# 隐马尔可夫模型

Hidden Markov Model

learning note For reading translation

我真的不懂忧郁

Delft University of Technology

TU Delft

# 隐马尔可夫模型

## Hidden Markov Model
## learning note For reading translation

by


## 我真的不懂忧郁

| Student Name | Student Number |
|---|---|
| First Surname | 1234567 |



Instructor: I. Surname
Teaching Assistant: I. Surname
Project Duration: Month, Year - Month, Year
Faculty: Faculty of Aerospace Engineering, Delft

Cover: Canadarm 2 Robotic Arm Grapples SpaceX Dragon by NASA under CC BY-NC 2.0 (Modified)
Style: TU Delft Report Style, with modifications by Daan Zwaneveld


TU Delft

# Preface

*A preface...*

我真的不懂忧郁

*Delft, June 2024*

# Summary

*A summary...*

# 目录

# Nomenclature

*If a nomenclature is required, a simple template can be found below for convenience. Feel free to use, adapt or completely remove.*

## Abbreviations

| Abbreviation | Definition |
|---|---|
| ISA | International Standard Atmosphere |
| ... | |

## Symbols

| Symbol | Definition | Unit |
|---|---|---|
| $V$ | Velocity | [m/s] |
| ... | | |
| $\rho$ | Density | [kg/m$^3$] |
| ... | | |

# Chapter 1

# Background

## 1.1. 概览

概率图模型按照有向和无向分类

$$
\text{概率图模型}\begin{cases}\text{有向} \to \textit{Bayesian Network} \\ \text{无向} \to \textit{Markov Random Field}\end{cases}
$$

概率图模型加上对时序的考量的话

$$
\text{概率图模型} \to \textit{Dynamic Model}\begin{cases}\text{状态离散} \to \textit{HMM} \\ \text{状态连续}\begin{cases}\text{Kalmen Filter} \\ \text{Particle Filter}\end{cases}\end{cases}
$$

**隐马尔可夫模型**（*Hidden Markov Model*，*HMM*）是关于时序的概率模型，描述由一个隐藏的马尔可夫链随机生成不可观测的状态随机序列，再由各个状态生成一个观测而产生观测随机序列的过程。隐藏的马尔可夫链随机生成的状态的序列，称为**状态序列**；每个状态生成一个观测，而由此产生的观测的随机序列，称为**观测序列**，序列的每一个位置又可以看作是一个时刻。其形式定义如下

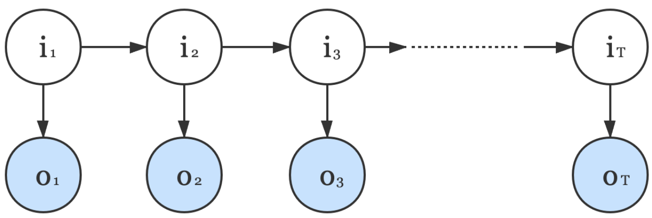


**图 1.1:** 隐马尔可夫模型，$i$ 是隐藏变量，$o$ 是观测变量

HMM 可以看作是一个三元组 $\lambda = (\pi, \mathcal{A}, \mathcal{B})$，其中 ,$\mathcal{A}$ 是状态转移矩阵，,$\mathcal{B}$ 是发射矩阵。HMM 模型主是三个基本问题和两个基本假设

$$\text{三个基本问题} \begin{cases} \text{Evaluation} \rightarrow P(O|\lambda) \text{ 前向后向} \\ \text{Learning} \rightarrow P(O|\lambda) \text{ 的 } \lambda \text{ 如何求} \\ \text{Decoding} \end{cases}$$

$$\text{两个基本假设} \begin{cases} \text{齐次马尔可夫假设} \\ \text{观测独立性假设} \end{cases}$$

## 两个假设

**齐次马尔可夫假设**: 未来与过去无关，只和当前的状态有关，即

$$P(i_{t+1}|i_t, i_{t-1}, \cdots, i_1, o_t, \cdots, o_1) = P(i_{t+1}|i_t) \tag{1.1}$$

**观测独立性假设**: 观测独立性假设是指在给定隐状态序列的条件下，观测序列中的每一个观测值都是相互独立的。

$$P(o_1, o_2, \cdots, o_T) = prod_{i=1}^{T} \tag{1.2}$$

# Chapter 2

# Evaluation 问题

首先我们来整理一下现在所有的前提假设，首先我们有所有可能的状态变量 $i_t$ 的状态集合 $Q$ 和所有可能的状态变量 $o_t$ 的观测集合 $V$:

$$Q = \{q_1, q_2, \cdots, q_N\}, \quad V = \{v_1, v_2, \cdots, v_M\}, \tag{2.1}$$

其中 $N$ 是可能的状态数，$M$ 是可能的观测数。
$I$ 是长度为 $T$ 的状态序列，$O$ 是对应的观测序列:

$$I = (I_1, I_2, \cdots, I_T), \quad O = (O_1, O_2, \cdots, O_T) \tag{2.2}$$

同时我们有参数 $\lambda = (\pi, \mathcal{A}, \mathcal{B})$，其中 $\pi$ 为初始状态概率向量

$$\pi = (\pi_i)_N, \quad \pi_i = P(I_1 = q_i),\ i = 1, 2, \cdots, N \tag{2.3}$$

$\mathcal{A}$ 是状态转移矩阵，

$$\mathcal{A} = [a_{ij}]_{N\times N} \tag{2.4}$$

其中 $a_{ij} = P(I_{t+1} = q_j | I_t = q_i)$。
$\mathcal{B}$ 是观测概率矩阵，

$$\mathcal{B} = [b_j(k)]_{N\times M} \tag{2.5}$$

其中 $b_j(k) = P(O_t = v_k | I_t = q_j)$。

## 2.1. 直接计算

直接计算就是直接依照概率图模型直接展开，但是计算量很大，计算复杂度是 $O(TN^T)$ 阶的。

$$P(O|\lambda) = \sum_{I_1} \cdot \sum_{I_2} \cdots \cdot \sum_{I_T} \pi_{i_1} \prod_{t=2}^{T} a_{i_{t-1},i_t} \prod_{t=2}^{T} b_{i_t}(O_t) \tag{2.6}$$

一共有 $T$ 个状态，每个状态 $N$ 种可能，所以复杂度为 $\mathcal{O}(N^T)$

## 2.2. 前向算法

首先我们先展示一下 Hidden Markov Model 的拓扑结构

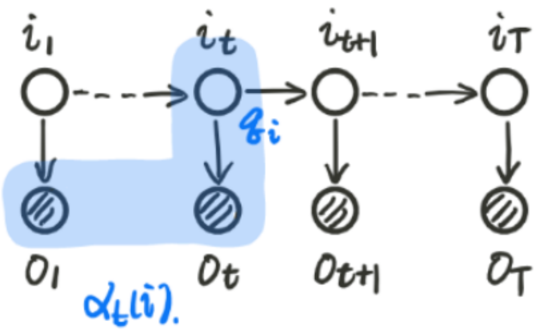


**图 2.1:** 前向传播算法的拓扑结构

前向算法的主要思想是，在之前所有的观测变量 $O_t$ 的前提下求出当前时刻的隐变量的概率，即

$$P(O|\lambda) = \sum_{i=1}^{n} P(O, I_T = q_i|\lambda) = \sum_{i=1}^{n} \alpha_T(i) \tag{2.7}$$

我们希望的是从 $\alpha_1(i)$ 开始，一步一步向前推从而获得 $\alpha_T$，也就是我们需要的是 $\alpha_t(i)$ 和 $\alpha_{t+1}(i)$ 具有某种递推关系，下面我们来推导这种关系

**proposition 2.2.1:** $\alpha_{t+1}(j) = b_j(O_{t+1}) \cdot a_{ij} \cdot \alpha_i(i)$，其中 $b_j(O_{t+1}) = \sum_{k=1}^{M} \nu_k$

$$\begin{aligned}
\alpha_{t+1}(j) &= P(O_1, \cdots, O_t, O_{t+1}, I_{t+1} = q_j|\lambda) \\
&= \sum_{i=1}^{N} P(O_1, \cdots, O_t, O_{t+1}, I_t = q_i, I_{t+1} = q_j|\lambda) \\
&= \sum_{i=1}^{N} \underbrace{P(O_{t+1}|I_{t+1} = q_j, \lambda)}_{A} \underbrace{P(O_1, O_2, \cdots, O_t, I_t = q_i, I_{t+1} = q_j|\lambda)}_{B}
\end{aligned} \tag{2.8}$$

其中 $A$ 式刚好是观测概率矩阵的元素 $b_j(O_{t+1})$

$$A = P(O_{t+1}|I_{t+1} = q_j, \lambda) = \sum_{k=1}^{M} b(O_{t+1} = \nu_k|I_{t+1=q_j}, \lambda) = b_j(O_{t+1}) \tag{2.9}$$

主要来关注 $B$，我们发现 $B$ 还有关于 $t+1$ 的项，我们想要这一项只和 $t$ 及其之前的序列相关，因此对 $B$ 做继续展开

$$\begin{aligned}
B &= P(O_1, O_2, \cdots, O_t, I_t = q_i, I_{t+1} = q_j|\lambda) \\
&= P(I_{t+1} = q_j|O_1, O_2, \cdots, O_t, I_t = q_i, \lambda) \cdot P(O_1, O_2, \cdots, O_t, I_t = q_i|\lambda)
\end{aligned} \tag{2.10}$$

根据齐次马尔可夫假设

$$B = P(I_{t+1} = q_j|I_t = q_i, \lambda) \cdot P(O_1, O_2, \cdots, O_t, I_t = q_i|\lambda) \tag{2.11}$$

上式子左边第一项刚好是转移矩阵，第二项刚好是 $\alpha_t(i)$ 所以

$$B = a_{ij} \cdot \alpha_t(i) \tag{2.12}$$

然后我们把 $A$ 和 $B$ 最后的计算结果带回 $\alpha_{t+1}(j)$

$$\alpha_{t+1}(j) = b_j(O_{t+1}) \cdot a_{ij} \cdot \alpha_t(i) \tag{2.13}$$

这个递推关系有什么意义呢？我们的联合概率分布是

$$P(O|\lambda) = \sum_{i=1}^{n} P(O, I_T = q_i|\lambda) = \sum_{i=1}^{n} \alpha_T(i) \tag{2.14}$$

利用递推关系，我们可以从简单的 $\alpha_1$ 开始，用 $\alpha_1$ 推导 $\alpha_2$、$\alpha_2$ 推导 $\alpha_3 \cdots$，以此类推，最终算出来 $\alpha_T(j)$，代入求和式就是联合概率分布。而 $\alpha_1$ 是只和一个状态结点和一个观测结点相关，因此是容易确定的。

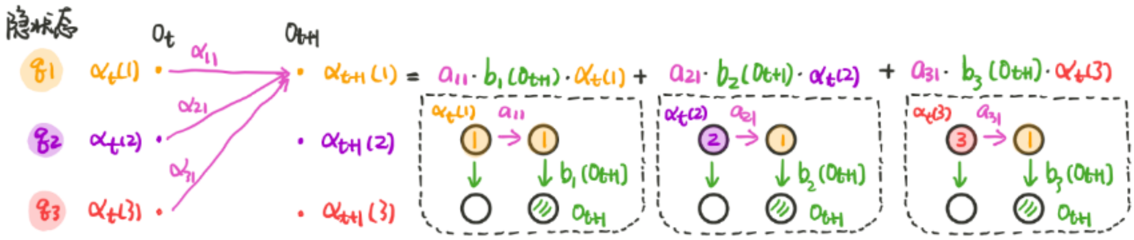


图 **2.2:** HMM 前向传播示意图

前向算法实际上是基于"状态序列的路径结构"递推计算联合概率分布的算法。其关键在于局部计算前向概率，利用路径结构将前向概率递推到全局，从而减少每一次计算直接引用前一个时刻的计算结果，避免重复计算，每次计算，隐状态的状态空间数为 $N$，序列长度为 $T$，因此这样利用前向计算算法的计算量是 $O(N^2T)$ 阶的。

## 2.3. 后向算法

后向概率的推导实际上比前向概率的理解要难一些，前向算法实际上是一个联合概率，而后向算法则是一个条件概率，所以后向的概率实际上比前向难求很多。

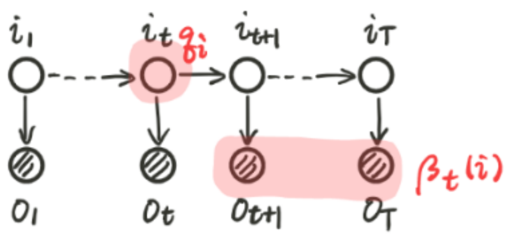


图 **2.3:** 后向算法示意图

我们设 $\beta_t(i) = P(O_{t+1}, \cdots, O_T | I_t = q_i, \lambda)$，计算观测的联合概率分布

$$
\begin{aligned}
P(O|\lambda) &= P(O_1, O_2, \cdots, O_N|\lambda) \\
&= \sum_{i=1}^{N} P(O_1, O_2, \cdots, O_N, I_1 = q_i|\lambda) \\
&= \sum_{i=1}^{N} P(O_1, O_2, \cdots, O_N|I_1 = q_i, \lambda) \cdot P(I_1 = q_i|\lambda) \\
&= \sum_{i=1}^{N} P(O_1|O_2, \cdots, O_N, I_1 = q_i|\lambda) \cdot P(O_2, \cdots, O_N, I_1 = q_i|\lambda) \cdot \pi_i \\
&= \sum_{i=1}^{N} P(O_1|I_1 = q_i, \lambda) \cdot \beta_i(i) \cdot \pi_i \\
&= \sum_{i=1}^{N} b_i(O_1) \cdot \beta_1(i) \cdot \pi_i
\end{aligned} \tag{2.15}
$$

现在我们成功找到了 $P(O|\lambda)$ 和第一个状态之间的关系，如下图所示

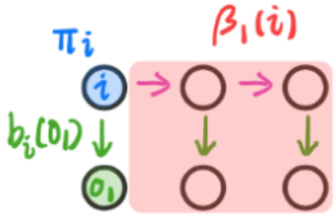


**图 2.4:** $P(O|\lambda)$ 与第一个状态的关系

现在我们要通过递推找最后一个状态和第一个状态的关系

$$
\begin{aligned}
\beta_t(i) &= P(O_{t+1}, \cdots, O_T | I_t = q_i) \\
&= \sum_{j=1}^{N} P(O_{t+1}, \cdots, O_T | I_t = q_i, i_{t+1} = q_j) \cdot P(I_{t+1} = q_j | I_t = q_i)
\end{aligned} \tag{2.16}
$$

根据概率图模型，$I_t$ 和 $t+1$ 后面所有的状态都没有关系，因此

$$
\begin{aligned}
\beta_t(i) &= \sum_{j=1}^{N} P(O_{t+1}, \cdots, O_T | I_{t+1} = q_j) \cdot a_{ij} \\
&= \sum_{j=1}^{N} P(O_{t+1}|O_{t+1}, \cdots, O_T, I_{t+1} = q_j) \cdot \underbrace{P(O_{t+2}, \cdots, O_T | I_{t+1} = q_j)}_{\beta_{t+1}(j)} \cdot a_{ij} \\
&= \sum_{j=1}^{N} b_j(O_{t+1}) \cdot \beta_{t+1}(j) \cdot a_{ij}
\end{aligned} \tag{2.17}
$$

马尔可夫链中每一个状态都是后一个状态的充分统计量，与之前的状态没有关系

$$
P(O_{t+1}, \cdots, O_T | I_{t+1} = q_j, I_t = q_i) = P(O_{t+1}, \cdots, O_T | I_{t+1} = q_j) \tag{2.18}
$$

通过这样的迭代方法从后往前推就可以得到 $\beta_1(i)$ 的概率了，从而推断 $P(O|\lambda)$。

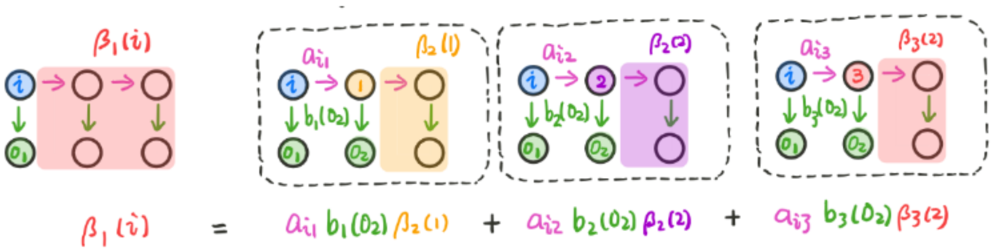


**图 2.5:** 后向传播算法的拓扑结构

# Chapter 3

# Learning 问题

在 Evaluation 问题中，我们在已知模型 $\lambda$ 的情况下，基于前向算法或者后向算法可以得出观测序列的联合概率分布 $P(O|\lambda)$。而 Learning 问题就是在已知观测数据的情况下求参数 $\lambda$。

$$\lambda_{MLE} = \arg\max_{\lambda} P(O|\lambda) \tag{3.1}$$

## 3.1. Learning

我们需要计算的目标是

$$\lambda_{MLE} = \arg\max_{\lambda} P(O|\lambda) \tag{3.2}$$

联合概率分布表示为

$$P(O|\lambda) = \sum_{I_1} \cdot \sum_{I_2} \cdots \cdot \sum_{I_T} \pi_{i_1} \prod_{t=2}^{T} a_{i_t-1,i_t} \prod_{t=2}^{T} b_{i_t}(O_t) \tag{3.3}$$

直接计算非常困难，因此考虑使用 *EM 算法*。即

$$\theta^{(t+1)} = \arg\max_{\theta} \int_z logP(X,Z|\theta) \cdot P(Z|X,\theta^{(t)})dz \tag{3.4}$$

这里 $X$ 替换为观测表量 $O$，$Z$ 替换为隐变量 $I$，$\theta$ 替换参数 $\lambda$，并且注意到情况是离散的，所以

$$Q(\lambda,\lambda^{(t)}) = \sum_{I} logP(O,I|\lambda) \cdot P(I|O,\lambda) \tag{3.5}$$

$$\lambda^{(t+1)} = \arg\max_{\lambda} Q(\lambda,\lambda^{(t)}) \tag{3.6}$$

*EM 算法*就是最大化 $Q$ 函数。代入联合分布概率 $P(O,I|\lambda)$ 有

$$Q(\lambda,\lambda^{(t)}) = \sum_{I} \left[ \left( log\pi_{i_1} + \sum_{t=2}^{T} log\ a_{i_{t-1},i_t} + \sum_{t=2}^{T} log\ b_{i_t}(O_t) \right) \cdot P(O,I|\lambda^{(t)}) \right] \tag{3.7}$$

## 3.2. Baum-Welch Algorithm

得到了 $Q$ 函数，learning 问题的下一步就是极大化这个 $Q$ 函数，求模型的参数 $\lambda = (\pi, \mathcal{A}, \mathcal{B})$。$Q$ 函数稍微做一下变化

$$Q(\lambda, \lambda^{(t)}) = (\sum_I [log\pi_{i_1}] + \sum_I \left[\sum_{t=2}^{T} log\ a_{i_{t-1},i_t}\right] + \sum_I \left[\sum_{t=2}^{T} log\ b_{i_t}(O_t)\right]) \cdot P(O, I|\lambda^{(t)}) \tag{3.8}$$

可以发现 $\lambda$ 的三个参量 $\pi, \mathcal{A}, \mathcal{B}$ 分别都是独立的，我们可以分开求解。以 $\pi^{(t+1)}$ 为例

$$\pi^{(t+1)} = \arg\max \sum_I \left[log\pi_{i_1} \cdot P(O, I|\lambda^{(t)})\right] \tag{3.9}$$

下面来计算这个式子

$$\pi^{(t+1)} = \arg\max \sum_{i_1} \cdots \sum_{i_T} log\ \pi_{i_1} \cdot P(O, i_1, \cdots, i_T|\lambda^{(t)}) \tag{3.10}$$

联合概率分布的求和可以得到边缘概率，所以

$$\pi^{(t+1)} = \arg\max \sum_{i_1} log\ \pi_{i_1} \cdot P(O, i_1 = q_i|\lambda^{(t)}) \tag{3.11}$$

同时还有约束条件

$$\sum_{i=1}^{N} \pi_i = 1 \tag{3.12}$$

那么自然根据拉格朗日乘子法，构造损失函数

$$\mathcal{L}(\pi, \eta) = \sum_{i_1} log\ \pi_{i_1} \cdot P(O, i_1 = q_i|\lambda^{(t)}) + \eta \left(\sum_{i=1}^{N} \pi_i - 1\right) \tag{3.13}$$

求解该拉格朗日函数的极值可以得出

$$\pi_i^{(t+1)} = \frac{P(O, i_1 = q_i|\lambda^{(t)})}{P(O|\lambda^{(t)})} \tag{3.14}$$

这样我们就能推出 $\pi^{(t+1)} = (\pi_1^{(t+1)}, \cdots, \pi_N^{(t+1)})$，对于 $\mathcal{A}^{(t+1)}$ 和 $\mathcal{B}^{(t+1)}$ 也是同样的过程，因此不在赘述。

**算法 10.4（Baum-Welch 算法）**

输入：观测数据 $O=(o_1,o_2,\cdots,o_T)$；

输出：隐马尔可夫模型参数。

（1）初始化。对 $n=0$，选取 $a_{ij}^{(0)}$，$b_j(k)^{(0)}$，$\pi_i^{(0)}$，得到模型 $\lambda^{(0)}=(A^{(0)},B^{(0)},\pi^{(0)})$。

（2）递推。对 $n=1,2,\cdots,$

$$a_{ij}^{(n+1)}=\frac{\sum\limits_{t=1}^{T-1}\xi_t(i,j)}{\sum\limits_{t=1}^{T-1}\gamma_t(i)}、$$

$$b_j(k)^{(n+1)}=\frac{\sum\limits_{t=1,o_t=v_k}^{T}\gamma_t(j)}{\sum\limits_{t=1}^{T}\gamma_t(j)}$$

$$\pi_i^{(n+1)}=\gamma_1(i)$$

右端各值按观测 $O=(o_1,o_2,\cdots,o_T)$ 和模型 $\lambda^{(n)}=(A^{(n)},B^{(n)},\pi^{(n)})$ 计算。式中 $\gamma_t(i)$，$\xi_t(i,j)$ 由式 (10.24) 和式 (10.26) 给出。

（3）终止。得到模型参数 $\lambda^{(n+1)}=(A^{(n+1)},B^{(n+1)},\pi^{(n+1)})$。 ■

# Chapter 4

# Decoding 问题

*Decoding* 问题就是在给定观测序列的情况下，寻找最大概率可能出现的隐概率状态序列，即

$$\hat{I} = \arg\max_{I} P(I|O,\lambda) \tag{4.1}$$

## 4.1. Decoding

首先画出 HMM 拓扑模型

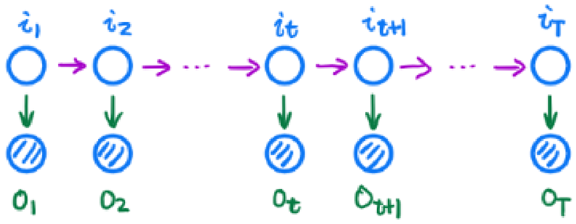


图 **4.1:** Hidden Markov Model 拓扑模型

我们要看 $o_t$ 的的概率分别是多少时 $i_t$ 的概率最大，这里实际上就是一个动态规划问题，只不过将平时的最大距离问题等价于最大概率问题，每个时刻都有 $N$ 个状态，从 $N^T$ 个可能的序列中找出概率最大的排列

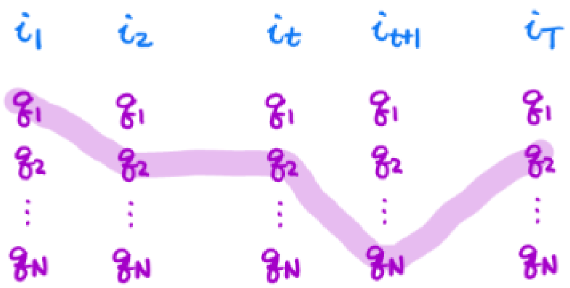


图 4.2: Hidden Markov Model Decoding 的动态规划

## 4.2. Viterbi Algorithm

我们假设

$$\delta_t(i) = \max_{i_1,\cdots,i_{t-1}} P(o_1,\cdots,o_t,i_1,\cdots,i_{t-1},i_t=q_i) \tag{4.2}$$

这个等式的意思是，假设前面 $t-1$ 个时刻随便走，第 $t$ 个时刻走到 $q_i$，从中选取概率最大的序列，我们下一步的目标就是在知道 $\delta_t(i)$ 的情况下如何求 $\delta_{t+1}(i)$

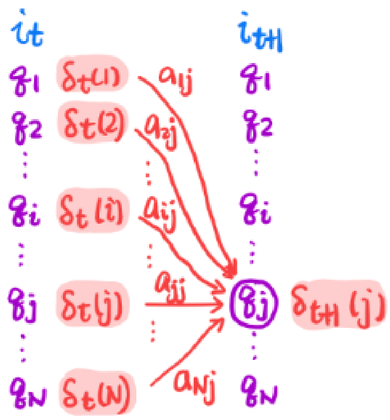


图 4.3: Viterbi 算法示意图

$$\begin{aligned}\delta_{t+1}(j) &= \max_{i_1,\cdots,i_T} P(o_1,\cdots,o_{t+1},i_1,\cdots,i_t,i_{t+1}=q_j) \\ &= \max_{i_1,\cdots,i_T} \delta_t(i)\cdot a_{ij}\cdot b_j(o_{t+1})\end{aligned} \tag{4.3}$$

这就是 Viterbi 算法，但是这个算法最后求的是一个值，但没有办法求路径，如果想要求路径，需要引入一个变量

$$\varphi_{t+1}(j) = \arg\max_{1\leqslant t\leqslant N} \delta_t(i)\cdot a_{ij}\cdot b_j(o_{t+1}) \tag{4.4}$$

这个函数用来记录每一次迭代过程中经过的状态 index，最终得到

$$\{\varphi_1,\varphi_2,\cdots,\varphi_T\} \tag{4.5}$$

# Chapter 5

## Summary

# References

[1] I. Surname, I. Surname, and I. Surname. “The Title of the Article”. In: *The Title of the Journal* 1.2 (2000), pp. 123–456.

# Chapter A

## Source Code Example

*Adding source code to your report/thesis is supported with the package* listings. *An example can be found below. Files can be added using* \lstinputlisting[language=<language>]{<filename>}.

```python
"""
ISA Calculator: import the function, specify the height and it will return a
list in the following format: [Temperature,Density,Pressure,Speed of Sound].
Note that there is no check to see if the maximum altitude is reached.
"""

import math
g0 = 9.80665
R = 287.0
layer1 = [0, 288.15, 101325.0]
alt = [0,11000,20000,32000,47000,51000,71000,86000]
a = [-.0065,0,.0010,.0028,0,-.0028,-.0020]

def atmosphere(h):
    for i in range(0,len(alt)-1):
        if h >= alt[i]:
            layer0 = layer1[:]
            layer1[0] = min(h,alt[i+1])
            if a[i] != 0:
                layer1[1] = layer0[1] + a[i]*(layer1[0]-layer0[0])
                layer1[2] = layer0[2] * (layer1[1]/layer0[1])**(-g0/(a[i]*R))
            else:
                layer1[2] = layer0[2]*math.exp((-g0/(R*layer1[1]))*(layer1[0]-layer0[0]))
    return [layer1[1],layer1[2]/(R*layer1[1]),layer1[2],math.sqrt(1.4*R*layer1[1])]
```

# Chapter B

## Task Division Example

*If a task division is required, a simple template can be found below for convenience. Feel free to use, adapt or completely remove.*

表 **B.1:** Distribution of the workload

| | Task | Student Name(s) |
|---|---|---|
| | Summary | |
| Chapter 1 | Introduction | |
| Chapter 2 | | |
| Chapter 3 | | |
| Chapter * | | |
| Chapter * | Conclusion | |
| | Editors | |
| | CAD and Figures | |
| | Document Design and Layout | |