23: 逐元素加和句向量

其优点是不需要额外的模型训练，得到的句子是固定维数的，句子向量的长度等于词向量的长度。但它的缺点是丢失了句子中词汇的顺序信息，而词序在句子中起着重要的作用，不同的词序可能导致句子意思截然不同。此外，句子的表达也不会是简单地词语的堆叠，所以通过这种简单相加方式直接得来的句子向量表达是不准确的。

2、Pooling (Tang et.al. ,2014; Vo and Zhang,2015)

相较于前一种方法，这种句向量的表达由三部分构成，首先选出 n 个词中最大的词向量和最小的词向量，然后再算出 n 组词向量的平均值词向量，最后将这三个向量拼接起来作为该句子的向量表示。

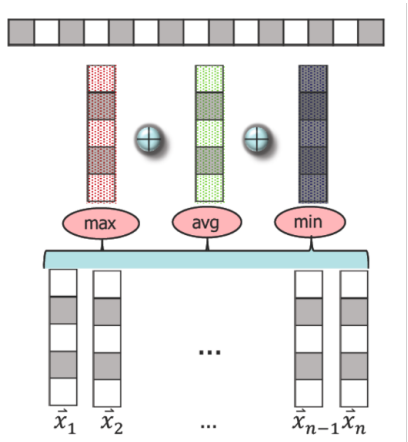


图 9.24: 池化平均句向量

但这两种句向量的构成都过于简单，无法有效捕捉句子蕴含的信息，于是有人提出利用神经网络的方式获得句子向量。

3、CNN(Kim,2014)

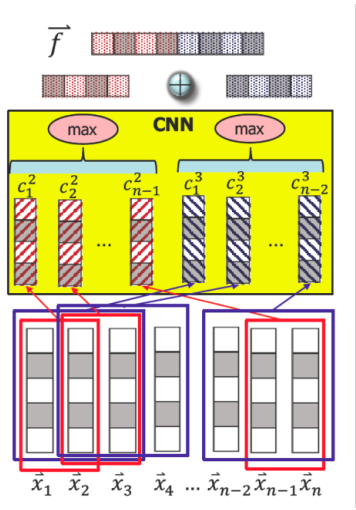


图 9.25: 卷积句向量

在该模型中，卷积神经网络（CNN）被用来学习句子向量。如下图所示，首先我们选择一个 kernel 1（图中为红色框表示），该 kernel 一次滤过两个词向量（window size = 2）提取出特征，并返回一个新的向量。以步长为 1，依次滑过 n 个词向量。对于返回这一系列向量，再用 max pooling 的办法，选出每一个通道上的最大值，构成最终的一个 4 维向量（如图红色行向量所示）。之后，再选择一个新的 kernel 2 （图中蓝色框所示），进行同样的步骤，最后返回另一个 4 维向量（如图灰色行向量所示）。最后将这两个向量拼接起来，获得句子向量表达。

这个模型相较于之前的模型，首先它能学习到词序信息，此外其句子向量的构成不再是单一的加法运算，利用神经网络的非线性性，是的句向量的表达蕴含更丰富的信息。

类似的句子向量模型其实还有很多，但是实际上，以上这些句向量模型，在实际运用中都不再用到。句向量的表达在之后介绍完循环神经网络（RNN）章节后，利用 RNN 强大的表达和记忆能力，会有新的获得方式。

9.3.3 文档嵌入模型

当获得句子向量之后，自然希望获得由一个一个句子所构成的文档向量。其实文档向量的构成和句子向量的构成思路是相似的。在这里举一个例子让大家感受一下如何获得文章的向量表达形式。

1、LSTM/CNN-GRU （Tang et al.,2015）

在该模型中，首先我们获得每一个词的词向量，然后我们将构成一个句子的词向量喂入到一个神经网络当中（CNN 或 LSTM），获得句向量的表达，然后我们再将每一个句子向量，喂入一个双向的带有门的循环神经网络中（如 LSTM/GRU），将前向与后向所学习到的向量进行拼接，最后再将这些拼接起来。

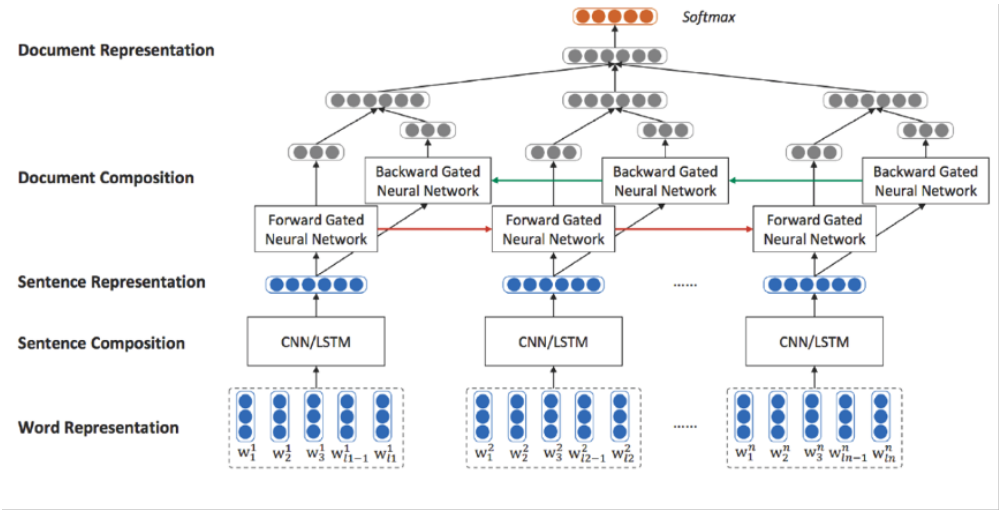


图 9.26: 文档嵌入模型

向量进行线性运算，获得最后关于整篇文章的向量表达形式。说到这里，我们反复提到循环神经网络，在后面的章节中我们将重点介绍这一类模型。

9.3.4 前馈神经网络与自然语言处理

下图为前馈神经网络的结构原理，如图我们其实可以看出循环神经网络存在如下缺陷：

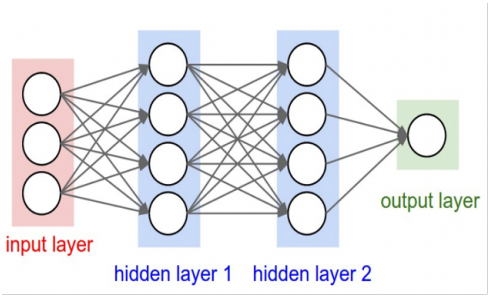


图 9.27: 前馈神经网络

1、输入句子的词汇节点位置无关，无法对语言中的词序进行建模。例如我们的输入文本是“She loves cat.”，这个文本的喂入前馈神经网络的输入词向量就应该分别是 she、loves、cat。但由于节点直接互相没有链接，因此在这个例子当中，机器就将不能识别词汇的顺序，可能将句子理解“Cat loves she.”

2、层与层之间有链接，但节点之间无连接。同样还是上面那个句子，“She loves cat. ”由于 She 是第三人称单数，所以动词 love 将变成 loves 的形式，然而前馈神经网络将无法体现句子中词汇间的关系，这就会导致机器学习的准确率下降。

3、输入和输出的维数是固定的，无法处理变长的序列数据。当我们想分析一片文章的情感时，我们应该将文章中的每一个句子的作为神经网络的输入去进行学习，得到一个关于情感倾向的输出。而前馈神经网络的输入要求维数相同，如下图中，输入层有三个元素，意味着输入有三

个单词。若一个句子包含的词汇数不等于 3，机器的训练由于句子长短不一将失去意义。

9.3.5 简单循环神经网络

循环神经网络 (Recurrent Neural Network，RNN) 是一类具有短期记忆能力的神经网络. 在循环神经网络中，神经元不但可以接受其他神经元的信息，也可以接受自身的信息，形成具有环路的网络结构。和前馈神经网络相比，循环神经网络更加符合生物神经网络的结构. 循环神经网络已经被广泛应用在语音识别、语言模型以及自然语言生成等任务上。循环神经网络的参数学习可以通过随时间反向传播算法 [Werbos, 1990] 来学习。随时间反向传播算法即按照时间的逆序将错误信息一步步地往前传递. 当输入序列比较长时，会存在梯度爆炸和消失问题 [Bengio et al., 1994; Hochreiter et al., 1997, 2001]，也称为长程依赖问题. 为了解决这个问题，人们对循环神经网络进行了很多的改进，其中最有效的改进方式是引入门控机制 (Gating Mechanism)。

简单循环网络 (Simple Recurrent Network，SRN)[Elman, 1990] 是一个非常简单的循环神经网络，只有一个隐藏层的神经网络. 在一个两层的前馈神经网络中，连接只存在于相邻的层与层之间，隐藏层的节点之间是无连接的. 而简单循环网络增加了从隐藏层到隐藏层的反馈连接。

令向量 $x_t \in \mathbb{R}^{D\times D}$ 表示在时刻 t 时网络的输入，$h_t \in \mathbb{R}^{D\times D}$ 表示隐藏层状态 (即隐藏层神经元活性值)，则 h_t 不仅和当前时刻的输入 x_t 相关，也和上一个时刻的隐藏层状态 h_{t-1} 相关. 简单循环网络在时刻 t 的更新公式为:

$$z_t = Uh_{t-1} + Wx_t + b \tag{3.1}$$

$$h_t = f(z_t) \tag{3.2}$$

其中 z_t 为隐藏层的净输入，$U \in \mathbb{R}^{D\times D}$ 为状态-状态权重矩阵，$W \in \mathbb{R}^{D\times M}$ 为状态-输入权重矩阵，$b \in \mathbb{R}^{D}$ 为偏置向量，f 是非线性激活函数，通常为 Logistic 函数或 Tanh 函数. 公式 (3.1) 和公式 (3.2) 也经常直接写为:

$$h_t = f(Uh_{t-1} + Wx_t + b) \tag{3.3}$$

如果我们把每个时刻的状态都看作前馈神经网络的一层，循环神经网络可以看作在时间维度上权值共享的神经网络。

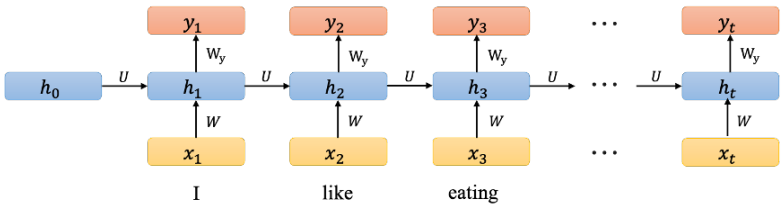


图 9.28: 按时间展开的循环神经网络

上图给出了按时间展开的循环神经网络。基本原理：假设时刻 t 时，输入为 x_t 隐藏层状态为 h_t。h_t 不仅和当前时刻的输入相关，也和上一个时刻的隐层状态相关。

$$h_t = f(Uh_{t-1} + Wx_t + b) \tag{3.4}$$

$$y_t = softmax(W_y h_t) \tag{3.5}$$

其中 f 是非线性函数，通常为 sigmoid 函数或 tan 函数。

9.3.6 长短时记忆神经网络（LSTM）

从前面我们知道 RNN 面临长期依赖问题，即很久以前的输入，对当前时刻的网络影响较小。在反向传播时，梯度也很难影响到很久以前的输入。然而，在自然语言中，时常面临句子中靠后的词与靠前的词拥有依赖关系的情况，例如：

The **cat** , which already ate a bunch of food, **was** full.

The **cats** ,which already ate a bunch of food, **were** full.

在这个例子中，谓语 be 动词的选择就取决于句子最开始的主语是单数还是复数。为了解决这一问题，一个基于循环神经网络的变体——长短时记忆神经网络（Long Short-Term Memory Neural Network，LSTM）被发明以有效的解决长期依赖问题以及梯度消失问题。

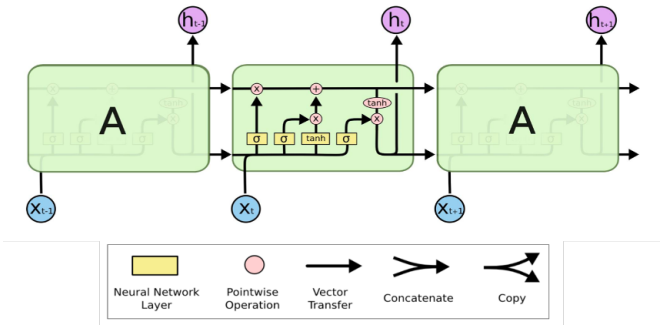


图 9.29: 长短时记忆神经网络

LSTM 模型的关键是引入了一组记忆单元 c_t，以及三个控制“门”。它们允许网络可以学习何时遗忘历史信息，何时用新信息更新记忆单元。在时刻 t 时，记忆单元 c_t 记录了到当前时刻为止的所有历史信息，并受三个“门”控制，三个门中元素的数值取值范围是 $(0, 1)$：

$$Input\ Gate : i_t = \sigma(U_i h_{t-1} + W_i x_t + b_i)$$

$$Forget\ Gate : f_t = \sigma(U_f h_{t-1} + W_f x_t + b_f)$$

$$Output\ Gate : o_t = \sigma(U_o h_{t-1} + W_o x_t + b_o)$$

接下来我们会一步步拆分并说明 LSTM 网络的结构。

记忆细胞状态与“门”机制

首先我们先看 LSTM 的核心机制：记忆细胞状态与“门”机制，及其更新方式。

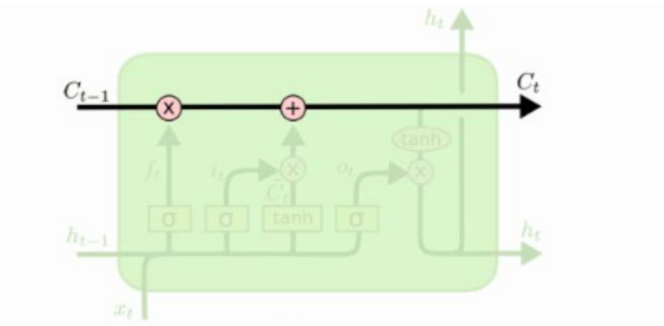


图 9.30: 记忆细胞

如上图所示，记忆细胞 c_t 在整条链上传递，这条链上只有一些小的线性操作，使信息保持不变的流过整条链。记忆细胞 c_t 中包含到当前时刻 t 之前所有有效的历史信息，这些信息将会随着网络一直传递下去，保证了在学习当前时刻输入信息的同时，不会遗忘掉之前学习到的信息。

门（Gate）是一种让信息有选择地通过的方式。它由一个 Sigmoid 函数和一个点乘运算组成。

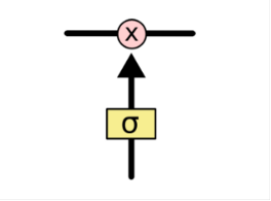


图 9.31: 门

Sigmoid 函数返回 0 到 1 一个之间的数字，这个数字描述每个元素有多少信息可以通过，0 表示不通过任何信息，1 表示全部通过。

门机制控制了对于当前时刻的信息的多少将被保留，多少将被遗忘。也将控制新信息的多少将被加入记忆信息，多少信息又将从记忆信息中被提取出作为当前时刻的输出。LSTM 中总共有三个门：遗忘门、输入门以及输出门。

遗忘门（forget gate）

首先我们先看遗忘门 f_t 的更新机制：

$$f_t = \sigma(U_f h_{t-1} + W_f x_t + b_f)$$

其中，U_f 是关于前一个隐藏状态 h_{t-1} 的参数矩阵，W_f 是关于当前输入 x_t 的参数矩阵 b_f 是一个偏置向量。由于 sigmoid 函数的存在，f_t 的取值范围在 $(0, 1)$ 之间。

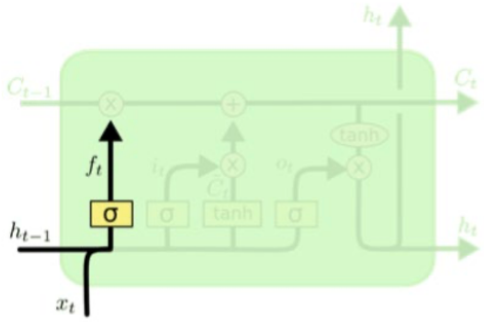


图 9.32: 遗忘门

遗忘门 f_t: 根据 h_{t-1}（前一个隐藏状态）和 x_t（当前输入），为记忆状态 C_{t-1}（t-1 时刻状态）中的每个元素输出 0 ~ 1 之间的数字，以决定每一个 C_{t-1} 元素保留多少信息。1 代表完全保留，0 代表彻底删除。

输入门（input gate）

输入门 i_t: 决定我们要在细胞状态中存储多少当前的新信息。

$$i_t = \sigma(U_i h_{t-1} + W_i x_t + b_i)$$

与遗忘门相同，U_i 是关于前一个隐藏状态 h_{t-1} 的参数矩阵，W_i 是关于当前输入 x_t 的参数矩阵，b_i 是一个偏置向量。同理，输入门 i_t 的取值范围在 $(0, 1)$ 之间。它决定对于当前输入的信息 $\tilde{C}_t$ 保留多少信息。那么 $\tilde{C}_t$ 如何获得的呢？下面式子给出了 $\tilde{C}_i$ 的计算方式：

$$\tilde{C}_t = tanh(U_c h_{t-1} + W_c x_t + b_c)$$

类似遗忘门与输入门，我们同样利用上一个时刻隐藏层向量 h_{t-1} 与当前输入向量 x_t 分别乘上参数矩阵 U_c 与 W_c, 再加上偏置 b_i 首先进行一个线性变化，不同的是由于 $\tilde{C}_t$ 不是一个门，我们不需要其取值范围限定在 $(0, 1)$ 之间，于是选择用 tanh() 函数代替 sigmoid 函数进行非线性激活。

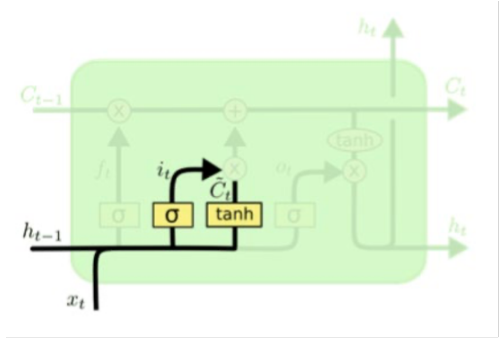


图 9.33: 输入门

更新记忆细胞状态

在有了遗忘门 f_t，输入门 i_t, 以及当前输入信息 $\tilde{C}_t$，便可以更新记忆细胞状态 C_t。

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

通过遗忘门 f_t，将计算出历史记忆信息中要被保留的部分即 $C_{t-1} * f_t$。通过输入门 i_t，计算出要被添加进历史记忆信息的部分，即 $i_t * \tilde{C}_t$，通过将这两者相加，获得到一个当前时刻的历史记忆信息 C_t。

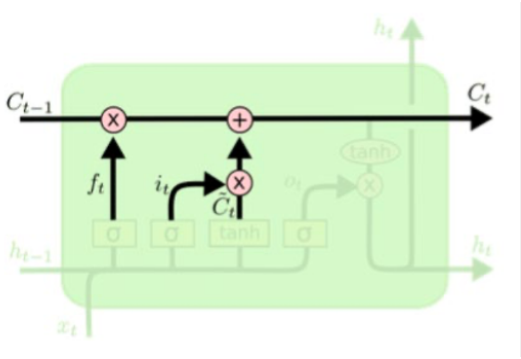


图 9.34: 记忆细胞状态更新

输出门（output gate）

最后我们来看输出门 o_t：决定基于当前的 c_t，我们要输出什么作为当前时刻隐藏层的信息 h_t。o_t 的更新方法依旧类似于之前的遗忘门与输出门：

$$o_t = \sigma(U_o h_{t-1} + W_o x_t + b_o)$$

其中，U_o 是关于前一个隐藏状态 h_{t-1} 的参数矩阵，W_0 是关于当前输入，o_t 的参数矩阵，b_0 是一个偏置向量。同理，由于 sigmoid 函数，输入门 o_tt 的取值范围也在 $(0, 1)$ 之间。在获得 o_t 之后，利用之前更新好的当前时刻历史记忆信息 C_t，进行当前时刻隐藏层 h_t 信息的更新:

$$h_t = o_t * \tanh(C_t)$$

在更新 h_t 的时候，我们首先用非线性函数 tanh() 对 C_t 进行激活，然后在与 o_t 进行点乘运算，保证向量维度保持不变。h_t 会作为下一个时刻的输入之一，在参与未来新的一系列更新。

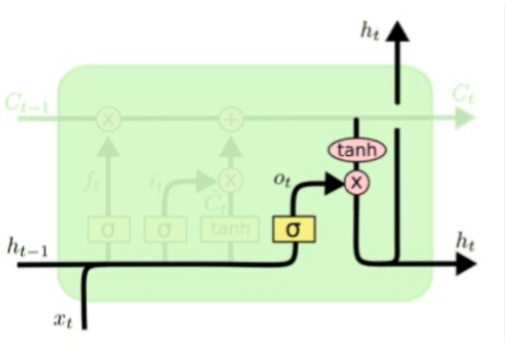


图 9.35: 输出门

LSTM 结构总结

在回顾 LSTM 的更新结构之前，有两点需要强调的是:

1、尽管三个门，以及当前输入信息 $\tilde{C}_t$ 的更新都具有相同的形式，但是每各更新所用的参数都是独立不同的，这保证了反向传播时对三个门控单元即当前记忆信息独立地进行更新。

2、以上公式中的所有参数都是可学习的参数。它们在最初是自动随机生成的随机数，但是随着模型的不断训练，这些参数会随着反向传播算法不断的进行更新，最终使得损失函数数值尽可能的小以提高模型的表达。

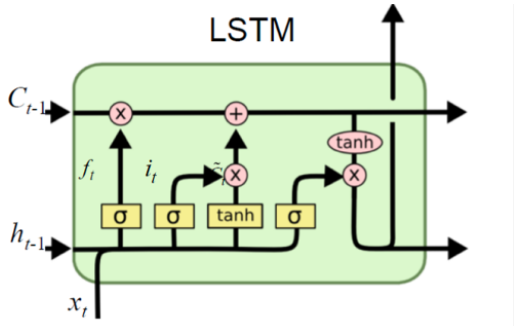


图 9.36: LSTM 结构

LSTM 结构回顾:

(1) LSTM 有单独的细胞状态，该状态贯穿整个 LSTM 网络;

(2) 用遗忘门 $\tilde{C}_t$ 和输入门 i_t 决定历史记忆信息与新记忆信息的保留或放弃

(3) 当前输入信息 $\tilde{C}_t$ 来源于 h_{t-1} 和 x_t;

(4) 当前记忆信息由 $f_t * C_{t-1} + i_t * \tilde{C}_t$ 计算得出;

(5) 输出门控制细胞状态的输出 $h_t = o_t * tanh(C_t)$

9.3.7 门限循环单元 (GRU)

由于 LSTM 网络结构比较复杂，有学者提出了一种新的 RNN 模型变体，门限循环单元 (GRU)。GRU 是一种比 LSTM 更加简化的模型。在 LSTM 中，输入门和遗忘门是互补关系，

因为同时用两个门比较冗余。GRU 将输入门和遗忘门合并成一个门：更新门（Update Gate），同时还合并了记忆单元和隐藏神经元。

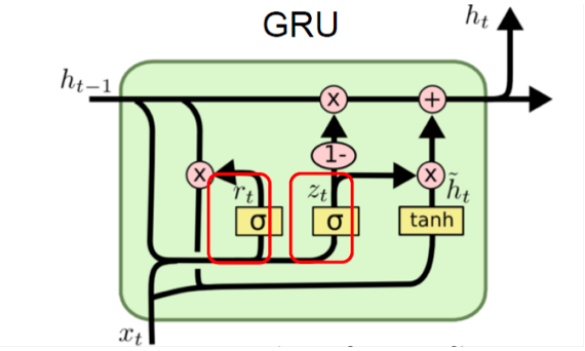


图 9.37: 门限循环单元

- 重置门 $r_t = \sigma(U_r h_{t-1} + W_r x_t + b_r)$
- 新信息 $\tilde{h_t} = tanh(U_h(r_h * h_{t-1}) + W_h x_t + b_i)$
- 更新门 $z_t = \sigma(U_z h_{t-1} + W_z x_t + b_z)$
- 隐藏状态 $h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h_t}$

GRU 将输入门和遗忘门合并成一个更新门。同时还合并了记忆单元和隐藏神经元。GRU 的重置门 r_t 用来控制上一时刻隐藏状态 h_{t-1} 的哪些部分将会被用来计算新信息 $\tilde{h_t}$。GRU 的更新门 z_t 控制上一时刻隐藏状态 h_{t-1} 的哪些部分将被遗忘，以及 $\tilde{h_t}$ 中的哪些信息将被保留，类似于 LSTM 模型中遗忘门和输入门的合并，用于更新当前时刻的隐藏层状态 h_t。

总结：GRU 没有单独的细胞状态，用更新门决定保留或放弃，$\tilde{h_t}$ 由重置门决定 $\tilde{h_{t-1}}$ 的信息，并且直接输出隐藏状态。

9.3.8 循环神经网络的其他变体

双向循环神经网络

在之前的循环神经网络中，无论是 RNN，LSTM，还是 GRU，我们注意到模型的输入都是同一个方向，即从左向右的将句子里的每一个词一次输入到模型中。新的词语的隐藏层信息的表达，会受之前词汇的影响，但是在自然语言中，位于句子之后的词往往也会影响于句子前端的词汇。例如：

我今天不舒服，我打算... 一天

只根据“不舒服”可能推算出我打算“去医院”，“睡觉”，“请假”等等。但如果加上后面的“一天”，选择范围就将变小，“去医院”被选中概率将减小，而“请假”、“休息”之类的被选概率将会增大。

为了解决这种问题，我们除了将词语正序地传入循环神经网络的同时，我们也将同时倒序地将词语传入循环神经网络中，这就形成了双向循环神经网络。

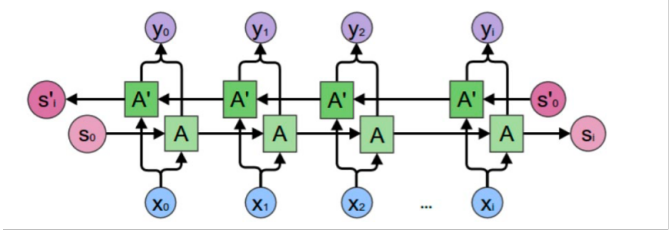


图 9.38: 双向循环神经网络

在 Forward 层从 0 时刻到 i 时刻正向计算一遍，得到并保存每个时刻向前隐含层的输出。

在 Backward 层从 i 时刻到 0 时刻反向计算一遍，得到并保存每个时刻向后隐含层的输出。

最后在每个时刻结合 Forward 层和 Backward 层的相应时刻输出的结果得到最终输出。

堆叠循环神经网络

有时我们可能觉得循环神经网络纵向上的深度不够，可能导致无法具有足够的表达性来捕获句子信息，于是会在第一层循环神经网络的输出上再添加一层循环神经网络。第二层循环神经网络的输入，是第一层循环神经网络的输出。而整个模型的最终输出取自第二层模型的输出。在实际运用中，循环神经网络的网络的层数没有特别的限制。层数越多，模型的表达能力越强，可能获得更好的效果，但是相应的模型也会越复杂，导致训练时间较长，甚至难以训练。

9.3.9 循环神经网络的输入与输出

循环神经网络在 NLP 中应用的各种形式多种多样，如一对多，多对一，多对多等。

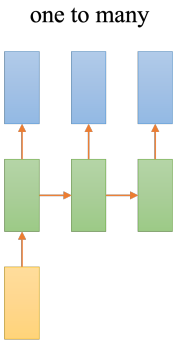


图 9.39: 一对多结构

1、一对多（One to many）

如图所示，输入元素为一，输出元素有多个的即为“一对多（One to Many）”的输出形式。这种形式主要是针对我们想让机器能通过对一张图片进行训练和识别，反馈一段对于该图片的相关文字描述。

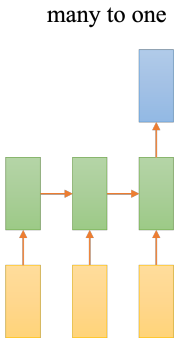


图 9.40: 多对一结构

2、多对一（many to one）

如图所示。其输入为多个元素，输出元素只有一个。这种输入输出方式即为“多对一（many to one）”。其主要应用于文本分类或情感分析中。就文本分类而言，我们将多篇文章做为输入喂入神经网络，通过深度学习，机器将反馈给我们关于其文章所属类别的输出。情感分析也是同样的道理，反馈结果将是对该文章情感倾向的输出。

3、多对多（many to many）

如图所示。其输入为多个元素，输出元素也有多个。这种输入输出方式即为“多对多（many to many）”。“多对多”的形式还包含两种类型。一种是如图（左）所示，这是一种“编码—解码（Encoder-decoder）”结构。通过对输入元素进行打包编码进行信息汇总，再通过解码端进行解码，得到对应的输出。这种形式主要应用于自动文摘，文本翻译等方向。另一种类型是如图（右）所示。这种形式是每一个输入都对应一个输出，主要应用于词性标注，命名体识别等领域。

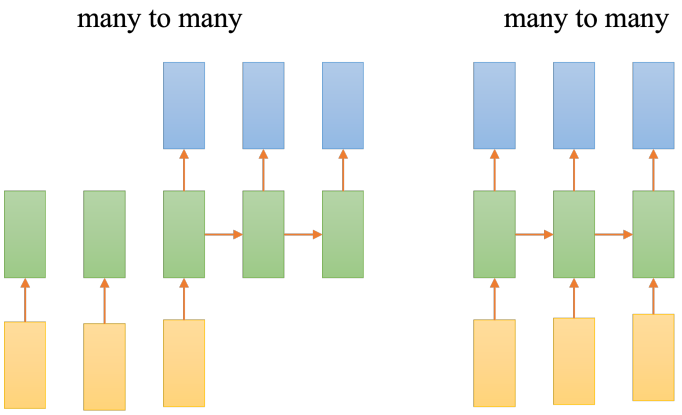


图 9.41: 多对多结构

9.3.10 循环神经网络的应用

1、对话系统

Encoder 端：对话中上文（问句），例如 How are you?

Decoder 端：对话中的下句（回复句），例如 I am fine.

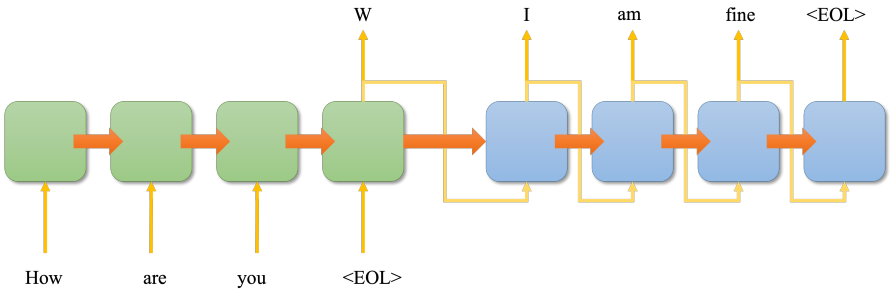


图 9.42: Encoder-Decoder 结构

2、情感分析

输入为序列，输出为类别。比如在文本分类中，输入数据为单词序列，输出为该文本或句子的类别。

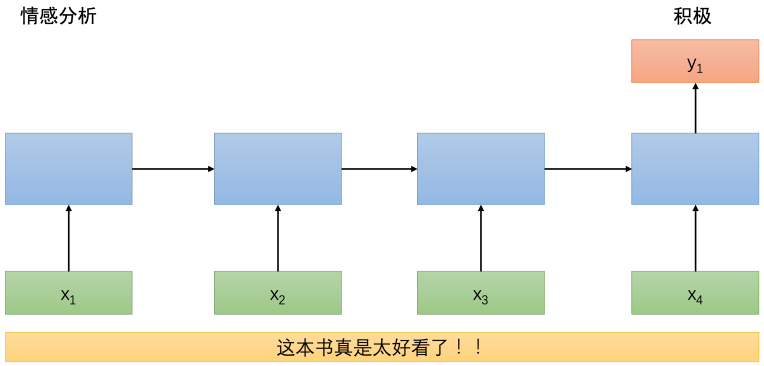


图 9.43: 情感分析模型

3、自动作曲

输入的是前几个旋律，输出的是一首完整的曲子。哈杰里斯和帕切特开发了一个名为“DeepBach”（深度巴赫）的神经网络。经过训练，DeepBach 能够创作出与巴赫风格高度相近的作品，几乎到了“以假乱真”的地步。

除此之外，循环神经网络在自然语言处理中的应用还包括文本分类（Text Classification, supervised）、文本聚类（Text Clustering, unsupervised）、信息抽取、自动文摘（自动将文本转换成简短摘要的一种信息压缩技术）、信息推荐（根据用户习惯，偏好，兴趣等提供个性化的服务）、信息推断等诸多领域的应用。当前最新的注意力机制（Attention）技术也使自然语言处理技术发展到了一个新的阶段。

9.4 代码实例

在本小节中，我们将以 IMDB 数据集为例展示循环神经网络在自然语言处理中的应用。

IMDB 数据集包含了 50000 条偏向明显的评论，其中 25000 条作为训练集，25000 作为测试集。label 为 pos(positive) 和 neg(negative), 属于文本分类问题中的情感分析任务。

详细构骤如下：

(1) 导入 tensorflow 后端并设置绘图辅助函数。

```
import tensorflow_datasets as tfds
import tensorflow as tf

##导入 matplotlib 并创建一个辅助函数来绘制计算图:
import matplotlib.pyplot as plt
def plot_graphs(history, metric):
    plt.plot(history.history[metric])
    plt.plot(history.history['val_'+metric], '')
    plt.xlabel("Epochs")
    plt.ylabel(metric)
    plt.legend([metric, 'val_'+metric])
    plt.show()
```

(2) 下载 IMDB 数据集，加载编码器，并进行训练数据预处理。

```
##使用 TFDS 下载数据集
dataset, info = tfds.load('imdb_reviews/subwords8k', with_info=True,
                          as_supervised=True)
train_dataset, test_dataset = dataset['train'], dataset['test']
```

将文本喂入 RNN 前，我们需要对文本进行编码，这里使用的是数据集 info 中的编码器。

```
encoder = info.features['text'].encoder
print('Vocabulary size: {}'.format(encoder.vocab_size))
##输出结果
Vocabulary size: 8185
```

此文本编码器将以可逆方式对任何字符串进行编码，即需要时可以将编码退回至文本。

```
sample_string = 'Hello TensorFlow.'
encoded_string = encoder.encode(sample_string)
print('Encoded string is {}'.format(encoded_string))
original_string = encoder.decode(encoded_string)
print('The original string: "{}"'.format(original_string))
##输出结果
Encoded string is [4025, 222, 6307, 2327, 4043, 2120, 7975]
The original string: "Hello TensorFlow."
```

```
assert original_string == sample_string
for index in encoded_string:
print('{} ----&gt; {}'.format(index, encoder.decode([index])))
##输出结果
4025 ----&gt; Hell
222 ----&gt; o
6307 ----&gt; Ten
2327 ----&gt; sor
```

```
4043 ----&gt; Fl
2120 ----&gt; ow
7975 ----&gt; .
```

接下来对编码后的文本设置批次，并利用 padded_batch 指令将空序列填充至批次中最长文本的长度：

```
BUFFER_SIZE = 10000
BATCH_SIZE = 64
train_dataset = train_dataset.shuffle(BUFFER_SIZE)
train_dataset = train_dataset.padded_batch(BATCH_SIZE)
test_dataset = test_dataset.padded_batch(BATCH_SIZE)
```

(3) 构建 LSTM 模型

```
model = tf.keras.Sequential([
tf.keras.layers.Embedding(encoder.vocab_size, 64),
tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64)),
tf.keras.layers.Dense(64, activation='relu'),
tf.keras.layers.Dense(1)
])
#编译 Keras 模型以配置训练过程：
model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
          optimizer=tf.keras.optimizers.Adam(1e-4),
          metrics=['accuracy'])
```

(4) 训练模型

```
history = model.fit(train_dataset, epochs=10,
                validation_data=test_dataset,
                validation_steps=30)
##部分输出如下
Epoch 1/10
391/391 [==============================] - 53s 123ms/step - loss: 0.6715 - accuracy: 0.5157 -
                                            val_loss: 0.5733 - val_accuracy: 0.7245
Epoch 2/10
391/391 [==============================] - 50s 125ms/step - loss: 0.3973 - accuracy: 0.8309 -
                                            val_loss: 0.3611 - val_accuracy: 0.8562
Epoch 3/10
391/391 [==============================] - 50s 126ms/step - loss: 0.2736 - accuracy: 0.8927 -
                                            val_loss: 0.3334 - val_accuracy: 0.8516
Epoch 4/10
391/391 [==============================] - 50s 126ms/step - loss: 0.2226 - accuracy: 0.9161 -
                                            val_loss: 0.3484 - val_accuracy: 0.8760
Epoch 5/10
391/391 [==============================] - 49s 125ms/step - loss: 0.1959 - accuracy: 0.9285 -
                                            val_loss: 0.3301 - val_accuracy: 0.8687
Epoch 6/10
391/391 [==============================] - 50s 127ms/step - loss: 0.1708 - accuracy: 0.9377 -
                                            val_loss: 0.3414 - val_accuracy: 0.8771
```

```
test_loss, test_acc = model.evaluate(test_dataset)

print('Test Loss: {}'.format(test_loss))
```

```
print('Test Accuracy: {}'.format(test_acc))
##输出结果
391/391 [==============================] - 20s 52ms/step - loss: 0.4349 - accuracy: 0.8540
Test Loss: 0.43485820293426514
Test Accuracy: 0.8539999723434448
```

```
plot_graphs(history, 'accuracy')
plot_graphs(history, 'loss')
```

输出结果如下：

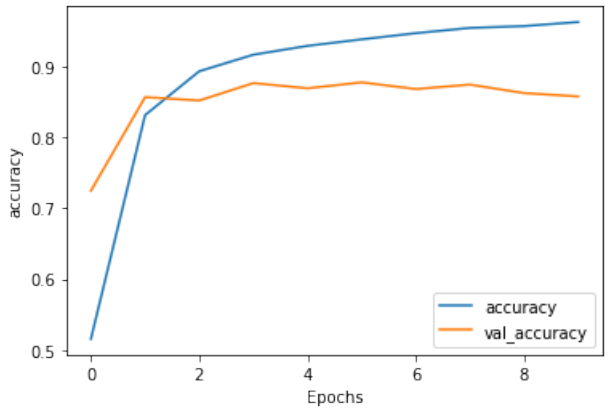


图 9.44: Accuracy

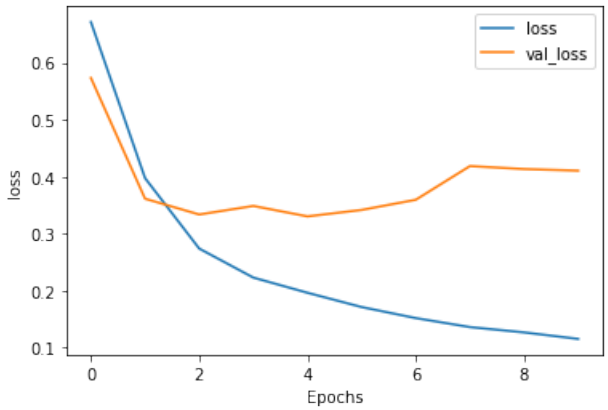


图 9.45: Loss

第 10 章　保形预测

10.1　介绍

在机器学习中，我们不仅需要做出预测，还需要量化预测的不确定性。例如，考虑一个股票自动交易系统，其中人工智能系统预测股票价格。由于股票市场的高度随机性，人工智能的点预测可能与实际值有很大不同。但是，另一方面，如果人工智能系统可以估计出保证覆盖真实价值的高概率范围，交易系统就可以计算出最好和最差的奖励，并做出更明智的决定。所以，给定一个预测的可信范围是很重要的。

10.1.1　定义

保形预测（Conformal Prediction）是一种量化机器学习中此类不确定性的技术。当我们在处理回归或分类问题时，给定输入，保形预测可以输出一个预测区间或者一个预测集，如图10.1所示：

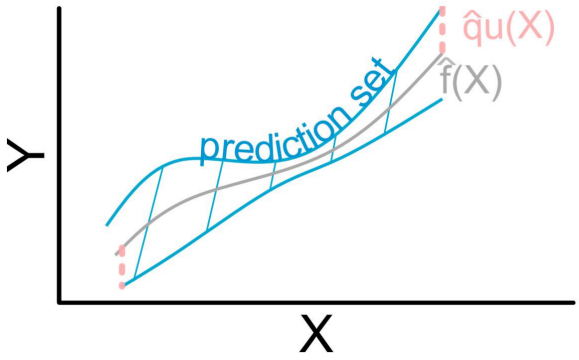


图 10.1: 保形预测预测集

图10.2 中我们展示了类狐松鼠和保形预测产生的预测集的三个渐进困难的例子。这些预测区间和集合都可以保证以较高的概率覆盖真实值。

在进行预测的时候，也许我们会思考这样的问题：我们的预测有多好？如果我们对一个新对象的标签 y 进行预测，我们有多少信心认为我们的预测结果 $\hat{y} = y$？如果标签 y 是一个数字，我们得到的 $\hat{y}$ 与标签 y 有多接近？在机器学习中，这些问题通常以过去的经验相当粗略地回答。

在 Glenn Shafer 和 Vladimir Vovk 著作的 A Tutorial on Conformal Prediction 中，他们是这样定义保形预测的：保形预测使用过去的经验来确定准确的预测置信度。给定一种预测的方

图 10.2: 网络图片上的预测集示例

法，保形预测产生一个 95% 的预测区间，即包含 y 的概率至少为 95% 的集合 $\Gamma^{0.05}$。通常 $\Gamma^{0.05}$ 也包含预测值 $\hat{y}$。我们称 $\hat{y}$ 为点预测，称 $\Gamma^{0.05}$ 为区间预测。在分类的情况下，标签 y 有有限个可能值，$\Gamma^{0.05}$ 可能由这些值中的几个值组成，或者在理想的情况下 $\Gamma^{0.05}$ 只是这些值中的一个值。在回归的情况下，y 是一个数字，$\Gamma^{0.05}$ 通常是附近的一个区间。

10.1.2 简单运用

在介绍保形预测的流程之前，我们要先知道两个定义：一个是随机变量序列的可交换性，另一个是不符合度量。

可交换性 (Exchangeability)

一个有限的随机变量序列是可交换的 (exchangeable)，是指随机变量的联合概率分布对随机变量的排列不变。

$$\mathbb{P}(x_1, x_2, \ldots, x_N) = \mathbb{P}\left(x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(N)}\right)$$

这里 $\pi(1), \pi(2), \ldots, \pi(N)$ 代表自然数 $1, 2, \ldots, N$ 的任意一个排列。一个无限的随机变量序列是无限可交换 (infinitely exchangeable) 的，是指它的任意一个有限子序列都是可交换的。

如果一个随机变量序列 $X_1, X_2, \ldots, X_N, \ldots$ 是独立同分布的，那么它们是无限可交换的。反之不然。

不符合度量 (Nonconformity measure)

保形预测的起点是不符合度量，它度量一个新例子与旧例子的不同程度，也可以说是度量预测值与真实值之间的不符合程度。因此，我们需要寻找一种方法来度量预测值与真实值之间的距离。

保形预测要求我们首先选择一个不符合度度量，而度量预测值和真实值之间距离的方法与数据类型和预测器的选择相关：在回归问题中，最常用的不符合度量是残差的绝对值：$R_i = |y_i - \hat{y}_i|$。在分类问题中，最常用的不符合度量是：$s_i = 1 - \hat{y}_i$ 。

给定一个不符合测度，保形算法对每一个误覆盖水平 α 产生一个预测区域 $\hat{C}_n$。$\hat{C}_n$ 为 $(1-\alpha)$ 预测区间; 它有至少为 $(1-\alpha)$ 的概率包含真实值 y。不同 α 得到的预测区间是嵌套的：

当 $\alpha_1 \geq \alpha_2$ 时，$(1-\alpha_1)$ 的置信度低于 $(1-\alpha_2)$，我们有 $\hat{C}_{n,\alpha_1} \subseteq \hat{C}_{n,\alpha_2}$。

在明确了以上两个概念之后，我们介绍运用保形预测来构造有效的预测集的几个基本步骤。假设我们有一个独立同分布的数据集：

$$\{(X_1, Y_1), \ldots, (X_n, Y_n)\}$$

其中，$X's$ 是输入的特征，$Y's$ 是标签，n 是数据点的数量。我们假设一个机器学习模型：f: $X \to Y$ 已经被训练。这个模型可以是一个经典的机器学习模型例如线性回归、支持向量机，或者深度学习技术例如全连接或卷积网络。目标是去估计模型输出的预测集。下面我们将描述保形预测的步骤：

保形预测的基本流程

a. 选择合适的不符合度量，计算不符合分数

确定一个适当的不符合度量 $s(X_1, Y_1) \in R$ 来测量模型输出 $\hat{y}'s$ 与标签 $y's$ 之间的差异。这个不符合度量非常重要，因为它决定了我们能得到什么样的预测集。例如，在回归问题中，我们可以用 $|\hat{y} - y|$ 作为评分函数。通过这种方式，所得到的预测集的值在预测值 $\hat{y}$ 周围的 L_1 范式球内；在分类问题中，我们可以用 $(1 - \hat{y_i})$ 作为评分函数，其中 $\hat{y_i}$ 是真实类别对应的预测结果。

b. 计算不符合分数的 $(1 - a)$ 分位数

将 $\hat{q}$ 计算为不符合分数 $s_1 = s(X_1, Y_1), \ldots, s_n = s(X_n, Y_n)$ 的 $(1 - a)$ 分位数。在完全保形预测方法中，我们需要训练 m 个模型来计算评分并构造预测集，其中 m 为可能的取值的个数。这无疑在计算上非常昂贵。为了降低计算复杂度，可以采用分裂保形预测的方法，分裂保形预测将整个训练集分割成合适的训练集和校准集 (calibration set)，然后，只对训练集进行训练，仅在校准集上计算不符合评分，这样，我们只需要对模型进行一次训练。

c. 使用模型预测和不符合分数的 $(1 - a)$ 分位数构造预测集

使用分位数来构建新的数据的预测集，在使用保形预测进行分类时，得到的预测集可以表示为：$\tau(X_{n+1}) = \{y : s(X_{n+1}, y) \leq \hat{q}\}$。在使用保形预测进行分类时，得到的预测集可以表示为：$\hat{C}_n = [\hat{f}_{n1}(x) - \hat{q_n}, \hat{f}_{n1}(x) + \hat{q_n}]$

10.1.3 预测集的有效性

我们的保形预测总是有效的。Glenn Shafer 和 Vladimir Vovk 着眼于可交换性和独立性之间的关系，得出了可交换序列的大数定律，这为我们认为 95% 的预测区域在 95% 的时间内是正确的提供了信心基础。保形预测最重要的新颖之处在于其连续误差是概率独立的。这使得我们能够以一种不同寻常的直接方式来解释 “95% 的正确率”。

对于保形预测的有效性的精确讨论实际上需要我们区分两种有效性: 保守的和精确的。一般来说，保形预测是保守有效的: 当它输出一个 $1-\alpha$ 集 (也就是说，一组预测集置信水平为 $1-\alpha$ 时错误的概率不大于 α，并且在预测连续的例子时，它所犯的错误之间几乎没有相关性。这意味着，根据大数定律，在置信水平 $1 - \alpha$ 上的长期错误频率约为 α 或更小。在实践中，保守性往往不是很大，尤其是当 n 很大时，经验结果显示，长期误差的频率与 α 非常接近。然而，从理论

的角度来看，为了获得准确的有效性，我们必须在预测过程中引入一个随机误差 α，其中 $1-\alpha$ 集出现错误的概率正好是 α，错误在不同的试验中是独立产生的，而长期错误的频率收敛到 α。

保形预测可以提供数学上严格的保证。设 Y_{n+1} 为真值。Y 可以是分类问题中的一个分类标签，也可以是回归问题中的一个实值。设 $\tau(X_{n+1})$ 是一个预测集（或区间）。如果 Y_{n+1} 在 $\tau(X_{n+1})$ 内，我们将其定义为 $\tau(X_{n+1})$ 覆盖 Y_{n+1}，即：$Y_{n+1} \in \tau(X_{n+1})$。然后，给定一组同独立分布样本 $\{(X_1, Y_1), \ldots, (X_n, Y_n)\}$，保形预测集满足以下覆盖保证：$P(Y_{n+1} \in \tau(X_{n+1})) \geq 1-a$。

基于可交换性 (exchangeability) 假设的覆盖保证的证明可以在文章 A Gentle Introduction to Conformal Prediction and Distribution-Free Uncertainty Quantification 的附录中找到。以上覆盖率保证在有限样本的情况下也是成立的，且与普通的统计推断通常都对变量的分布有更严格的假定 (比如 Gaussianity) 不同，该保证仅需要非常弱的条件 (可交换性),。

10.1.4 效率

在分类问题中，在不同的置信度下，保形预测的到的预测集中可能包含一个标签，也可能包含多个标签。在回归问题中，预测集通常是值的区间。在保证一定置信度的情况下，我们希望预测集越小越好。这就是我们通常所说的效率。

10.2 保形回归

保形预测可以用于回归，也可以用于分类，接下来我们首先以回归为例，解释保形预测的思想和原理。保形预测理论背后的基本思想与样本分位数有关。

10.2.1 保形预测基本思想

假设有独立同分布的标量随机变量样本 $U_1, \ldots, U_n$（事实上，此处独立同分布的假设可以被更弱的假设——可交换性替代）。对于一个给定的误覆盖水平 $\alpha \in (0,1)$ 和另一个独立同分布的样本 U_{n+1}，当我们基于 $U_1, \ldots, U_n$ 定义样本分位数 $\widehat{q}_{1-\alpha}$ 为：

$$\widehat{q}_{1-\alpha} = \begin{cases} U_{(\lceil (n+1)(1-\alpha) \rceil)} & \text{if } \lceil (n+1)(1-\alpha) \rceil \leq n \\ \infty & \text{otherwise} \end{cases},$$

可以得到：

$$\mathbb{P}\left(U_{n+1} \leq \widehat{q}_{1-\alpha}\right) \geq 1-\alpha,$$

其中，$U_{(1)} \leq \cdots \leq U_{(n)}$ 表示 $U_1, \ldots, U_n$ 的次序统计量。通过可交换性，U_{n+1} 在 $U_1, \ldots, U_n, U_{n+1}$ 中的排序是在集合 $\{1, \ldots, n+1\}$ 上均匀分布的，因此以上有限样本覆盖性质是很容易被验证的。

在回归问题中，我们观测到独立同分布的样本 $Z_i = (X_i, Y_i) \in \mathbb{R}^d \times \mathbb{R} \sim P, i = 1, \ldots, n$，我们可以考虑以下方法来构造 Y_{n+1} 在新特征值 X_{n+1} 处的预测区间，其中 (X_{n+1}, Y_{n+1}) 是分布 P 中一个独立的随机变量。按照上述思想，我们可以形成以下预测区间：

$$C_{\text{naive}}\ (X_{n+1}) = \left[\widehat{\mu}\left(X_{n+1}\right) - \widehat{F}_n^{-1}(1-\alpha), \widehat{\mu}\left(X_{n+1}\right) + \widehat{F}_n^{-1}(1-\alpha)\right],$$

其中，$\hat{\mu}$ 是估计的回归函数预测器，$\hat{F}_n$ 是拟合残差 $\left|Y_i - \widehat{\mu}\left(X_i\right)\right|, \quad i = 1, \ldots, n$ 的经验分布，$\widehat{F}_n^{-1}(1-\alpha)$ 是 $\hat{F}_n$ 的 $(1-\alpha)$ 分位数。假设估计的回归函数 $\hat{\mu}$ 是准确的，则该预测区间在大样

本情况下是有效的 (即，估计的拟合残差分布的 $(1-\alpha)$ 分位数 $\widehat{F}_n^{-1}(1-\alpha)$ 足够接近总体残差 $|Y_i - \mu(X_i)|, \quad i = 1, \ldots, n$ 的 $(1-\alpha)$ 分位数。) 保证 $\hat{\mu}$ 的准确性通常需要数据分布 P 和预测器 $\hat{\mu}$ 本身均满足一定的条件, 例如适当地选择的预测模型和调优参数。

10.2.2 完全保形预测

一般来说，上述方法得到的预测区间可以粗略地覆盖真实值，因为拟合残差分布往往是向下倾斜的。保形预测区间 (Vovk, Gammerman 和 Shafer (2005);Vovk, Nouretdinov 和 Gammerman (2009);Lei, Robins, 和 Wasserman(2013);Lei 和 Wasserman(2014) 克服了上述原始方法预测区间的缺陷，并且，某种程度上值得注意的是，保形预测可以保证提供适当的有限样本覆盖，而无需对 P 或 μ 进行任何假设 (除了 μ 是数据点的对称函数)。

考虑以下策略: 对于每个值 $y \in \mathbb{R}$，我们构造一个增广回归估计器 $\hat{\mu}_y$，它在增广数据集 $Z_1 \ \ldots, Z_n, (X_{n+1}, y)$ 上训练。现在我们定义:

$$\begin{aligned} R_{y,i} &= |Y_i - \widehat{\mu}_y(X_i)|, \quad i = 1, \ldots, n \quad \text{和} \\ R_{y,n+1} &= |y - \widehat{\mu}_y(X_{n+1})|, \end{aligned}$$

并且我们在拟合残差中对 $R_{y,n+1}$ 排序，计算:

$$\begin{aligned} \pi(y) &= \frac{1}{n+1} \sum_{i=1}^{n+1} \mathbf{1}\{R_{y,i} \le R_{y,n+1}\} \\ &= \frac{1}{n+1} + \frac{1}{n+1} \sum_{i=1}^{n} \mathbf{1}\{R_{y,i} \le R_{y,n+1}\}, \end{aligned}$$

即增广样本中残差绝对值小于最后一个样本点的残差绝对值 $R_{y,n+1}$ 的样本所占的比例。其中 $\mathbf{1}\{\cdot\}$ 表示示性函数。由于数据点的可交换性和 $\hat{\mu}$ 的对称性，当我们在估计 $y = Y_{n+1}$ 时，我们发现构造的次序统计量 $\pi(Y_{n+1})$ 在集合 $\{1/(n+1), 2/(n+1), \ldots, 1\}$ 上是均匀分布的, 这意味着:

$$\mathbb{P}\left((n+1)\pi(Y_{n+1}) \le \lceil (1-\alpha)(n+1) \rceil\right) \ge 1 - \alpha.$$

我们可以将上述现象解释为: 对于零假设为 $H_0 : Y_{n+1} = y$ 的假设检验，$1-(Y_{n+1})$ 为其提供了一个有效的 (保守的)p 值。

通过对上式在所有 $y \in R$ 上可能值的变换，我们可以得到在 X_{n+1} 处的保形预测区间，即:

$$C_{\text{conf}}(X_{n+1}) = \{y \in \mathbb{R} : (n+1)\pi(y) \le \lceil (1-\alpha)(n+1) \rceil\}.$$

每次我们想要产生一个预测区间 (对一个新的特征值) 时，都必须重复以上步骤。为了完整起见，我们总结为算法 1。

表 10.1: 保形预测算法

算法 1：保形预测

输入： 数据 $(X_i, Y_i), i = 1, \ldots, n$, 显著性水平 $\alpha \in (0, 1)$, 回归算法 A, 用于构造预测区间的新的点 $X_{new} = \{X_{n+1}, X_{n+2}, \ldots\}$, 和作为试验集的值 $y_{trial} = \{y_1, y_2, \ldots\}$

输出： X_{new} 中每一个元素对应的预测区间

for $x \in X_{new}$ **do**

  for $y \in y_{new}$ **do**

    $\hat{\mu}_y = \mathcal{A}\left(\{(X_1, Y_1), \ldots, (X_n, Y_n), (x, y)\}\right)$

    $R_{y,i} = \left|Y_i - \hat{\mu}_y\left(X_i\right)\right|, \quad i = 1, \ldots, n,$

and $R_{y,n+1} = \left|y - \hat{\mu}_y\left(x\right)\right|$

    $\pi(y) = \left(1 + \sum_{i=1}^{n} \mathbf{1}\left\{R_{y,i} \leq R_{y,n+1}\right\}\right) / (n+1)$

  end for

  $C_{\text{conf}}\left(x\right) = \{y \in y_{\text{trial}} : (n+1)\pi(y) \leq \lceil (1-\alpha)(n+1) \rceil\}$

end for

Return $C_{\text{conf}}\left(x\right)$, for each $x \in X_{new}$

通过构造，上述保形预测得到的预测区间具有有效的有限样本覆盖，并且这个区间是精确的，意味着它没有过多地覆盖。以下我们对此进行说明：

如果 $(X_i, Y_i), i = 1, \ldots, n$ 为独立同分布的,那么对于一个新的独立同分布的点 (X_{n+1}, Y_{n+1}), 保形预测建立的预测带 $C_{\text{conf}}\left(X_{n+1}\right)$ 有以下覆盖保证：

$$\mathbb{P}\left(Y_{n+1} \in C_{\text{conf}}\left(X_{n+1}\right)\right) \geq 1 - \alpha,$$

如果我们另外假设对所有 $y \in R$，拟合的绝对残差 $R_{y,i} = \left|Y_i - \widehat{\mu}_y\left(X_i\right)\right|, i = 1, \ldots, n$ 有一个连续的联合分布，那么下式也成立：

$$\mathbb{P}\left(Y_{n+1} \in C_{\text{conf}}\left(X_{n+1}\right)\right) \leq 1 - \alpha + \frac{1}{n+1}.$$

该结论的证明详见文章 *Distribution-Free Predictive Inference for Regression*。

10.3 保形方法

10.3.1 分裂保形预测

上一节中讲述的原始的保形预测方法即完全保形法的计算量较大。对于任何 X_{n+1} 和 y，为了分辨 y 是否包含在 $C_{conf}(X_{n+1})$ 中，我们在增广数据集 (其中包括新的点 (X_{n+1}, y)) 上重新训练模型, 并重新计算和排序绝对残差。在某些应用中，X_{n+1} 不一定能被观测到，预测区间是通过在所有的 (x, y) 上估计 $\mathbf{1}\{y \in C_{\text{conf}}\left(x\right)\}$ 建立的。在核密度估计和核回归的特殊情况下，Lei, Robins, Wasserman(2013) 和 Lei, Wasserman(2014) 描述了对完全保形预测集的简单而准确的逼近。在低维线性回归中，使用 Sherman-Morrison 的升级方案可以降低完全保形方法的复杂度，

该方案可以节省每次查询点 (x, y) 改变时求解一个全线性系统的成本。但在高维回归中，我们可能会使用相对复杂的 (非线性) 预测器，如套索回归，执行有效的完全保形预测仍然是一个有待解决的问题。

幸运的是，有一种替代方法是完全通用的，我们称之为分裂保形预测，其计算成本只是完全保形方法的一小部分。分裂保形方法采用样本分割将拟合步骤和排序步骤分离，其计算代价与拟合步骤相对简单。类似的想法出现在在线预测文献中，被称为归纳保形预测 (Papadopoulos et al.(2002); Vovk, Gammerman 和 Shafer(2005)。算法 2 中总结的分裂保形算法改编自 Lei,Rinaldo 和 Wasserman(2015))。在这里，以及以后讨论分裂保形推理时，为了简单起见，我们假设样本容量 n 是偶数，因为当 n 是奇数时只需要非常小的变化。

表 10.2: 分裂保形预测算法

算法 2：分裂保形预测
输入： 数据 $(X_i, Y_i), i = 1, \ldots, n$, 显著性水平 $\alpha \in (0, 1)$, 回归算法 A,
输出： $x \in \mathbb{R}^d$ 上的预测带
将 {1, ..., n} 随机分裂为两个大小相等的数据集 $\mathcal{I}_1, \mathcal{I}_2$
$\hat{\mu} = \mathcal{A}(\{(X_i, Y_i) : i \in \mathcal{I}_1\})$
$R_i = \lvert Y_i - \hat{\mu}(X_i)\rvert, i \in \mathcal{I}_2$
$d = \{R_i : i \in \mathcal{I}_2\}$ 中第 k 小的值，其中 $k = \lceil (1-\alpha)(\frac{n}{2}+1)\rceil$
Return $C_{\text{split}}(x) = [\hat{\mu}(x) - d, \hat{\mu}(x) + d]$, for all $x \in \mathbb{R}^d$

如果 $(X_i, Y_i), i = 1, 2, \ldots, n$ 是独立同分布的，对于一个新的数据点 (X_{n+1}, Y_{n+1})，由算法 2 建立的分裂保形预测带 C_{split}，满足：

$$\mathbb{P}\left(Y_{n+1} \in C_{\text{split}}\ (X_{n+1})\right) \geq 1 - \alpha.$$

另外，如果我们假设残差 $R_i, i \in \mathcal{I}_2$ 具有连续的联合分布，那么

$$\mathbb{P}\left(Y_{n+1} \in C_{\text{split}}\ (X_{n+1})\right) \leq 1 - \alpha + \frac{2}{n+2}.$$

与原始保形方法相比，分裂保形预测除了具有极高的效率外，在记忆需求方面也具有优势。例如，如果回归过程 A(在算法 2 的表示法中) 涉及变量选择，如套索或前向逐步回归，当我们在估计新的点 $X_i, i \in \mathcal{I}_2$ 时，我们只需要存储选择的变量，并为排序步骤计算残差。当原始变量集非常大，而选择的变量集非常小时，这可以大大节省内存。

分裂保形预测也可以使用不均衡分裂实现。对于 $\rho \in (0, 1)$，$|\mathcal{I}_1| = \rho n$ 和 $|\mathcal{I}_2| = (1-\rho)n$, 在某些情况下，例如，当回归过程很复杂的时候，也许选择 $\rho > 0.5$ 可以使训练得到的预测器 $\hat{\mu}$ 更准确。

10.3.2 Jackknife 保形预测

Jackknife 保形预测是一种计算复杂性介于完全保形法和分裂保形法之间的保形预测方法。该方法利用留一残差的分位数来定义预测区间，具体内容如算法 3 所示。

表 10.3: Jackknife 保形预测算法

算法 3：Jackknife 预测带

输入： 数据 $(X_i, Y_i), i = 1, \ldots, n$, 显著性水平 $\alpha \in (0, 1)$, 回归算法 A,
输出： $x \in \mathbb{R}^d$ 上的预测带
for $i \in \{1, \ldots, n\}$ **do**
$\hat{\mu}^{(-i)} = \mathcal{A}(\{(X_\ell, Y_\ell) : \ell \neq i\})$
$R_i = \left|Y_i - \hat{\mu}^{(-i)}(X_i)\right|$
end for
$d = \{R_i : i \in \{1, \ldots, n\}\}$ 中第 k 小的值，其中 $k = \lceil n(1-\alpha) \rceil$
Return $C_{\text{Jack}}(x) = [\hat{\mu}(x) - d, \hat{\mu}(x) + d]$, for all $x \in \mathbb{R}^d$

与分裂保形法相比，Jackknife 法的一个优点是它在构造绝对残差时使用更多的训练数据，随后再构造分位数。这意味着它通常可以产生更短的间隔长度。我们注意到，由于对称性，Jackknife 法具有有限样本内覆盖：

$$\mathbb{P}\left(Y_i \in C_{\text{jack}}\ (X_i)\right) \geq 1 - \alpha, \quad \text{for all } i = 1, \ldots, n$$

但是就样本外覆盖而言 (真正的预测推断)，它的属性比较脆弱。Butler 和 Rothman(1980) 表明，在低维线性回归设置中，Jackknife 法在足够强的正则条件下产生渐近有效区间，它们也意味着线性回归估计的一致性。最近，Steinberger 和 Leeb(2016) 在高维回归设置中建立了 Jackknife 区间的渐近有效性，它们不需要基本估计量 $\hat{\mu}$ 的一致性，但它们需要 $\hat{\mu}$ 上的一致渐近均方误差界 (和渐近稳定条件)。保形预测法不需要这样的条件。此外，Butler 和 Rothman(1980) 和 Steinberger 和 Leeb(2016) 的分析假设一个标准的线性模型设置，其中回归函数本身是特征的线性函数，特征独立于误差，误差是同方差的，为了使分裂保形法 (和完全保形法) 具有有限的简单有效性，这些条件都是不需要的。

10.3.3 局部加权保形预测

当噪声是异方差的时候，我们可以用局部加权版本替换完全保形方法或分裂保形方法中的绝对残差，即：使用

$$V_i = \frac{|Y_i - \hat{f}_n(X_i)|}{\hat{\sigma}_n(X_i)}$$

替代

$$R_i = |Y_i - \hat{f}_n(X_i)|$$

其中 $\hat{\sigma}_n^2(x)$ 是绝对残差 $Var(|Y - \hat{f}_n(X)||X = x)$ 的方差函数的一个估计。(注意：$\hat{f}_n$ 和 $\hat{\sigma}_n$ 可以联合计算，也可以单独计算。)

局部加权对预测波段的影响：根据需要，其局部宽度可以有很大变化。在分裂保形预测版本中：

$$\tilde{C}_n = [\hat{f}_{n1}(x) - \hat{\sigma}_{n1}(x)\tilde{q}_n, \hat{f}_{n1}(x) + \hat{\sigma}_{n1}(x)\tilde{q}_n]$$

其中 $\tilde{q}_n = Quantile(1 - \alpha; \{V_i\}_{i \in D_2} \bigcup \{\infty\})$。

异方差噪声示例 (Heteroskedastic noise example)

Example: n=100, d=1, Heteroskedastic noise

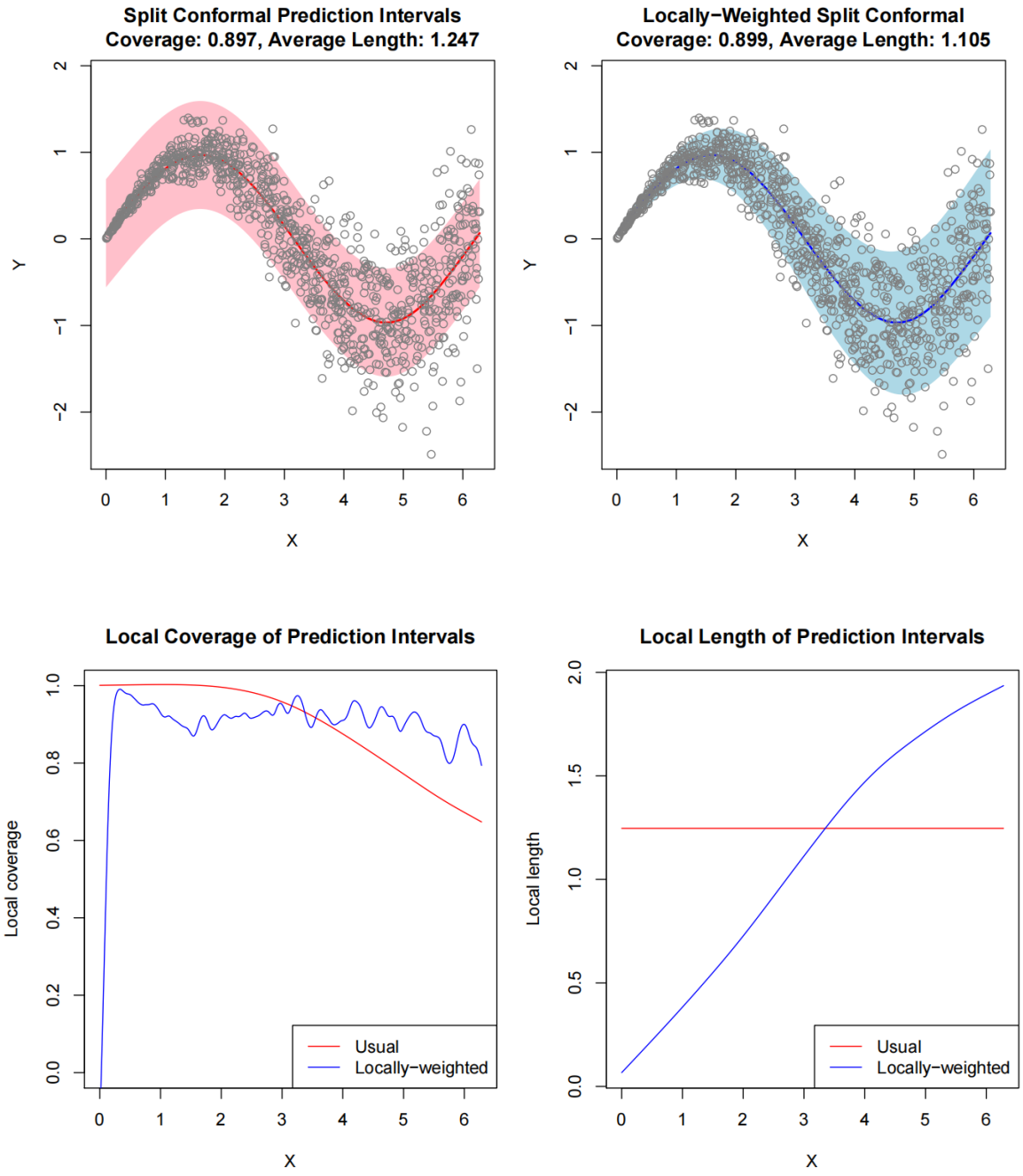


图 10.3: 异方差噪声

从图10.3 可以看出，通过引入局部权重，我们可以获得能适应数据异质性的预测区域，在保证和之前的方法具有相同的有效性的同时，它具有更小的平均长度，局部的覆盖率也约等于我们想要的值（比如说 0.9）。

10.4 保形分类

选择合适的不符合度量是进行保形预测的重要步骤，我们在使用保形预测法进行分类时，应该根据数据类型和预测方法来选择不符合度量。下面我们将介绍两种不同的不符合度量下保形预测的分类应用。

10.4.1 Softmax 法

在机器学习尤其是深度学习中，Softmax 是个非常常用而且比较重要的函数，尤其在多分类的场景中使用广泛。它把一些输入映射为 $0-1$ 之间的实数，并且归一化保证和为 1，因此多分类的概率之和也刚好为 1。Softmax 函数形式如下：

$$Softmax(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}, \quad i = 1, \ldots, n.$$

在分类保形预测中，我们可以将 Softmax 函数作为我们的预测函数，Softmax 方法经常用于图片分类，下面我们简单介绍 Softmax 方法下的保形预测。

假设我们有一个图像 X 和类别 $Y \in \{1, \ldots, k\}$。首先我们使用训练数据和 Softmax 函数得到预测模型，也可称其为分类器。分类器可以对于每个类别输出 Softmax 分数：$\hat{f}(x) \in [0,1]^k$。然后，我们取适当数量的未用于训练的新数据点 $(X_1, Y_1), \ldots,$
(X_n, Y_n) 作为校准数据集。利用 $\hat{f}$ 和校准数据，我们试图构建一个可能的标签的置信集 $\mathcal{T}(x) \subset \{1, \ldots, k\}$，且这个置信集在下面意义下是有效的:

$$1 - \alpha \leq \mathbb{P}\left(Y_{n+1} \in \mathcal{T}\left(X_{n+1}\right)\right) \leq 1 - \alpha + \frac{1}{n+1}.$$

其中，(X_{n+1}, Y_{n+1}) 是来自同一分布的新测试点。换句话说，置信集包含正确标签的概率几乎正好是 $(1-\alpha)$，我们称这种性质为边际覆盖。为了使用 $\hat{f}$ 和校准数据构造 $\mathcal{T}$，我们将执行一个简单的校准步骤。首先，我们设置不符合分数为 1 减去正确类别的 Softmax 输出：

$$s_i = 1 - \hat{f}\left(X_i\right)_{Y_i},$$

如果模型不好，s_i 将会很大。然后是关键步骤：定义 $\hat{q}_n$ 为 $s_1, \ldots, s_n$ 的 $\lceil (1-\alpha)(n+1) \rceil$ 分位数，这实际上是一个 $(1-\alpha)$ 分位数，但是有一个小小的修正。最后，对于一个新的数据点（X_{n+1} 已知，但 Y_{n+1} 未知），产生一个预测集：

$$\mathcal{T}\left(X_{n+1}\right) = \left\{y : \hat{f}\left(X_{n+1}\right)_y > 1 - \hat{q}\right\},$$

该预测集包含所有的 Softmax 输出足够大的类别，如图10.4 所示。值得注意的是，不论我们应用什么样的预测模型，选择任何分布的数据集，这个算法得到的预测集都是具有覆盖保证的。

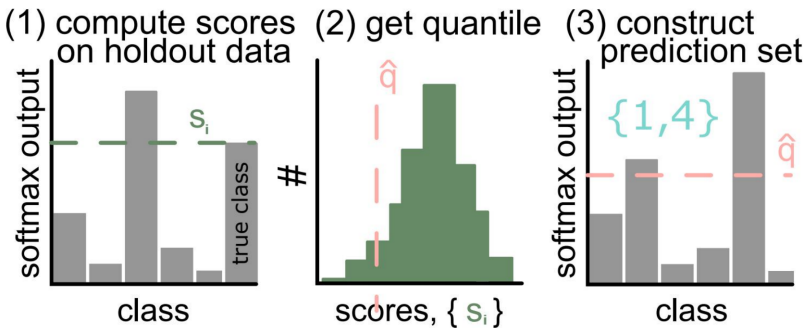


图 10.4: 保形分类——Softmax 法

上述校准步骤的 PyTorch 代码如下:

```
#First, get conformal scores. n = calib_y.shape[0]
scores = 1-model(calib_x).softmax(dim=1)[:,calib_y]
# Second,get the adjusted quantile
qhat = torch.quantile(scores,np.ceil((n+1)(1-alpha))/n)
#Finally, deploy!
prediction_sets = (model(test_x) > (1-qhat)).nonzero()
```

10.4.2 最近邻法

最近邻法也是我们进行分类时常用的方法之一。在保形预测中，我们使用最近邻法分类时，预测器将新的样本分类为与其距离最近的样本的类别。

假设有随机变量 $z_1 = (x_1, y_1), z_2 = (x_2, y_2), ..., z_{n-1} = (x_{n-1}, y_{n-1})$，每个 z_i 包含一个特征 x_i 和一个标签 y_i。然后我们对于一个新的样本点 $z = (x, y)$，我们只能观察到 x 而不知道 y 。最近邻法寻找距离 x 最近的 x_i，并使用它的类别 y_i 作为我们的预测 y 。如果仅仅有两个标签，或者没有合适的方法去测量标签之间的距离，我们不能度量这个预测的正确性。但是我们可以通过比较 x 到与其有相同标签的旧例子的距离和 x 到与其有不同标签的旧例子的距离，来判断新例子与旧例子之间的不符合度量，因此我们用下式表示我们的不符合分数:

$$A(B, z) = \frac{\min\{|x_i - x| : 1 \le i \le n-1, y_i = y\}}{\min\{|x_i - x| : 1 \le i \le n-1, y_i \neq y\}}$$

下面我们以鸢尾花数据为例，具体讲述最近邻法下的保形分类。鸢尾花数据集是是一类多重变量分析的数据集。数据集包含 150 个数据样本，分为 3 类，每类 50 个数据，每个数据包含 4 个属性。可通过花萼长度，花萼宽度，花瓣长度，花瓣宽度 4 个属性预测鸢尾花卉属于（Setosa，Versicolour，Virginica）三个种类中的哪一类。我们选取这个数据集中 50 株山鸢尾花和 50 株杂色鸢尾花的数据进行研究。鸢尾花的种类（山鸢尾花 s、杂色鸢尾花 v）作为标签。我们分类的依据是花萼长度和花瓣宽度。为了说明保形算法如何用于分类，我们从 100 株植物中随机选择了 25 株。前 24 株花萼的长度和鸢尾花的种类如图10.5 所示。

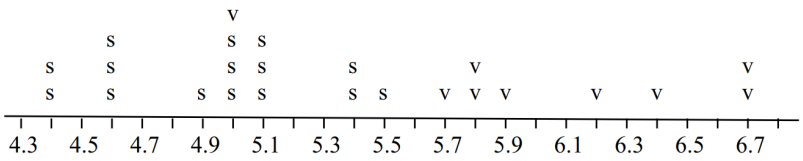


图 10.5: 随机选取的 25 株植物中前 24 株植物萼片长度和种类

第 25 株的花萼长度为 6.8，我们会把它归类为山鸢尾花还是杂色鸢尾花? 因为 6.8 是样本中最长的花萼，几乎任何合理的方法都会将该植物归类为杂色，实际上这也是正确的答案。但如果我们想知道这个分类有多大信心，应该怎么做? 这时我们就可以使用最近邻法，我们对该数据使用上面的不符合分数得到：

$$s_i = \frac{\min\{|x_j - x_i| : 1 \le j \le 25 \quad \& \quad j \ne i \quad \& \quad y_j = y_i\}}{\min\{|x_j - x_i| : 1 \le j \le 25 \quad \& \quad j \ne i \quad \& \quad y_j \ne y_i\}}$$

该分数的具体结果如表10.4 所示。表10.4 的第四列和第五列分别给出了 $y_{25} = s$ 和 $y_{25} = v$ 得到的不符合度 s_i。第四列显示 $z_{25} = (6.8, s)$，而第五列显示 $z_{25} = (6.8, v)$。如果上式的分子和分母都等于 0，我们让比率也为 0。例如，表10.4 中的第一株植物就是这样，它与第 7 株、第 13 株、第 15 株有相同的花萼长度，均为 5.0，其中第 7 株和第 13 株为山鸢尾花，与第一株类别相同，第 15 株为杂色鸢尾花，与第一株类别不同，此时，分子分母均为 0，则令其不符合分数也为 0。

表10.4 保形算法得到不符合分数的分位数分别为：$p_s = 0.08, p_v = 0.32$。因此当 $1-\alpha > 0.92$ 时，在 $1-\alpha$ 置信区间内认为第 25 株鸢尾花是山鸢尾花；当 $1-\alpha > 0.68$ 时，在 $1-\alpha$ 置信区间内认为第 25 株鸢尾花是杂色鸢尾花。以下是几个 α 水平的预测区域:

- $\Gamma^{0.08} = \{v\}$: 有 92% 的信心，预测 $y_{25} = v$。
- $\Gamma^{0.05} = \{s, v\}$: 如果我们将预测 y_{25} 的置信度提高到 95%，这个预测是完全不提供信息的。
- $\Gamma^{1/3} = \emptyset$: 如果我们将置信度降低到 2/3，我们得到的预测是我们不知道第 25 株鸢尾花的种类，y_{25} 将在空集中。

表 10.4: 最近邻法下的不符合分数

	sepal length	species	s_i for $y_{25} = s$	s_i for $y_{25} = v$
z_1	5.0	s	0	0
z_2	4.4	s	0	0
z_3	4.9	s	1	1
z_4	4.4	s	0	0
z_5	5.1	s	0	0
z_6	5.9	v	0.25	0.25
z_7	5.0	s	0	0
z_8	6.4	v	0.50	0.22

z_9	6.7	v	0	0
z_{10}	6.2	v	0.33	0.29
z_{11}	5.1	s	0	0
z_{12}	4.6	s	0	0
z_{13}	5.0	s	0	0
z_{14}	5.4	s	0	0
z_{15}	5.0	v	∞	∞
z_{16}	6.7	v	0	0
z_{17}	5.8	v	0	0
z_{18}	5.5	s	0.50	0.50
z_{19}	5.8	v	0	0
z_{20}	5.4	s	0	0
z_{21}	5.1	s	0	0
z_{22}	5.7	v	0.50	0.50
z_{23}	4.6	s	0	0
z_{24}	4.6	s	0	0
z_{25}	6.8	s	13	
z_{25}	6.8	v		0.077
p_s			0.08	
p_v				0.32

10.5 软件

下面我们通过 python 实现保形预测：

10.5.1 完全保形预测的简单代码

我们可以根据完全保形预测的原理，通过构造不符合分数，确定分位点，得到保形预测区间。

```
import random
import sys
import statistics

eps = 0.20 #显著性水平
M = 1000

#定义不符合分数
```

```
def A(B, z):
    z0 = statistics.mean(B) #z0为预测值
    ans = abs(z0 - z) #取预测值减真实值的绝对值为不符合分数
    return ans

#完全保形预测 判断分位数点
def pz(z, zn):
    an = A(z, zn)   #an = z - zn
    z.append(zn)   #z后加上zn
    cnt = 0
    n = len(z)   #z的长度为n
    for i in range(n):
        z0 = z[:i] + z[i + 1:]   #z0等于z去掉zi
        a = A(z0, z[i])
        if a >= an:
            cnt += 1
    z.pop()   #删除z中最后一个值，即加进去的zn
    return cnt / n   #返回分位数点


def G(z):
    ans = []
    for zn in range(M):
        p = pz(z, zn)   #样本分位数点
        if p > eps:
            ans.append(zn)   #如果样本分位数点大于显著性水平，则预测值zn包含在
                eps置信水平下的预测中
    return ans


Z = [17, 20, 10, 17, 12, 15, 19, 22, 17, 19, 14, 22, 18, 17, 13, 12, 18, 15, 17]

#定义gen为符合期望为mu，标准差为sigma的正态分布的随机数
def gen():
    # return Z.pop()
    return int(random.gauss(mu, sigma))


mu = 500
sigma = 100
n = 2000
z = []
z.append(gen())
correct = total = 0
for i in range(n - 1):
    x = gen()
```

```
    print("%03d-%03d" % (min(G(z)), max(G(z))))
    if x in G(z):
        correct += 1
    total += 1
    print("%03d/%03d = %.0f" % (correct, total, correct / total * 100))
    z.append(x)

print(correct / total)
```

10.5.2 保形分类和保形回归

另外我们也可以使用 python 调用 nonconformist 包，直接进行保形预测。下面是我们用该方法分别进行保形分类和保形回归的代码：

首先是保形分类，在这里，我们以使用支持向量机法对鸢尾花数据集进行分类为例：

```
from sklearn.datasets import load_iris
import numpy as np
from sklearn.svm import SVC
from nonconformist.cp import IcpClassifier
from nonconformist.nc import NcFactory

iris = load_iris()
idx = np.random.permutation(iris.target.size)   #对鸢尾花数据数量进行随机排列

# 将数据分为适当的训练集、校准集和测试集
idx_train, idx_cal, idx_test = idx[:50], idx[50:100], idx[100:]

model = SVC(probability=True)  # 建立预测模型
nc = NcFactory.create_nc(model)  # 建立不符合度量
icp = IcpClassifier(nc)  # 建立一个归纳保形分类器

# 用训练集拟合分类器
icp.fit(iris.data[idx_train, :], iris.target[idx_train])

# 用校准集校准分类器
icp.calibrate(iris.data[idx_cal, :], iris.target[idx_cal])

# 用测试集生成在95%置信度下的预测区间
prediction = icp.predict(iris.data[idx_test, :], significance=0.05)

# 打印预测结果
print(prediction[:5, :])
```

得到预测结果：

```
[[ True False False]
```

```
 [ True False False]
 [ True False False]
 [False  True False]
 [False  True False]]
```

该预测结果是一个布尔型 numpy 数组，行数是测试集的数量，列数是类别数，每一行是一个代表在特定显著性水平下的类标签是否包含在预测区间中的布尔向量。在这个例子中，对于给定的测试对象，我们也许会得到一个 $[True \quad True \quad False]$ 的向量，这意味着有 95% 的可能，True 对应的两个类别中有一个是正确的。

其次是保形回归，我们以使用随机森林法对波士顿数据集进行回归为例：

```
from sklearn.datasets import load_boston
import numpy as np
from sklearn.ensemble import RandomForestRegressor
from nonconformist.cp import IcpRegressor
from nonconformist.nc import NcFactory

boston = load_boston()
idx = np.random.permutation(boston.target.size)

# 将数据分为适当的训练集、校准集和测试集
idx_train, idx_cal, idx_test = idx[:300], idx[300:399], idx[399:]

model = RandomForestRegressor()  # 建立预测模型
nc = NcFactory.create_nc(model)  # 建立不符合度量
icp = IcpRegressor(nc)  # 建立一个归纳保形回归器

# 用训练集拟合回归模型
icp.fit(boston.data[idx_train, :], boston.target[idx_train])

# 用校准集校准模型
icp.calibrate(boston.data[idx_cal, :], boston.target[idx_cal])

# 用测试集生成在95%置信度下的预测区间
prediction = icp.predict(boston.data[idx_test, :], significance=0.05)

# 打印预测结果
print(prediction[:5, :])
```

得到预测结果：

```
[[ 26.47  39.45]
 [ 17.1   30.08]
 [ 15.79  28.77]
 [ 17.03  30.01]
 [  7.08  20.06]]
```

该预测结果是一个数值型 numpy 数组，行数是测试集的数量，列数是 2，每一行是一个代表区间下界和上界的向量，表示在特定显著性水平下的预测区间。在这个例子中，对于给定的测试对象，我们也许会得到一个数值向量 [26.47　39.45]，这意味着在 95% 置信度下得到的预测区间为 [26.47,　39.45]。