

본 챕터에서는 지금까지의 챕터에서 설명한 각종 기법을 취합하여 대어휘 연속 음성 인식 (large vocabulary continuous speech recognition; LVCSR) 엔진을 구성하는 방법에 대해 설명한다.

0.1 FST의 합성과 확률모델

지금까지 도입했던 음성인식의 요소를 나타내는 FST를 합성하고, 합성한 FST의 최단 경로 문제의 풀이로서 음성인식결과를 얻는 방법을 고찰한다. 음성인식의 통계모델을 표현하는 FST는 일반적으로는 아래의 4 종류가 있다.

- \mathbf{G} : 단어열 Acceptor (6.5)
- \mathbf{L} : 문맥에 의존하지 않는 음소열로부터 단어열 변환 (4.2.2)
- \mathbf{C} : 문맥에 의존하는 음소열로부터 문맥에 의존하지 않는 음소열로 변환 (5.3)
- \mathbf{H} : HMM state 시퀀스로부터 문맥에 의존하는 음소열로 변환 (5.3)

여기에 덧붙여서, 이하의 식에서 보여지고 있는 입력 (관측 벡터열)과 HMM 상태변수의 관계를 나타내는 FST \mathbf{E} 를 가상으로 도입함으로써, 인식할 때의 처리를 모두 FST 형식으로 기술할 수 있게 된다.

$$\begin{aligned} Q[\mathbf{E}] &= \{0, \dots, T\}, & I[\mathbf{E}] &= \{(0, \bar{1})\}, & F[\mathbf{E}] &= \{(T, \bar{1})\}, \\ E[\mathbf{E}] &= \{(t-1, t, \sigma, \sigma, -\log p(\mathbf{x}_t | s_t = \sigma)) : t \in 1, \dots, T, \sigma \in \Sigma[\mathbf{H}]\} \end{aligned} \quad (1)$$

여기에서 T 는 관측 벡터열의 길이를 의미한다.

음성인식 전체의 변환처리, 이른바 입력 프레임 시퀀스에서 단어열로의 변환은, 이 모든 FST를 합성한 $\mathbf{E} \circ \mathbf{H} \circ \mathbf{C} \circ \mathbf{L} \circ \mathbf{G}$ 로 나타낼 수 있다. 또한 그 FST의 변환 결과로써 가장 가중치가 작아지는 가설은 최단경로문제를 푸는 것으로써 근사적으로 구할 수 있다. 5.1에서 설명한 바와 같이, 이를 Viterbi 디코딩이라 한다. 그러나 많은 경우에, 이 FST는 거대하여, Viterbi 디코딩과 같은 근사적 해법으로도 풀기가 어렵다. 대어휘 연속 음성인식기술은, 이런 거대한 FST 위에서, Viterbi 디코딩을 가능케하는 기술이라 할 수 있다.

0.1.1 디코딩 네트워크의 구성과 탐색오류

입력에 의존하는 \mathbf{E} 를 제외한 나머지 모두를 합성한 FST가 음성인식에서 사용되는데, 이 FST를 디코딩 네트워크라고 부르고, \mathbf{D} 로 나타낸다.

$$\mathbf{D} = \mathbf{H} \circ \mathbf{C} \circ \mathbf{L} \circ \mathbf{G} \quad (2)$$

위의 식에서 \mathbf{H} , \mathbf{C} , \mathbf{L} , \mathbf{G} 는 각각 출력분포를 제외한 음향모델로, 음소 문맥 클러스터링, 발음모델, 언어모델에 해당한다고 할 수 있다. 디코딩 네트워크는 입력에 대해서는 변하지 않으면서, 합성과 최적화를 다시 계산해 두는 것이 가능하다. 음성인식은 출력분포를 나타내는 FST \mathbf{E} 와 디코딩 네트워크 \mathbf{D} 의 합성 FST를 계산해가면서, 그 최단 경로를 탐색하는 문제로 볼 수 있다. 따라서 등가성을 유지한

채로 D 에 대해 가능한한 작고 간단하게 바꿔 표현하는 것은 탐색효율 향상을 위해 중요하다.

대어휘 연속 음성인식에서는 디코딩 네트워크가 크기 때문에 엄밀한 최단 경로 탐색을 수행하는 것은 가능하지 않다. 뒤에서 언급할 빔 탐색 알고리즘은 최단 경로 문제의 근사적 해법이지만, 이런 알고리즘에는 근사 정확성과 계산량 사이에 트레이드오프가 존재한다. 디코딩 네트워크를 최적화하고 최단 경로 문제의 규모를 축소함으로써, 같은 계산량이더라도 보다 높은 정확성을 갖는 근사 탐색을 수행할 수 있으며, 음성인식의 정확도 형상에도 기여한다.

최단 경로 탐색의 근사에 기인하는 오류를 탐색 오류 (search error)라고 부른다. 탐색 오류는 계산량의 제약을 무시하고 근사 정확도를 최대로 높이는 경우의 음성인식 결과와, 실제 응용 상의 계산 효율을 고려한 근사를 이용한 경우의 음성인식 결과를 비교하여 평가할 수 있다. 그림 7.1에서 탐색 알고리즘의 근사 정도를 나타내는 파라미터 (빔폭; 자세히는 뒤에 설명)과 단어오류율의 관계를 나타낸다. 빔폭을 충분히 크게 설정하게 되면 탐색오류는 줄어들게 되고, 모델 자체에 의한 오류만 남게 된다. 따라서 실제로 사용되는 빔폭을 설정했을 때의 오류율과 모델 자체의 오류율을 비교함으로써 탐색처리의 근사에 의해 생기는 오류 정도를 평가할 수 있게 된다.

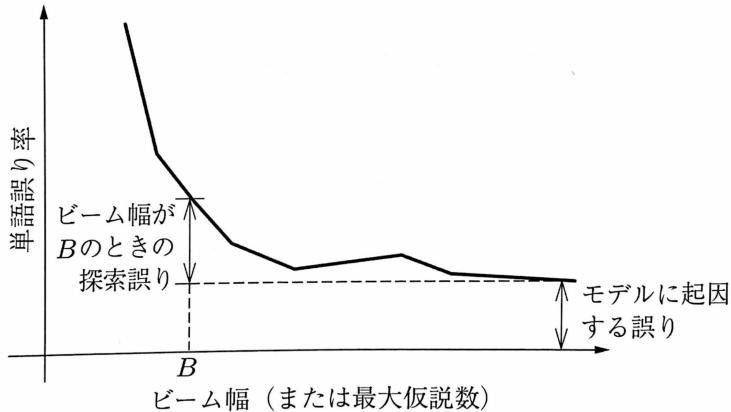


Figure 1: 탐색오류와 빔폭의 관계

디코딩 네트워크를 효율적인 표현으로 바꿈으로써, 탐색 알고리즘의 효율을 향상시켜서, 결과적으로 평가에 있어서 같은 계산 자원으로 보다 적은 탐색오류를 달성할 수 있게 된다.

0.1.2 disambiguation 심볼

대어휘 연속 음성인식에 있어서의 FST의 최적화에서는 결정화 (와 그 전처리로써 필요한 ϵ 제거) 및 최소화가 이루어진다. FST의 최적화 (Opt라고 한다)는 예를 들어 아래와 같은 것으로, 3개의 FST 변환 함수에 의해 나타낼 수 있다.

$$\text{Opt}(\mathbf{X}) \stackrel{\text{def}}{=} \text{Min}(\text{Det}(\text{RmEps}(\mathbf{X}))) \quad (3)$$

여기서 $RmEps$ 는 ϵ 제거 (3.6.2), Det 는 결정화 (3.6.4), Min 은 최소화 (3.6.5)를 의미한다.

이와 같은 최적화 연산에 의해서 등가 교환이 가능한 FST 가운데 가장 작은 결정성 FST를 얻을 수는 있으나, 결정성 FST는 잘 설계된 비결정성 FST에 비해 상태수가 많은 경우도 있고, 디코딩 네트워크 전체를 결정성 FST로 표현하는 것이 어렵다.¹ 게다가 디코딩 네트워크를 구성하는 각 요소의 결정화와 최소화를 수행함으로써 최종적인 디코딩 네트워크를 쉽게 유지하는 것을 생각할 수 있다.

결정화는 같은 접두사를 갖는 상태를 공유하는 효과가 있기 때문에, 특히 L 과 C 에 있어서는 상태의 갯수를 줄이는 효과를 기대할 수 있다. 실용적인 상태 갯수의 결정성 FST를 얻을 수 있다면, 최소화 처리에 의해 상태 갯수를 더욱 줄이는 것이 가능하다. 그러나 결정화가 가능하다는 것은 함수형 FST, 이른바 어떤 입력 시퀀스에 대해 단 하나의 출력 시퀀스만 가능한 FST로 한정되는데, 예를 들어 L 은 동음이의어를 갖는 경우, 이 전제를 만족할 수 없게 된다. 이 때 도입할 수 있는 것이 disambiguation 심볼이다.

赤い	a k a i	#1
赤井	a k a i	#2
青い	a o i	#1
葵	a o i	#2
青井	a o i	#3
良い	i i	
良い	y o i	
池	i k e	

Figure 2: disambiguation 심볼을 적용한 레시콘

그림 7.2에서는 disambiguation 심볼을 도입한 레시콘의 예를 보여주고 있다. disambiguation 심볼은 그림에서는 #1과 #2가 있는데, 이는 레시콘 내의 동음이의어를 구별하기 위해 필요한 만큼 준비할 수 있다. 그림에서의 레시콘의 경우, /a o i/에 대응하는 단어가 가장 많으며 그 수는 3개이기 때문에 #1부터 #3까지 disambiguation 심볼로써 추가할 수 있다. 이런 심볼을 도입함으로써 레시콘 FST를 결정화할 수 있다.

H 나 C 와 같은 다른 FST에도 disambiguation 심볼을 적용해 볼 수 있다면 레시콘 FST를 합성한 FST (예를 들어 $C \circ L$ 이나 $H \circ (C \circ L)$ 등)도 결정화 할 수 있게 되고, 합성 후의 표현을 최적화도 할 수 있게 된다. disambiguation 심볼을 H 나 C 에 적용하는 가장 단순한 방법으로써, H 나 C 의 모든 상태에 disambiguation 심볼을 입출력하는 상태 전이를 추가하는 방법을 소개한다. disambiguation 심볼의 집합을 Γ 로 하고, H 와 C 의 상태 전이 집합을 $E' = E \cup \{(q, d, d, q, \bar{1}) : q \in Q, d \in \Gamma\}$ 와

¹예를 들어 6.5의 방법에 의해 백오프나 보간 구조를 사용하여 구성한 N그램 언어모델의 FST를 결정화하는 것은 상태 갯수의 조합이 폭발적으로 늘어나기 때문에 많은 경우에 어렵다.

같이 확장하여, \mathbf{H}' 과 \mathbf{C}' 을 얻을 수 있다. 이를 이용한 합성 FST, 즉, $\mathbf{H}' \circ (\mathbf{C}' \circ \mathbf{L})$ 과 $\mathbf{C}' \circ \mathbf{L}$ 은 \mathbf{L} 의 입력 심볼로써 추가된 disambiguation 심볼을 입력 알파벳의 일부로 갖는 FST가 된다. disambiguation 심볼은 인식할 때에는 무시될 필요가 있으므로 최적화 이후에는 이를 ϵ 전이로 변환하여 필요하다면 이어서 ϵ 제거를 실행한다.

아래는 디코딩 네트워크의 구성한 한 예를 최적화 연산을 포함하여 기술하였다.

$$\mathbf{D} = \text{RmDisamb}(\text{Opt}(\mathbf{H}' \circ \text{Opt}(\mathbf{C}' \circ \text{Opt}(\mathbf{L}')))) \circ \mathbf{G} \quad (4)$$

여기에서 \mathbf{H}' , \mathbf{C}' , \mathbf{L}' 은 disambiguation 심볼이 추가된 \mathbf{H} , \mathbf{C} , \mathbf{L} 이며 RmDisamb는 disambiguation 심볼을 제거한 연산을 의미한다.

이런 구성법은 다음의 순서로 진행된다. 우선 렙시콘 FST \mathbf{L}' 을 단독으로 최적화시킨 후, 음소 문맥 FST \mathbf{C}' 을 합성하여 다시 한번 최적화한다. 그 후에 HMM 상태전이 FST \mathbf{H}' 을 합성하여 최적화하고, 다음으로 disambiguation 심볼을 제거한 언어모델 FST \mathbf{G} 와 합성한다. 이 뿐만 아니라 조합 방법은 여러가지를 생각해 볼 수 있고, 탐색 알고리즘의 성질과 실행하는 계산 특성, 각 모델의 특징을 고려하여 결정하여야 한다.

0.2 대어휘 연속 음성인식의 탐색문제

입력 \mathbf{X} 를 식 7.1과 같이 FST \mathbf{E} 로써 적용할 경우, 음성인식은 아래와 같이 나타낼 수 있다.

$$\hat{\mathbf{y}} = \underset{\mathbf{y}}{\operatorname{argmax}} p(\mathbf{X}|\mathbf{y})p(\mathbf{y}) = \underset{\mathbf{y}}{\operatorname{argmin}} \llbracket \mathbf{E} \circ \mathbf{D} \circ \mathbf{y} \rrbracket \quad (5)$$

이와 같은 최적화는 어렵기 때문에 실제로 사용할 때는 5.1에서 언급한 Viterbi 디코딩을 응용하게 되는데, FST의 최단경로에 대응하는 출력 레이블을 이용하여 아래와 같은 최적 출력을 근사한다.

$$\hat{\mathbf{y}} \simeq \delta(\hat{\pi}) \quad (6)$$

여기에서 δ 는 경로로부터 출력 알파벳 시퀀스를 꺼내는 함수를 의미하고, 최적 경로 $\hat{\pi}$ 는 격자 상의 최단 경로로써 아래와 같이 정의된다.

$$\hat{\pi} \stackrel{\text{def}}{=} \underset{\pi \in \prod[\mathbf{E} \circ \mathbf{D}]}{\operatorname{argmin}} \bar{\omega}(\pi) \quad (7)$$

여기에서 $\bar{\omega}(\pi)$ 는 경로 π 에 대한 가중치이다. 3.5절에서 최단경로문제에 대해 설명한 바와 같이, FST의 가중치는 Tropical Semiring이 된다.

격자 $\mathbf{E} \circ \mathbf{D}$ 가 거대하기 때문에, 이 최단경로문제의 엄밀하게 정확한 답은 얻기란 현실적으로 어렵다. 최단경로를 근사적으로 구하기 위해 시간 동기 범 탐색 (time-synchronous beam search)라고 불리는 방법을 사용할 수 있다. 범 탐색은 폭 우선 탐색 (breadth-first search)의 폭을 제한하여 탐색을 빠르게 하는 근사법이다. 또한 ”시간 동기”라는 말은 탐색 가설이 각 입력 레이블 (각각 하나의 시간에 대응)로 그룹화하여, 그에 대해 폭을 제한하는 것과 같은 가지치기 (Pruning)을 수행하는 것을 나타낸다.

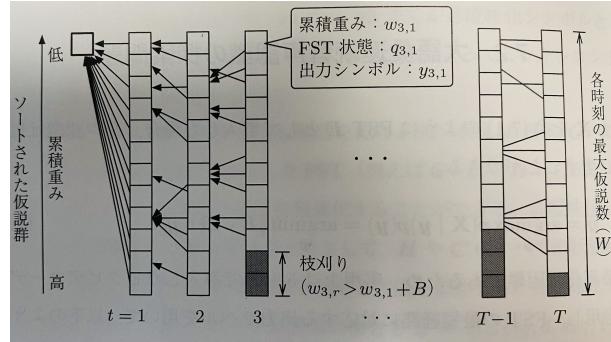


Figure 3: 시간 동기 빔 탐색의 데이터 구조

그림 7.3에서는 Beam 탐색에 이용되는 데이터 구조의 한 예를 보여주고 있다².

격자 FST의 state 변수는 $Q[\mathbf{E}] \times Q[\mathbf{D}] = \{0, \dots, T\} \times Q[\mathbf{D}]$ 이고, 각 state 는 시간 t 와 디코딩 네트워크 상의 state 변수 $q \in Q[\mathbf{D}]$ 의 쌍 $(t, q) \in Q[\mathbf{E} \circ \mathbf{D}]$ 으로 표현된다. 시간 동기 탐색에서는 격자 FST의 state 를 시간 t 마다 그룹화한다. 본래의 최단경로의 탐색문제에서는 초기 state로부터 특정 state (t, q) 에 이르는 모든 경로의 가중치의 \oplus 연산을 통해 전방향 스코어를 계산 (3.5절 참조) 하지만, 큰 규모에서는 이러한 계산이 불가능하기 때문에 근사치로써 그룹마다 거기에 이르기까지의 가중치가 가장 작은 몇몇 state만을 이용하여 다른 상태까지 가는 경로를 무시 (가지치기) 하여 전방향 스코어를 계산해가는 방법을 취한다. 이를 빔 탐색이라고 부른다.

시간 t 마다 디코딩 네트워크의 state q , 전방향 스코어 ω' , 출력 심볼 시퀀스 \mathbf{y} 와 백 포인터 b 의 4개 원소를 갖는 $(q, \omega', \mathbf{y}, b) \in (Q[\mathbf{D}] \times \mathbb{K} \times \Delta^* \times \mathbb{N})$ 의 배열 $H[t]$ 가 ω' 가 작은 순으로 정렬되어 있다고 하자. 백 포인터 b 는 시간 $t-1$ 에 디코딩 네트워크의 어떤 state에 이르고 있는가를 가설로써 배열 $H[t-1]$ 상에서 그 위치를 이용하여 자연수로 나타낸다.

가설 배열 $H[t-1]$ 의 각 요소 $H[t-1][r] = (q, \omega', \mathbf{y}, b)$ 에 대해서 그 지점에서부터 오는 state 변이 $e \in E^{\text{succ}}(q)$ 를 열거하고 그것을 이용하여 $q'[e]$ 로 전이하는 경우를 고려한다. $\delta[e] \neq \epsilon$ 이 되는 state 전이를 하는 경우, 시간 t 의 음향 특징 벡터 \mathbf{x}_t 의 확률이 HMM state $\delta[e]$ 에 대응하는 출력분포로 평가하고, 그 음의 대수확률을 가중치로써 상태 $(t, q'[e])$ 에 이른다. 따라서 가설 배열 $H[t]$ 에 아래와 같은 요소를 덧붙일 수 있다.

$$(q'[e], \omega' + \omega[e] - \log p(\mathbf{x}_t | s_t = \sigma[e]), \{\delta[e]\}, r) \quad (8)$$

$\sigma[e] = \epsilon$ 의 경우에는 입력을 처리하지 않고 디코딩 네트워크 상에서의 state를 변화시킬 필요가 있다. 격자상에서는 $(t-1, q)$ 로부터 $(t-1, q'[e])$ 로의 state 전이로 대응한다. 이를 실현하는 간단한 방법은 ϵ 이 나타날 때마다 현재 생각하고 있는 state 변이 집합을 확장하는 방법이다. 즉, state q 로부터 나오는

²음성인식의 탐색 알고리즘은 음성인식 이용시의 계산 비용에 가장 영향을 주는 부분이어서, 계산 고속화와 근사 계산이 많이 이용되고 있다. 또한 음성인식의 결과를 출력하는 부분이기 때문에, 그 용용에 있어서 요구되는 기능에 따라 확장되는 경우도 많다. 본 절에서 설명하는 것은 가장 간단한 구현 방법이고, 현실 시스템에 있어서 복잡한 디코더의 구현과는 괴리가 있다는 점에 주의해 주길 바란다.

0.3 대규모 FST 합성 기술

0.3.1 온 더 플라이 합성

0.3.2 디스크 기반 인식 시스템

0.4 N-Best 리스트 및 lattice 생성

0.4.1 lattice 생성

0.4.2 lattice로부터 N-Best 리스트 생성

인용 및 참고문헌