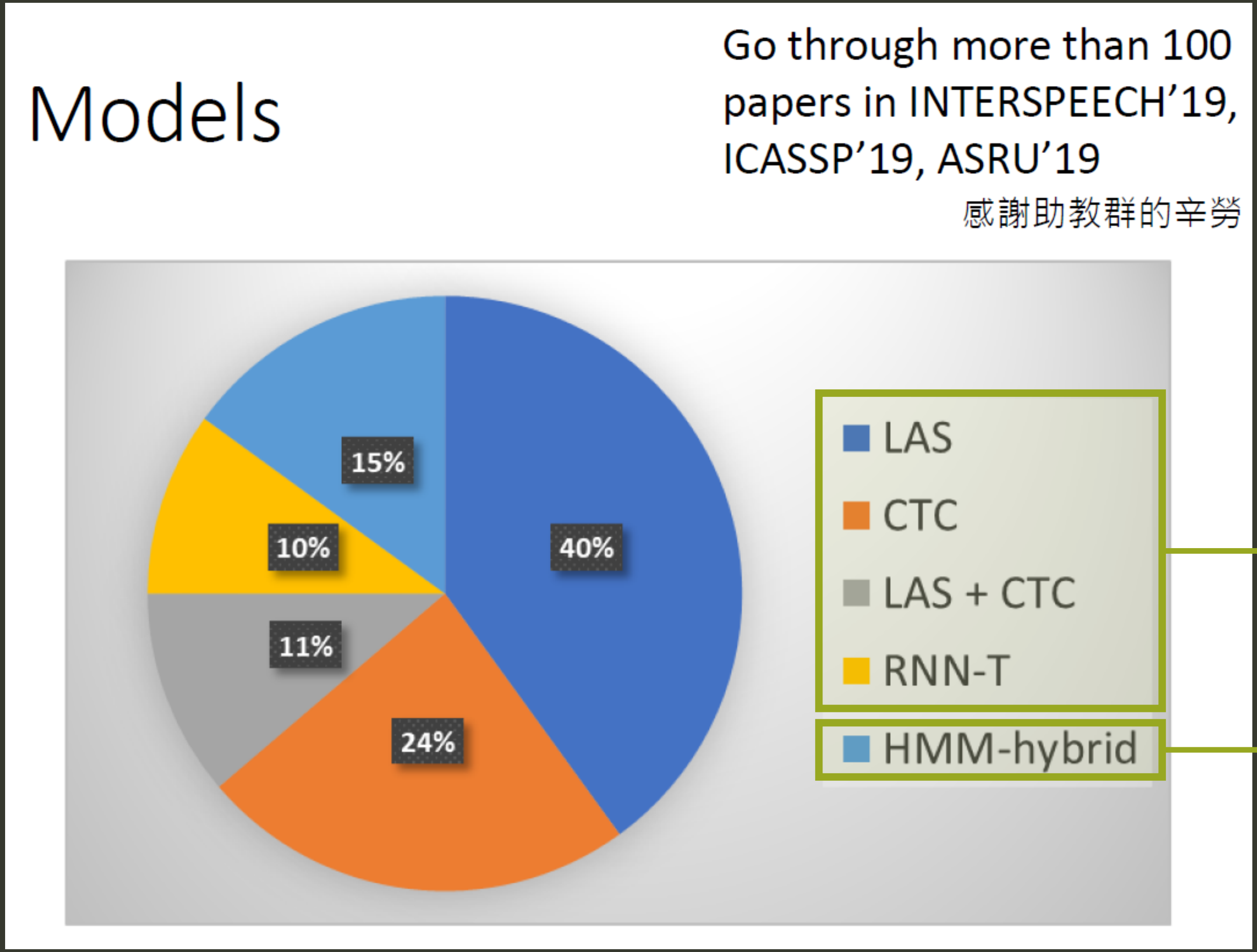
CTC Model and Loss

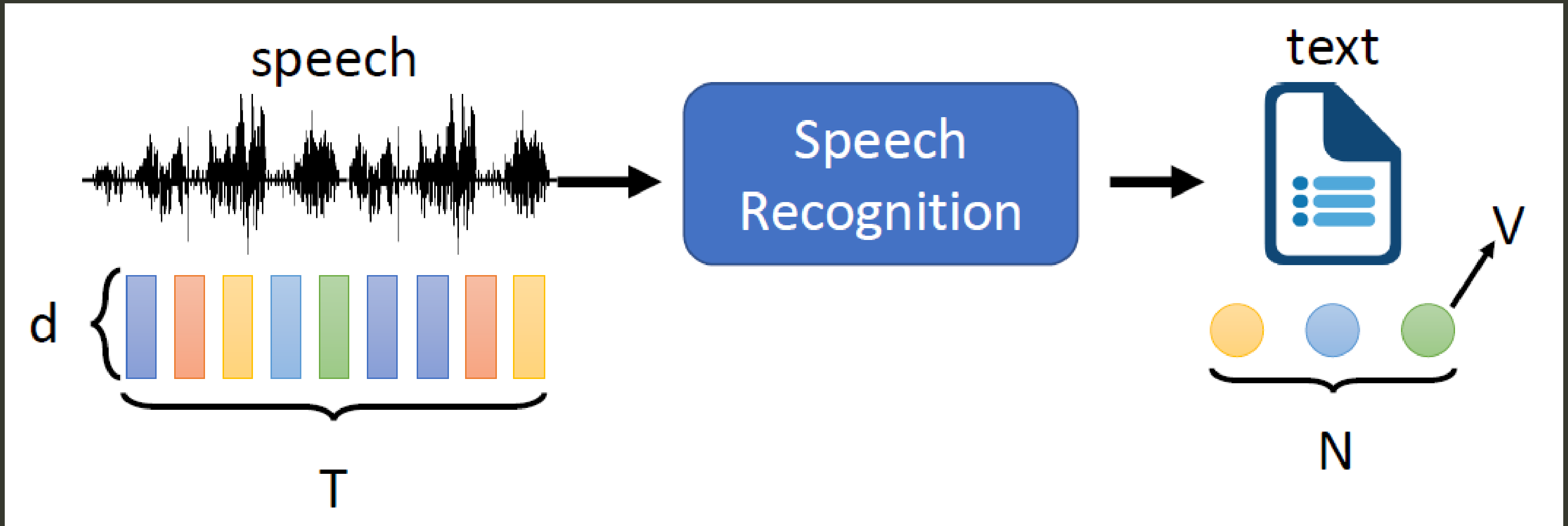
棒棒生

ASR 19’ Interspeech [Hungyi Lee]



End-to-end System

Kaldi

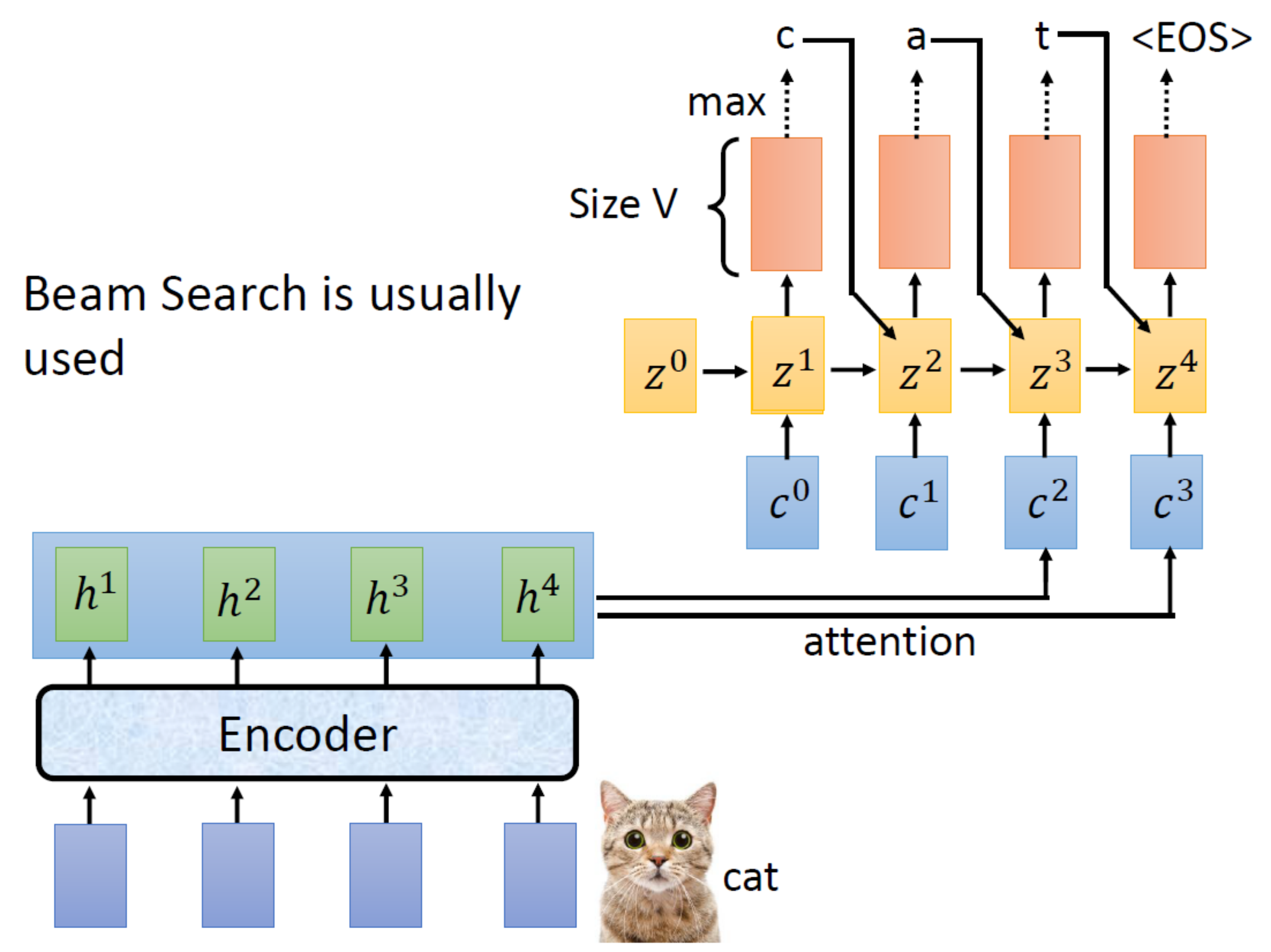


- Token 單位:
  - Phoneme
  - Grapheme: 書寫最小單位, 英文為{a,...,z}+{空白,...}, 中文為 {所有字}
  - Word ("詞"): Degas Lexicon, 英文~200K, 中文~700K
  - Morpheme: smallest meaningful unit (<word, >grapheme)

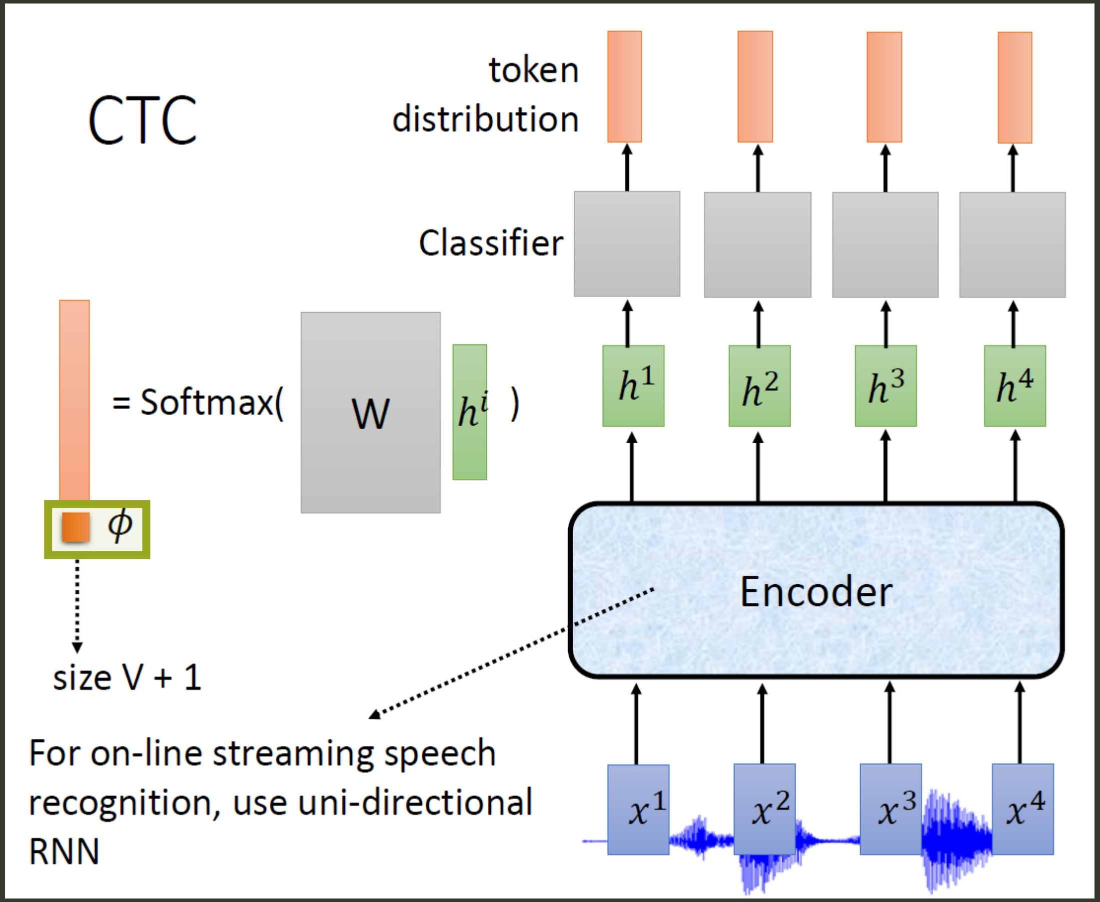
- End2End ASR 就是吃 feature vectors 直接吐出 Token vectors
  - 沒有 lexicon, 沒有 wfst
  - Training, decoding 比起 hybrid-system 單純很多 (就一個大的NN)

Listen, Attend Spell

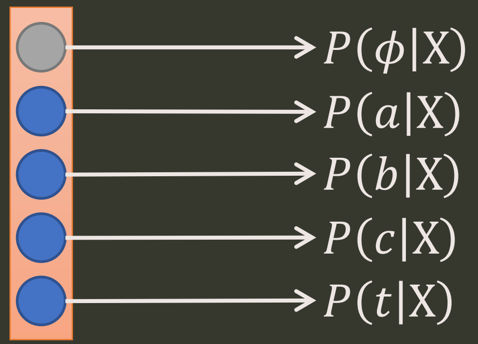
Beam Search is usually used



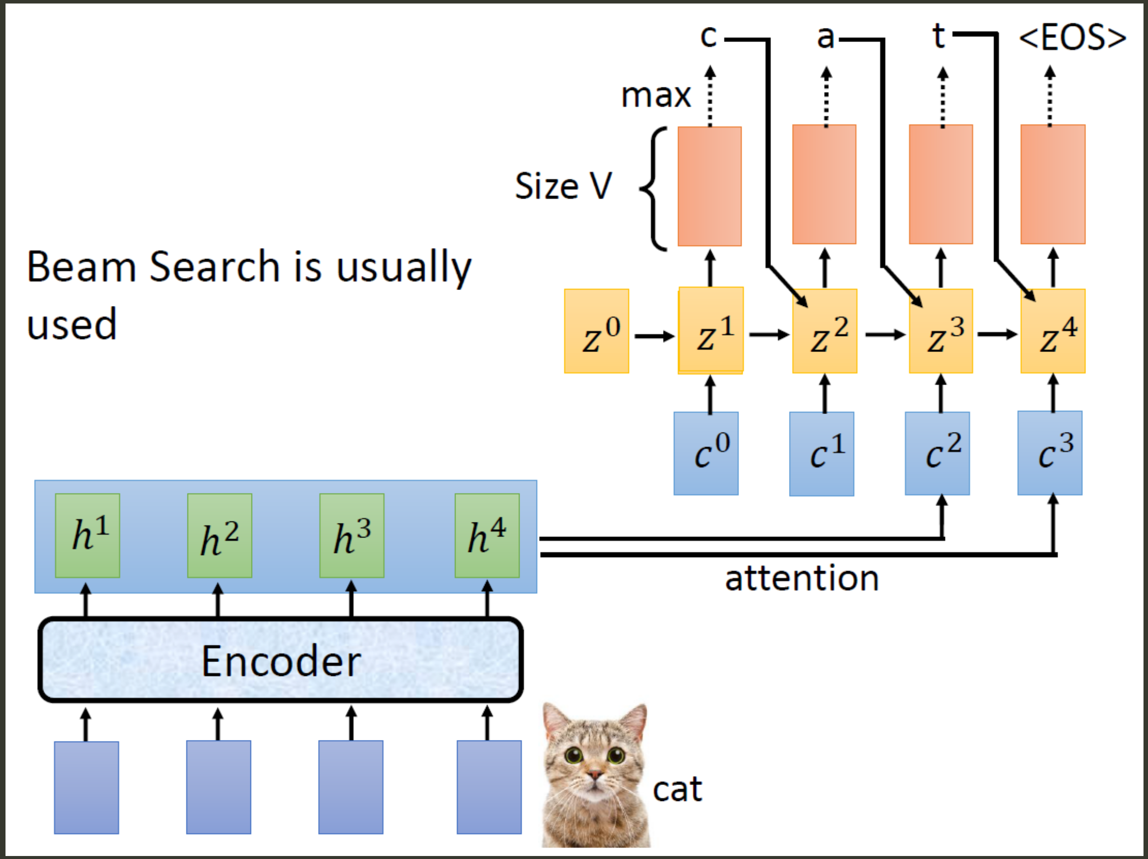
Model of CTC

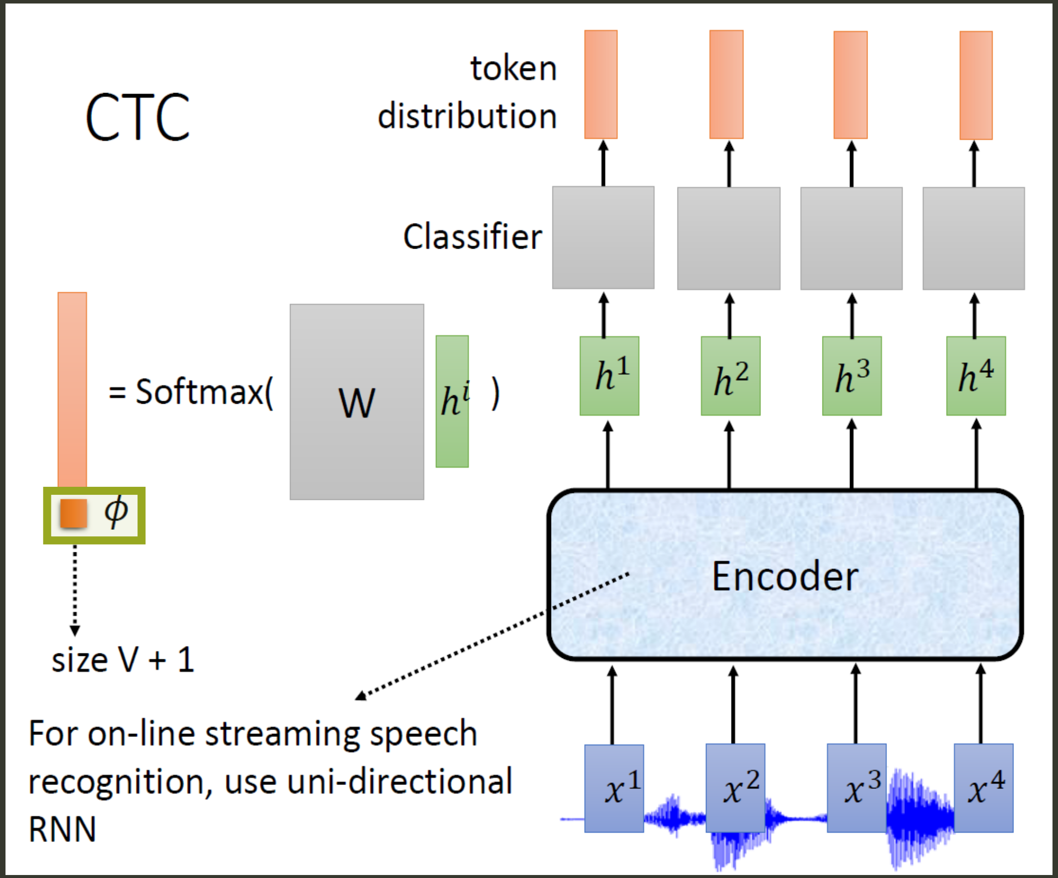


- ϕ 特殊 token, 代表 null (paper 稱 blank)
  - 不是每個時間點的 feature vector 都能有意義的解釋
- Encoder 每個時間點 t 都會 output token distribution
- Token distribution:

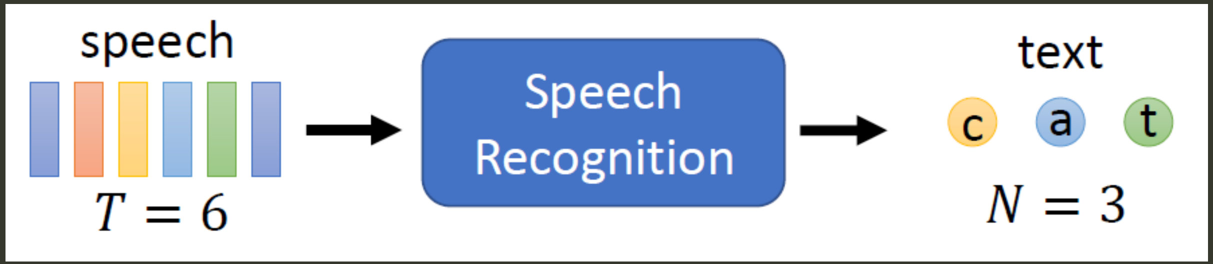


- LAS
  - Can NOT online
  - NO alignment issue
- CTC (RNN-T)
  - CAN online
  - Having alignment issue





Alignment 解釋方式 (HMM)



Align	Frame 1	Frame 2	Frame 3	Frame 4	Frame 5	Frame 6	Valid?
π_1	c	c	c	a	t	t	✓
π_2	c	a	a	a	a	t	✓
π_3	c	c	c	c	t	t	✗
⋮							

Alignment 解釋方式 (CTC)

1. Merging duplicate tokens
2. Removing ϕ

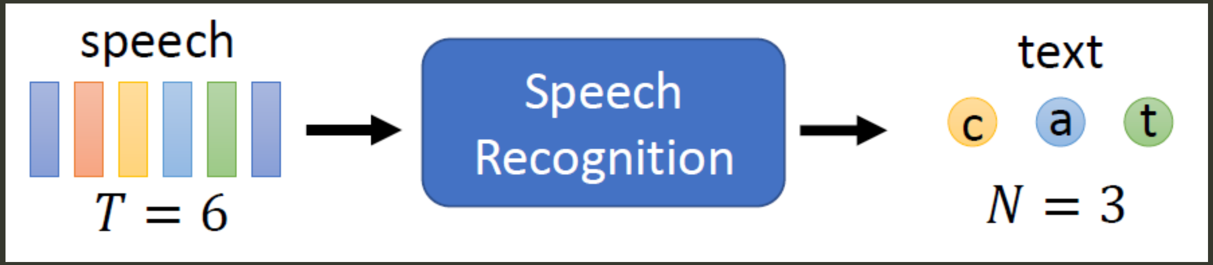
ϕ ϕ d d ϕ e ϕ e ϕ p p → d e e p

ϕ ϕ d d ϕ e e e ϕ p p → d e p

好 好 棒 棒 棒 棒 棒 → 好 棒

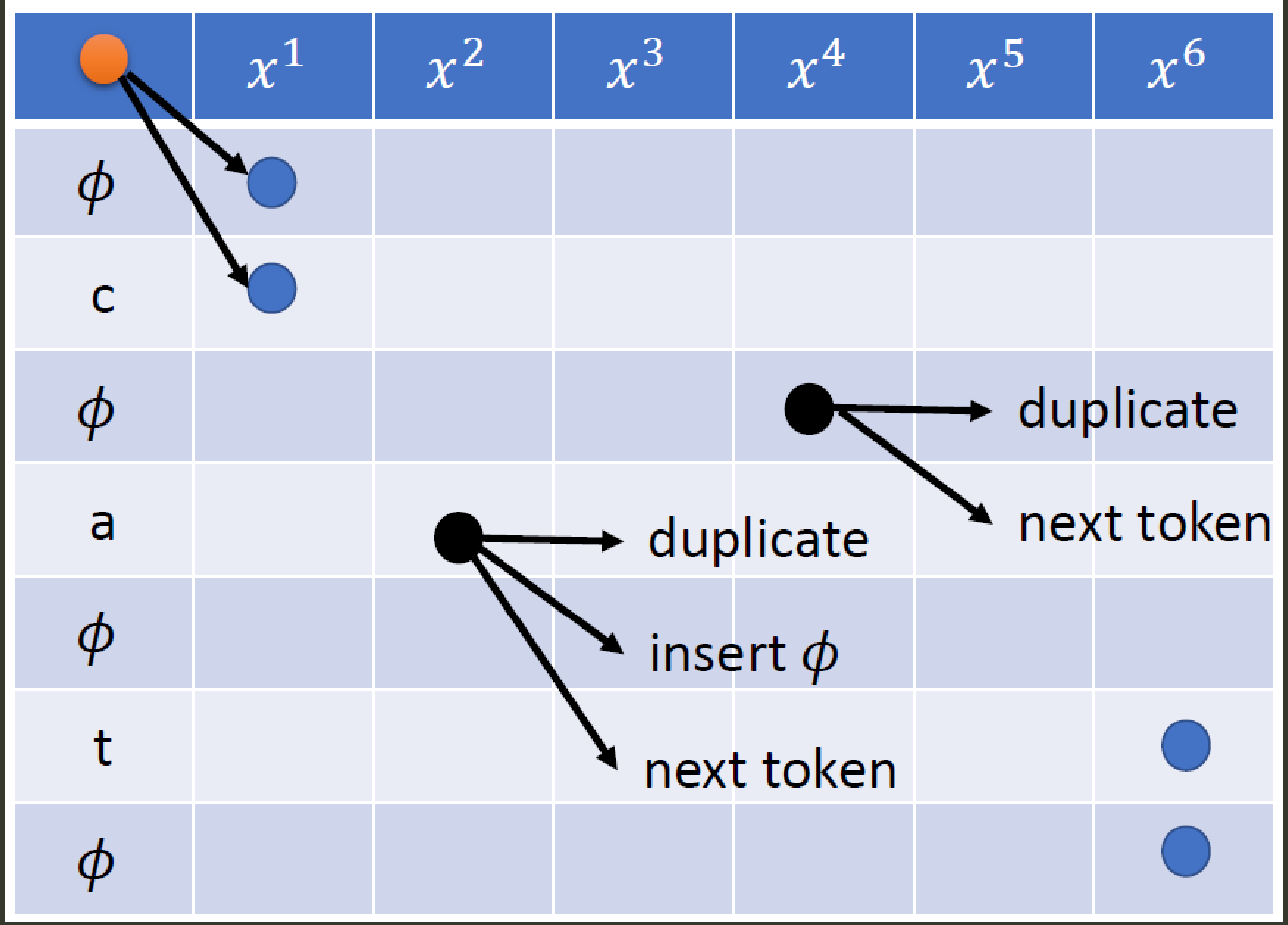
好 ϕ 棒 ϕ 棒 ϕ ϕ → 好 棒 棒

Alignment 解釋方式 (CTC)

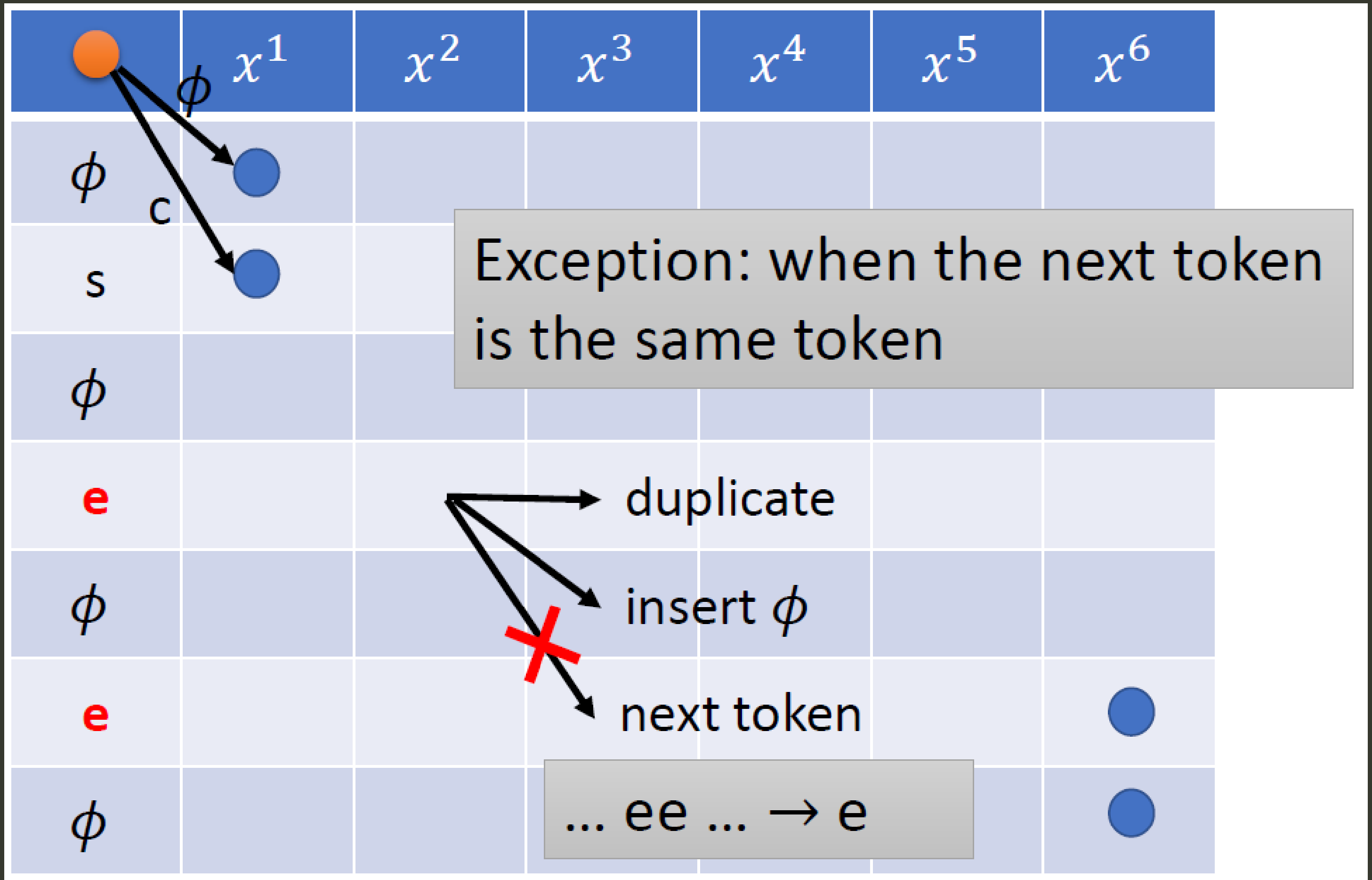


Align	Frame 1	Frame 2	Frame 3	Frame 4	Frame 5	Frame 6	Valid?	註解
π_1	c	c	c	a	t	t	✓	可以完全沒有ϕ
π_2	c	ϕ	ϕ	a	t	t	✓	ϕ一樣可以重複
π_3	ϕ	c	a	a	t	ϕ	✓	開頭結尾可以ϕ
π_4	ϕ	ϕ	ϕ	c	a	t	✓	可以最後暴衝一波
π_5	c	ϕ	c	a	t	t	✗	ccat
π_5	ϕ	ϕ	c	c	t	t	✗	ct

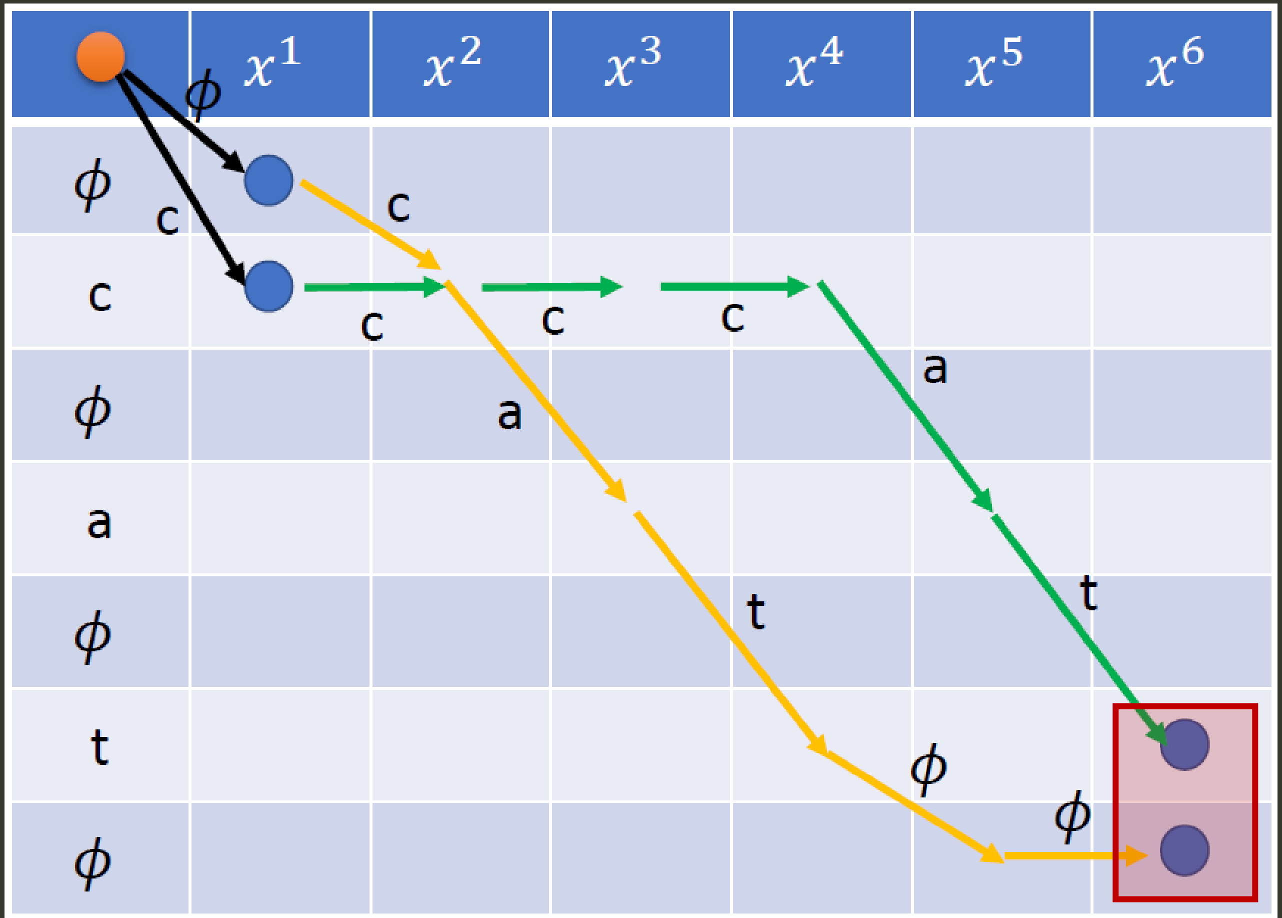
如何走完所有 Alignment? 遊戲規則



如何走完所有 Alignment? (一個特殊情形)



一條 path π 就是一個合法 alignment



$$P(\text{cat}|\mathrm{X}) = \sum_{\text{合法的}\pi} P(\pi|\mathrm{X})$$

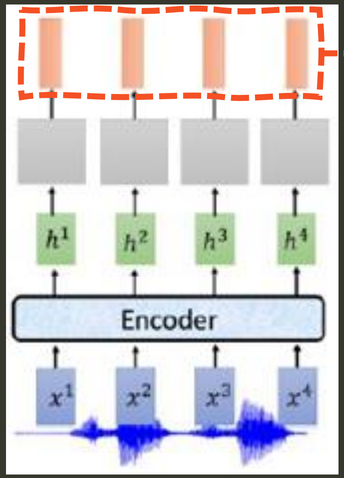
一條 Alignment π 的機率怎麼算?

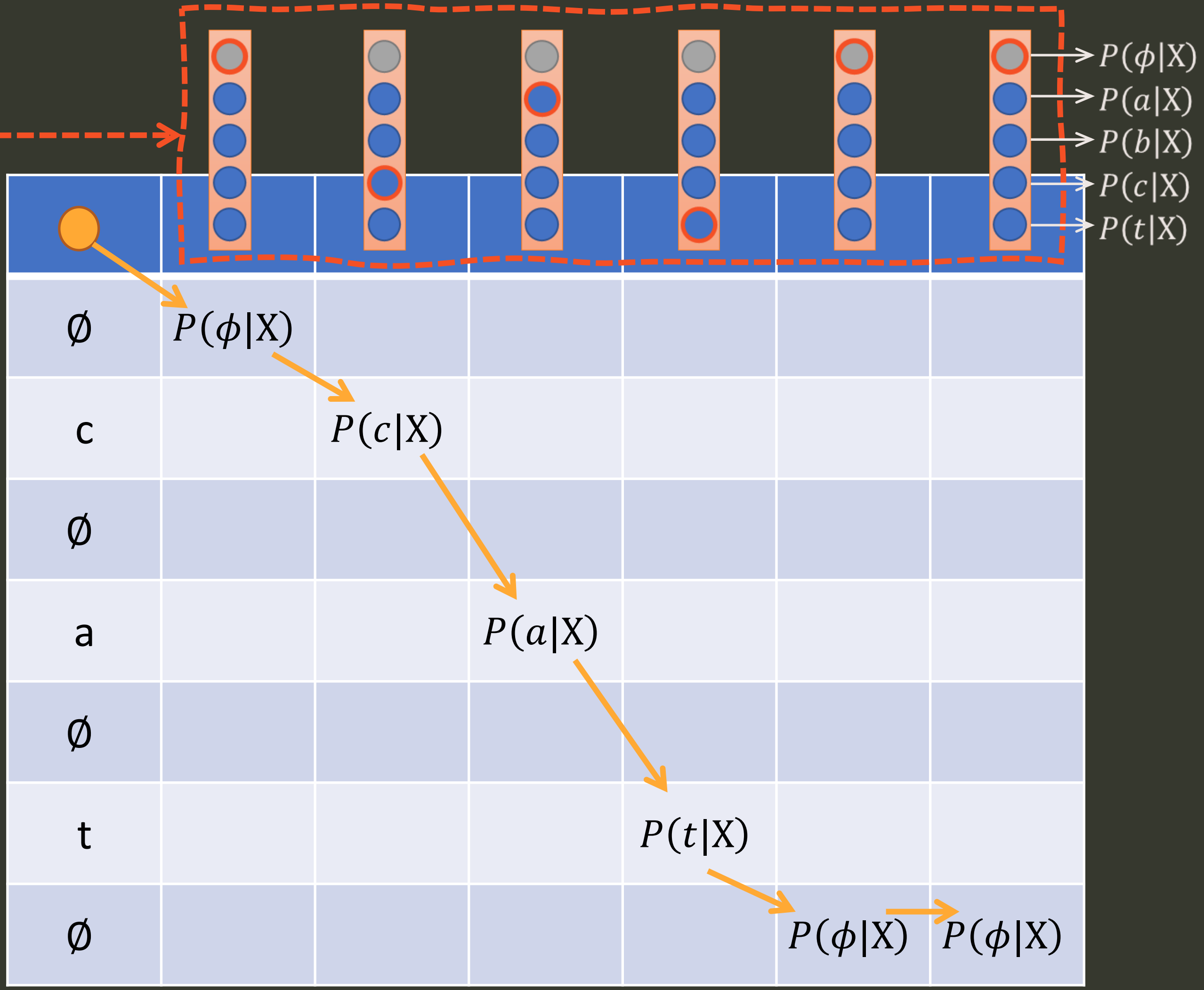
- $\pi = \{\pi_1, \pi_2, \pi_3, \pi_4, \dots, \pi_T\}$
- $P(\pi|\mathrm{X}) = P(\pi_1|\mathrm{X})P(\pi_2|\pi_1, \mathrm{X})P(\pi_3|\pi_1, \pi_2, \mathrm{X})P(\pi_4|\pi_1, \pi_2, \pi_3, \mathrm{X})\cdots$
- $P(\pi|\mathrm{X}) = P(\pi_1|\mathrm{X})P(\pi_2|\cancel{\pi_1}, \mathrm{X})P(\pi_3|\cancel{\pi_1, \pi_2}, \mathrm{X})P(\pi_4|\cancel{\pi_1, \pi_2, \pi_3}, \mathrm{X})\cdots$

CTC indep. assumption

- $P(\pi|\mathrm{X}) = P(\pi_1|\mathrm{X})P(\pi_2|\mathrm{X})P(\pi_3|\mathrm{X})P(\pi_4|\mathrm{X})\cdots$
- Example: $P(\phi cat\phi\phi|\mathrm{X}) = P(\phi|\mathrm{X})P(c|\mathrm{X})P(a|\mathrm{X})P(t|\mathrm{X})P(\phi|\mathrm{X})P(\phi|\mathrm{X})$

$$P(\phi cat\phi\phi|X) = P(\phi|X)P(c|X)P(a|X)P(t|X)P(\phi|X)P(\phi|X)$$





怎麼計算$P(cat|\mathrm{X})$? Forward/Backward DP

用一個實際例子舉例 (符號會有點不同)

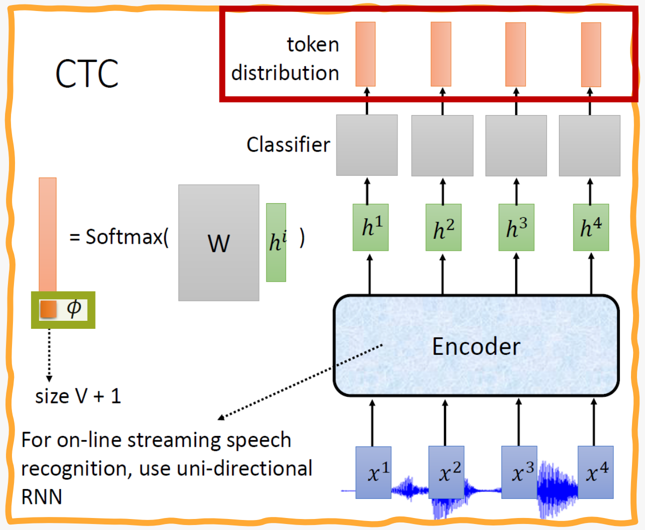
一些變數定義

1. `Vocab = [0, 1, 2, 3, 4]`, 其中 **0 表示 'blank'**.
2. `V = len(Vocab) = 5` 是字典大小, 譬如字典有 4 個 labels 加上一個 blank, 因此 V=5
3. `l` 是正確答案 (已經安插 blanks 了), 例如 `l = [0, 3, 0, 3, 0, 4, 0]`
4. `L = len(l) = 7`
5. **後驗概率** `y` (shape= `[V,T]`), 其中 `T` 表示 input sequence 長度, 所以 `y[k,t]` 表示時間點 `t` , label `k` 的後驗概率

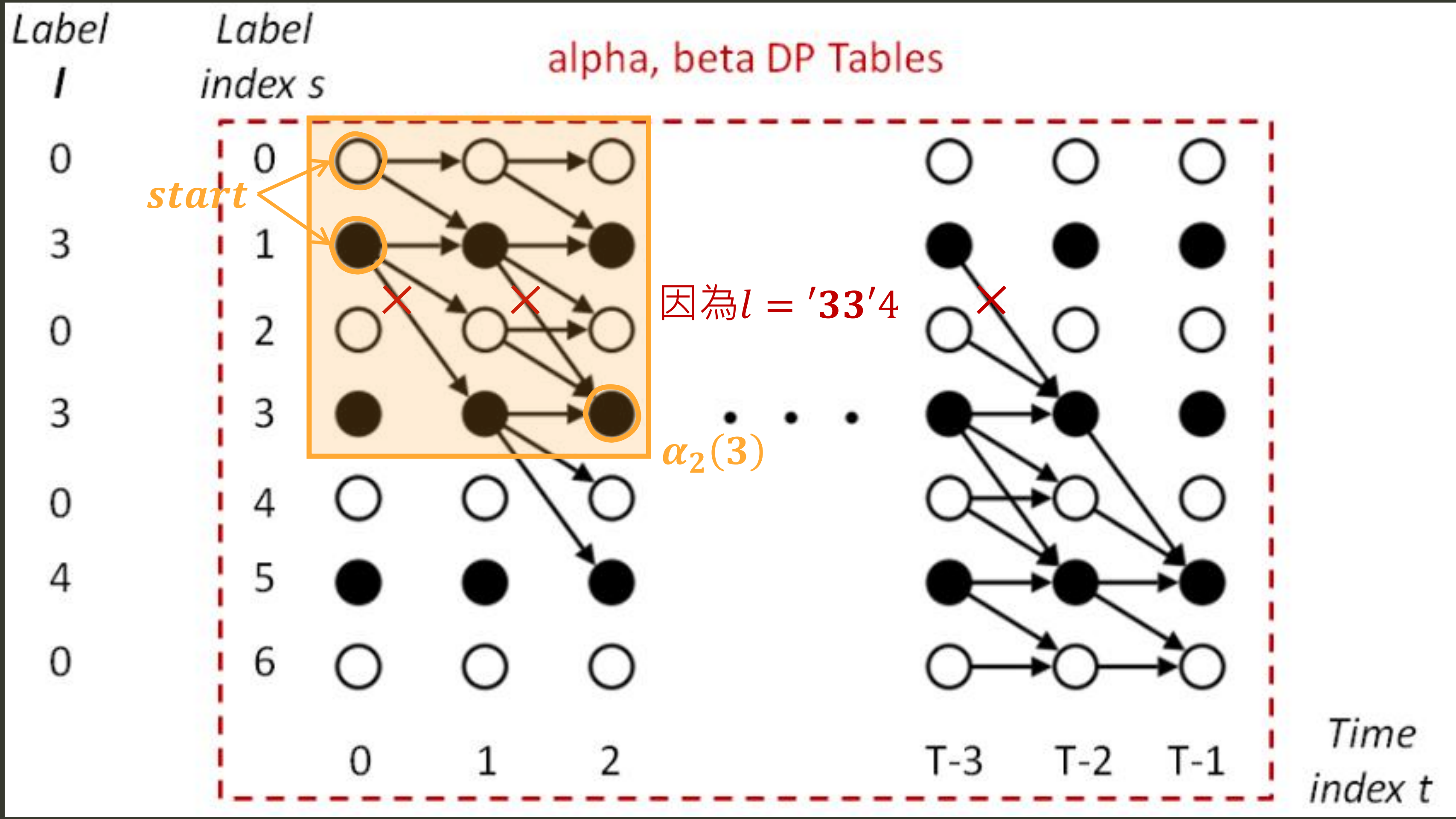
```python
Vocab = [0,1,2,3,4]
l = [0, 3, 0, 3, 0, 4, 0]
V, L = len(Vocab), len(l)
T = 12
logits = np.random.random([V,T])

def softmax(logits):
    max_value = np.max(logits, axis=0, keepdims=True)
    exp = np.exp(logits - max_value)
    exp_sum = np.sum(exp, axis=0, keepdims=True)
    dist = exp / exp_sum
    return dist

y = softmax(logits)
```

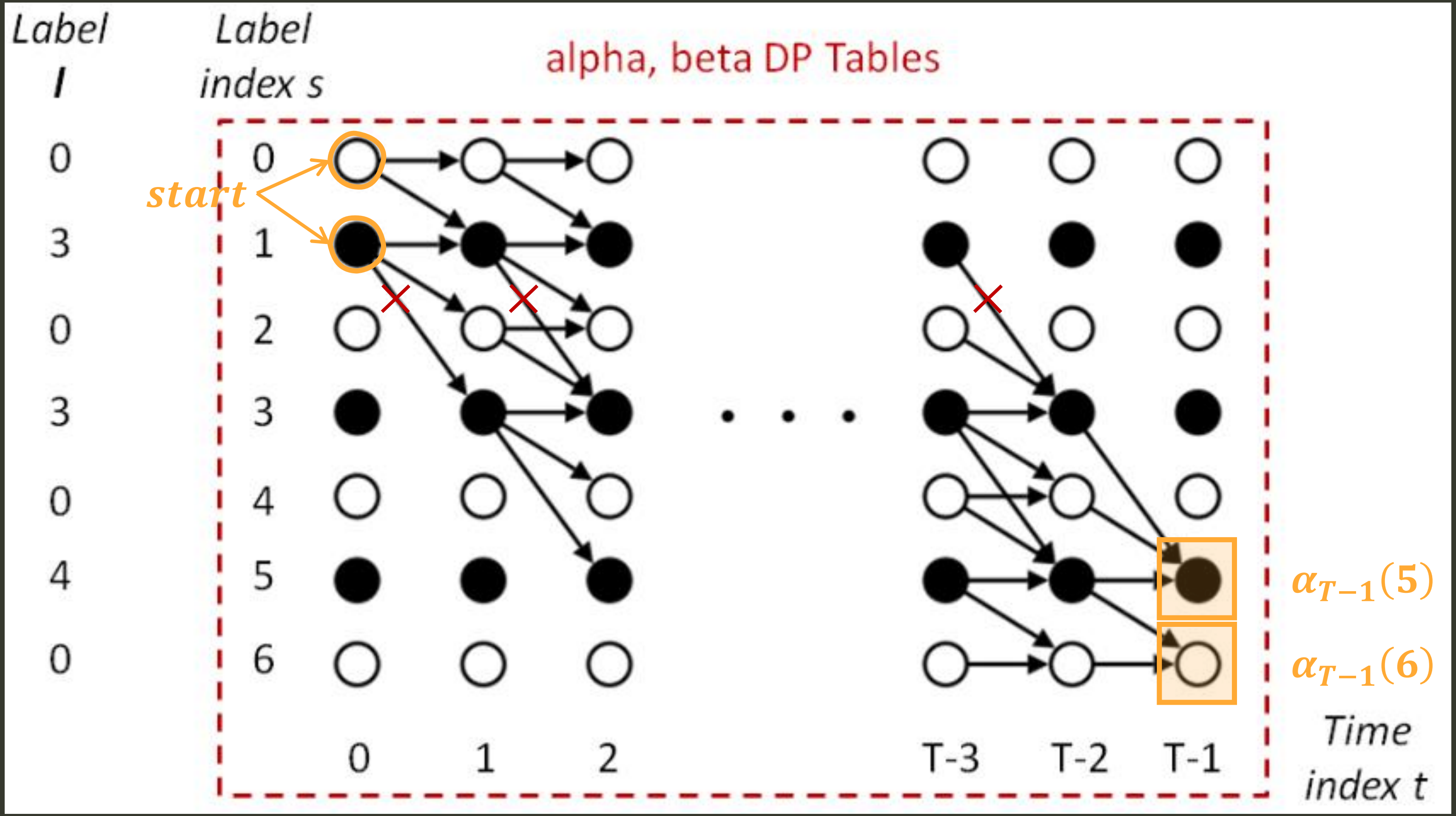


$\alpha_t(s)$



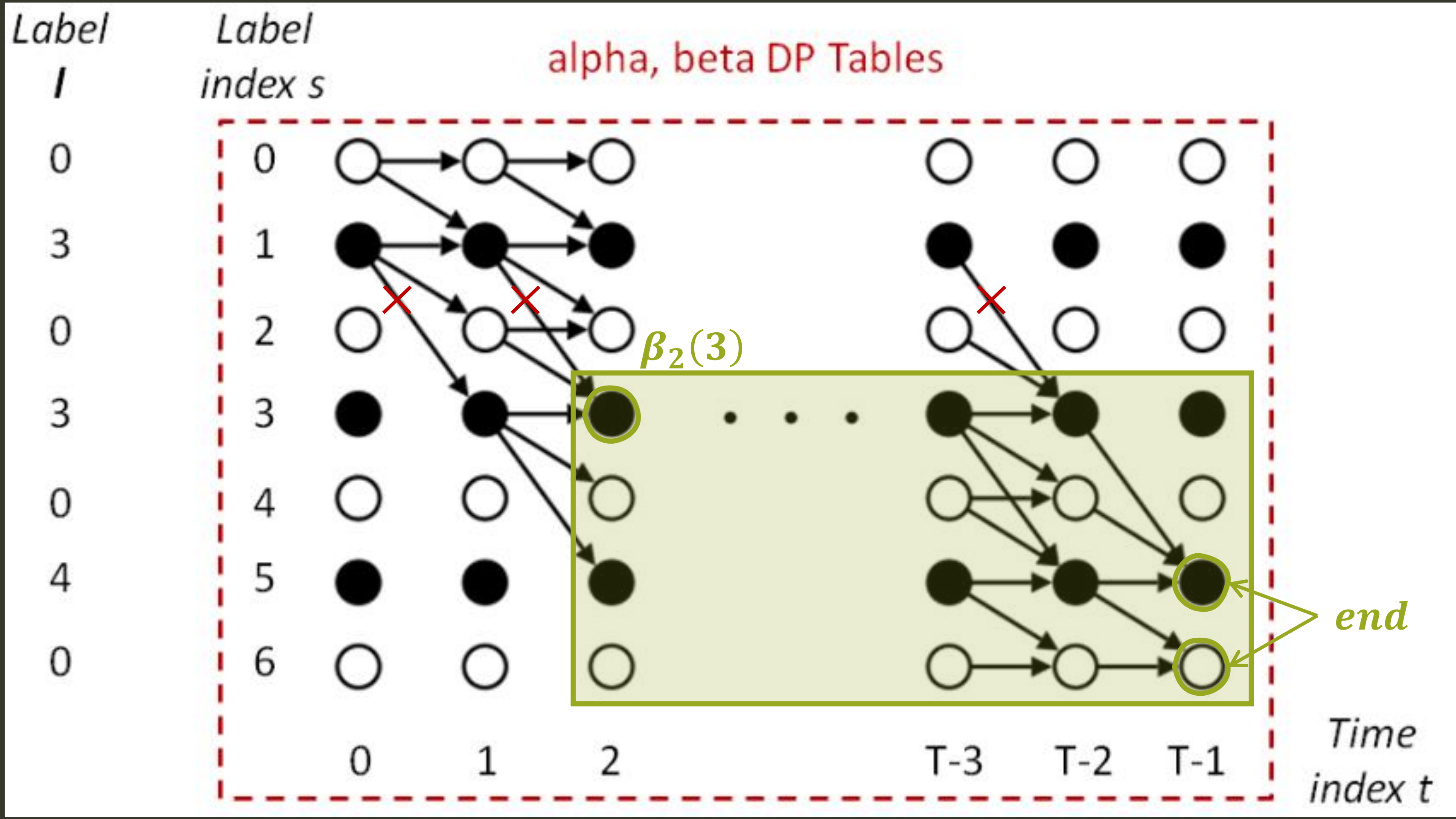
$\alpha_2(3)$ 表示所走到 $(t = 2, s = 3)$ 路徑的機率總合

$\alpha_t(s)$



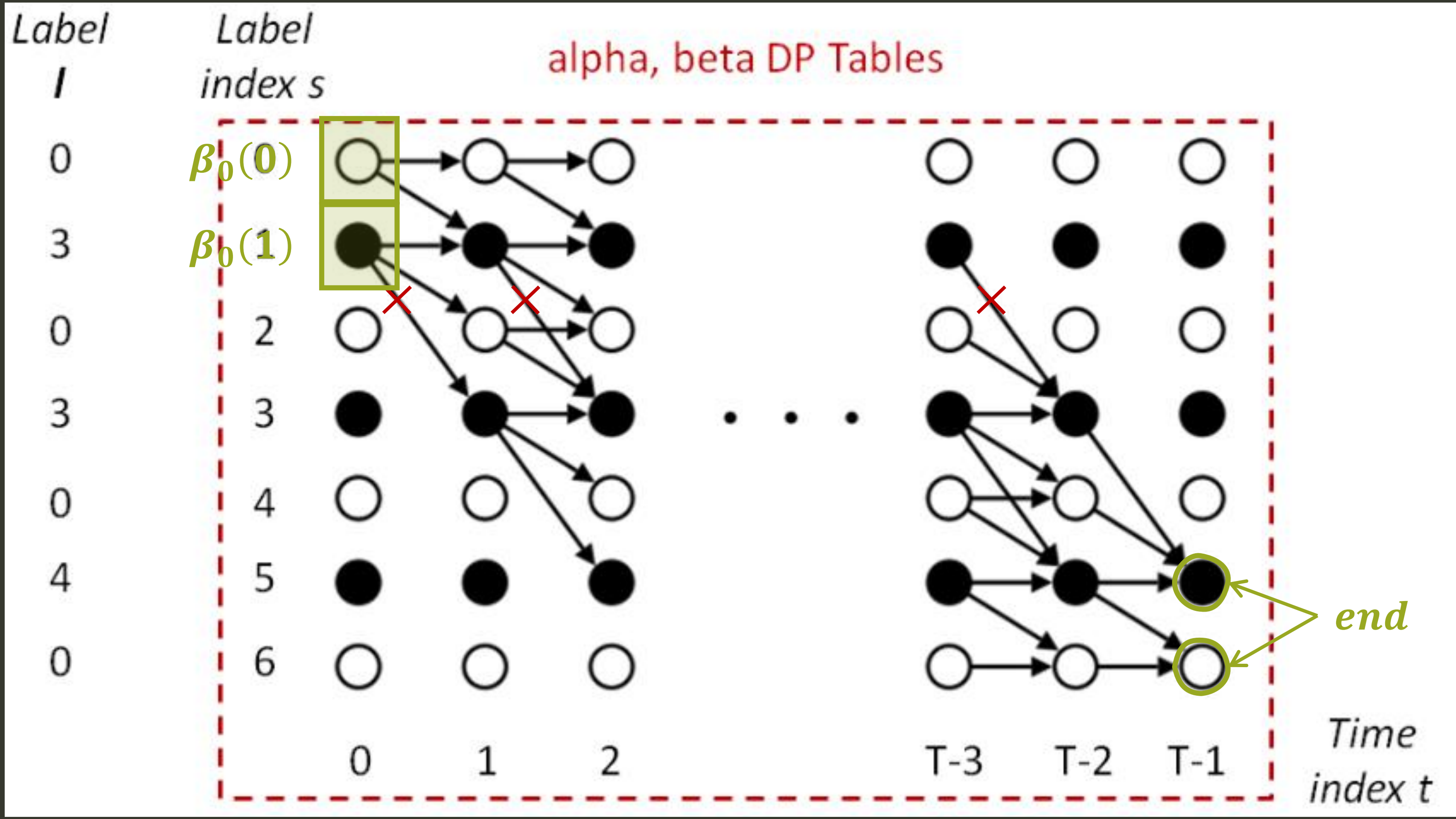
$$P(334|X) = \alpha_{T-1}(5) + \alpha_{T-1}(6)$$

$\beta_t(s)$



$\beta_2(3)$ 表示從 $(t = 2, s = 3)$ 開始到路徑結束的機率總合

$\beta_t(s)$



$$P(334|X) = \beta_0(0) + \beta_0(1)$$

Forward

1. `Vocab = [0, 1, 2, 3, 4]`, 其中 **0 表示 'blank'**.
2. `V = len(Vocab) = 5` 是字典大小, 譬如字典有 4 個 labels 加上一個 blank, 因此 V=5
3. `l` 是正確答案 (已經安插 blanks 了), 例如 `l = [0, 3, 0, 3, 0, 4, 0]`
4. `L = len(l) = 7`
5. **後驗概率 `y`** (shape= `[V,T]`), 其中 `T` 表示 input sequence 長度, 所以 `y[k,t]` 表示時間點 `t` , label `k` 的後驗概率

```
Vocab = [0,1,2,3,4]
l = [0, 3, 0, 3, 0, 4, 0]
V, L = len(Vocab), len(l)
T = 12
logits = np.random.random([V,T])

def softmax(logits):
    max_value = np.max(logits, axis=0, keepdims=True)
    exp = np.exp(logits - max_value)
    exp_sum = np.sum(exp, axis=0, keepdims=True)
    dist = exp / exp_sum
    return dist

y = softmax(logits)
```

```
def forward(y, label):
    L = len(label)
    V, T = y.shape
    alpha = np.zeros([L,T])
    # init first column
    alpha[0,0] = y[label[0],0]  # TODO
    alpha[1,0] = y[label[1],0]  # TODO
    # run dp
    for t in range(1,T):
        for s in range(L):
            k = label[s]
            y_k_t = y[k,t]
            alpha_tmp = alpha[s,t-1]
            if s>0:
                alpha_tmp += alpha[s-1,t-1]  # TODO
            if s>1 and k!=0 and k!=label[s-2]:
                alpha_tmp += alpha[s-2,t-1]  # TODO
            alpha[s,t] = alpha_tmp*y_k_t
    return alpha
```

Backward

1. `Vocab = [0, 1, 2, 3, 4]`, 其中 **0 表示 'blank'**.
2. `V = len(Vocab) = 5` 是字典大小, 譬如字典有 4 個 labels 加上一個 blank, 因此 V=5
3. `l` 是正確答案 (已經安插 blanks 了), 例如 `l = [0, 3, 0, 3, 0, 4, 0]`
4. `L = len(l) = 7`
5. **後驗概率 `y`** (shape= `[V,T]`), 其中 `T` 表示 input sequence 長度, 所以 `y[k,t]` 表示時間點 `t`, label `k` 的後驗概率

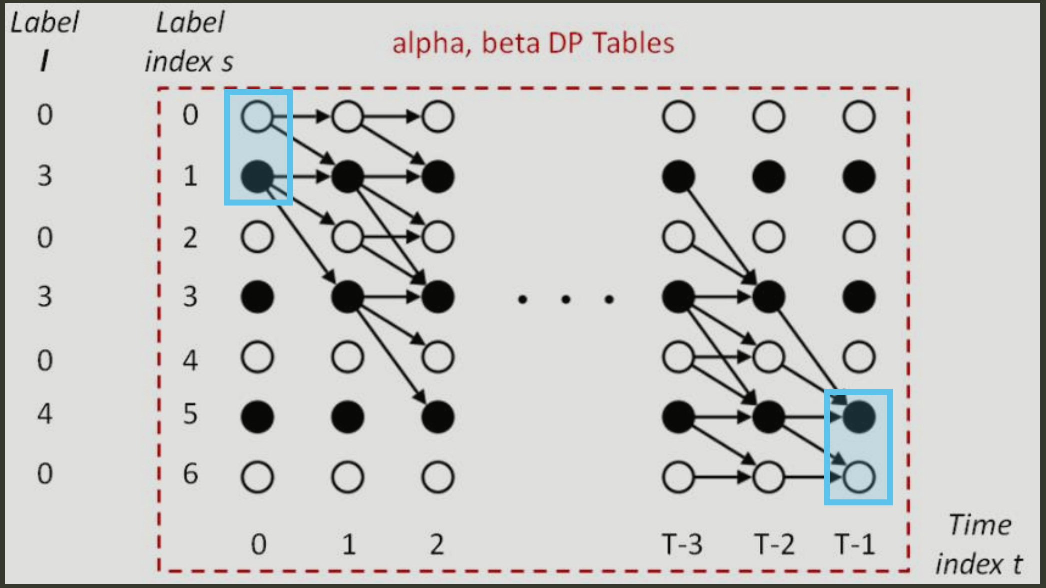
```
Vocab = [0,1,2,3,4]
l = [0, 3, 0, 3, 0, 4, 0]
V, L = len(Vocab), len(l)
T = 12
logits = np.random.random([V,T])

def softmax(logits):
    max_value = np.max(logits, axis=0, keepdims=True)
    exp = np.exp(logits - max_value)
    exp_sum = np.sum(exp, axis=0, keepdims=True)
    dist = exp / exp_sum
    return dist

y = softmax(logits)
```

```
def backward(y,label):
    L = len(label)
    V, T = y.shape
    beta = np.zeros([L,T])
    # init last column
    beta[-1,-1] = y[label[-1],-1]  # TODO
    beta[-2,-1] = y[label[-2],-1]  # TODO
    # run dp
    for t in range(T-2,-1,-1):
        for s in range(L):
            k = label[s]
            y_k_t = y[k,t]
            beta_tmp = beta[s,t+1]
            if s<L-1:
                beta_tmp += beta[s+1,t+1]  # TODO
            if s<L-2 and k!=0 and k!=label[s+2]:
                beta_tmp += beta[s+2,t+1]  # TODO
            beta[s,t] = beta_tmp*y_k_t
    return beta
```

Verify Forward and Backward

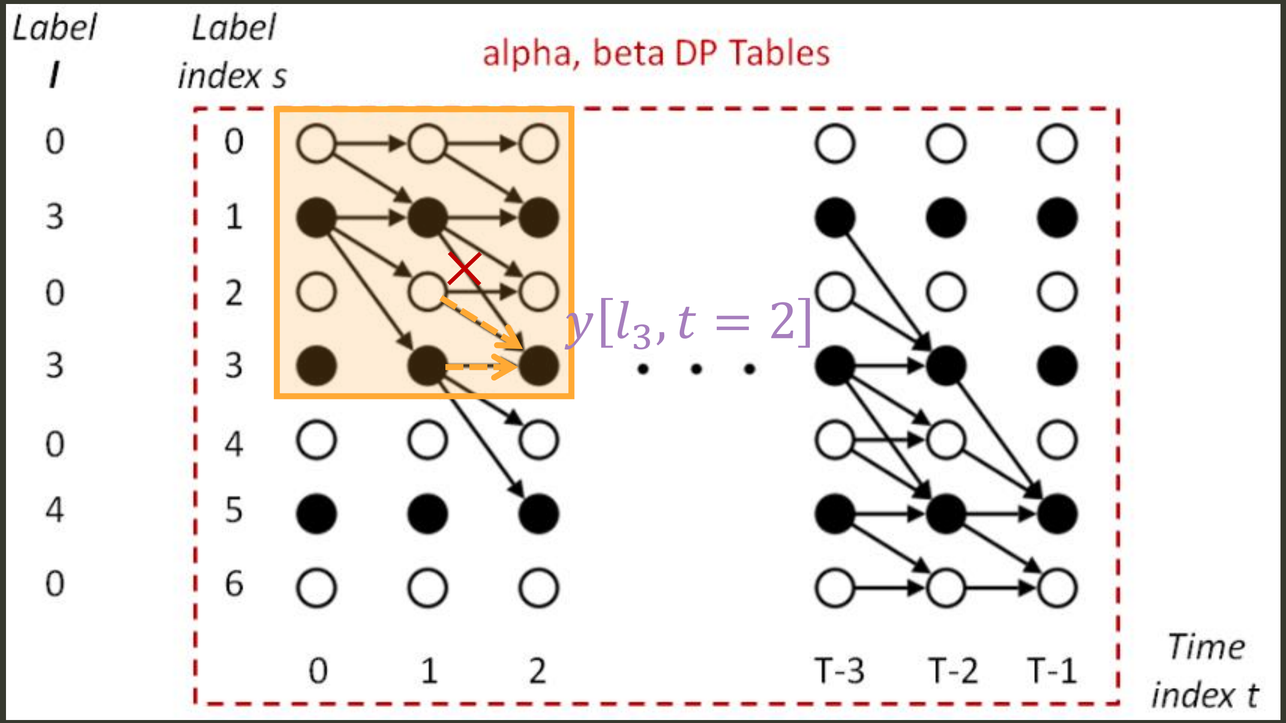



```python
# Forward and Backward likelihood should be very close
alpha = forward(y,l)
likelihood_by_forward =  alpha[-1,-1] + alpha[-2,-1]
print('likelihood_by_forward = {}'.format(likelihood_by_forward))

beta = backward(y,l)
likelihood_by_backword =  beta[0,0] + beta[1,0]
print('likelihood_by_backword = {}'.format(likelihood_by_backword))
```

```
likelihood_by_forward = 3.90062058761397e-06
likelihood_by_backword = 3.900620587613969e-06
```

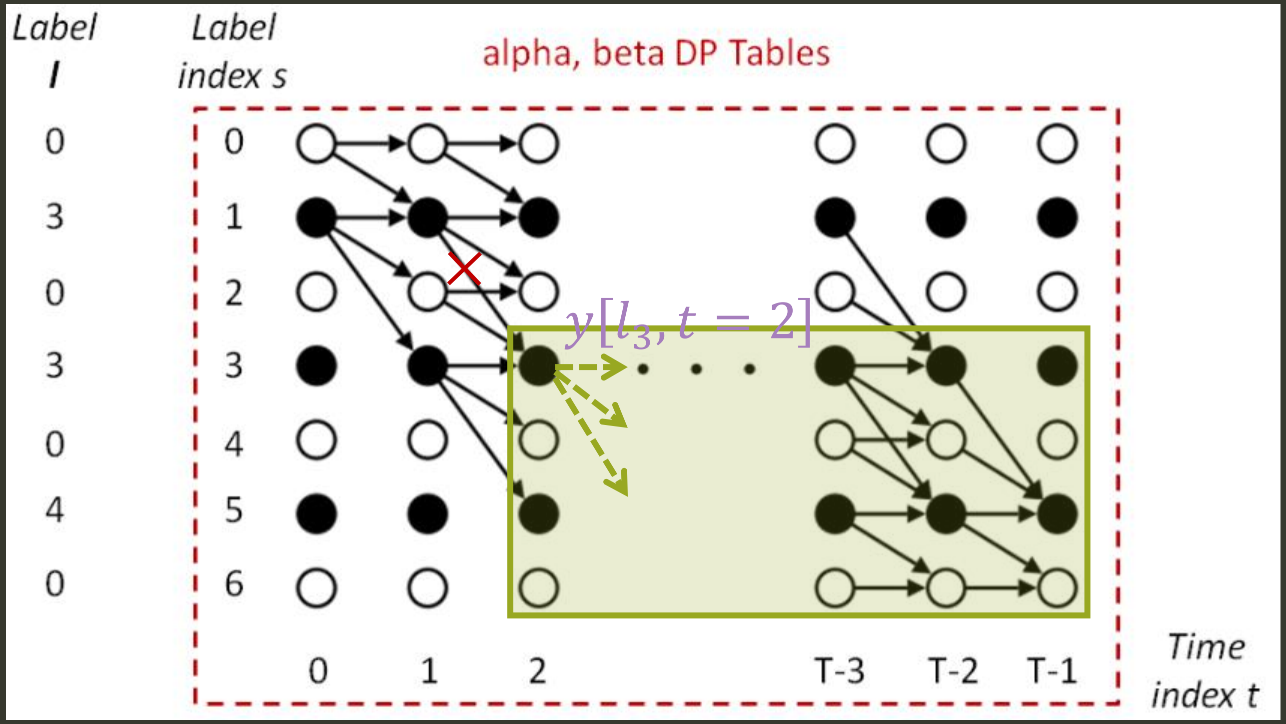
$$\alpha_t(s)\beta_t(s)$$



$$\alpha_2(3) = \left(\alpha_1(2) + \alpha_1(3)\right) y[l_3, t = 2]$$

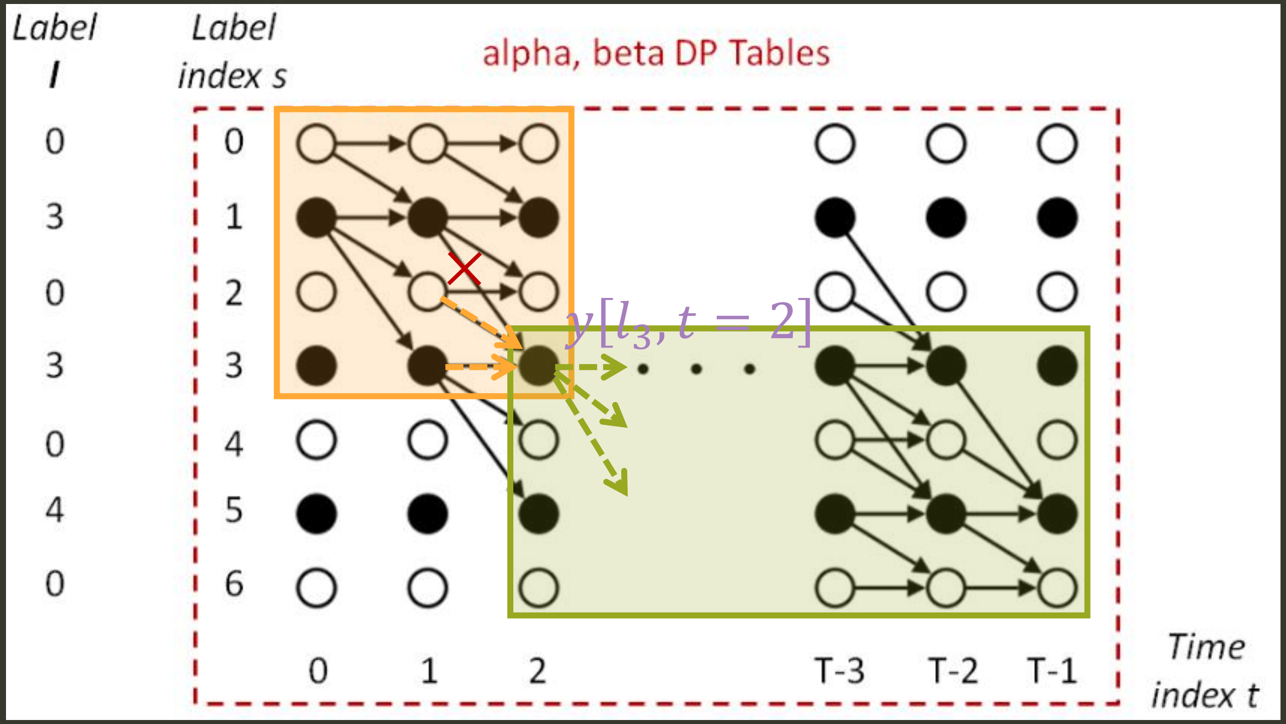
$$y[l_{s=3} = 3, t = 2]$$

$$\alpha_t(s)\beta_t(s)$$



$$\beta_2(3) = \big(\beta_3(3) + \beta_3(4) + \beta_3(5)\big)y[l_3, t = 2]$$

$$\alpha_t(s)\beta_t(s)$$

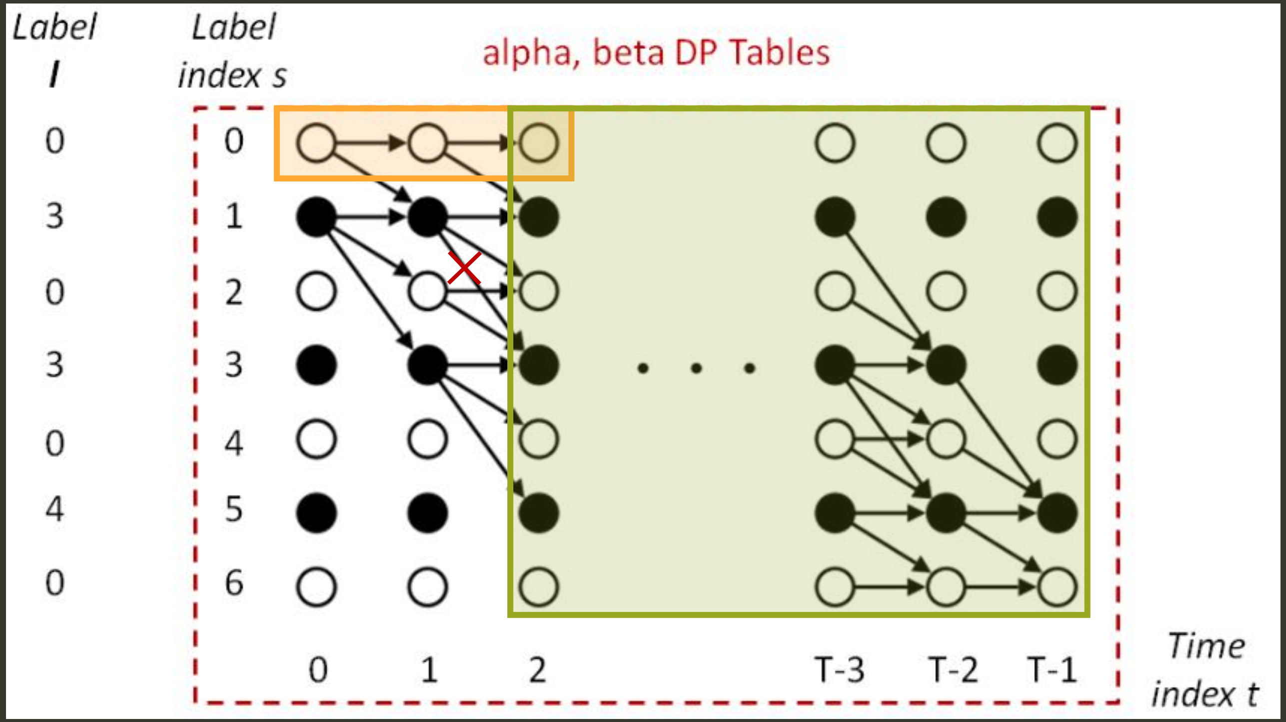


$$\alpha_2(3)\beta_2(3) = \underline{\big(\alpha_1(2) + \alpha_1(3)\big)y[l_3, t = 2]\big(\beta_3(3) + \beta_3(4) + \beta_3(5)\big)}y[l_3, t = 2]$$

$$= P_{t=2,s=3}(334|\mathrm{X})y[l_3, t = 2]$$

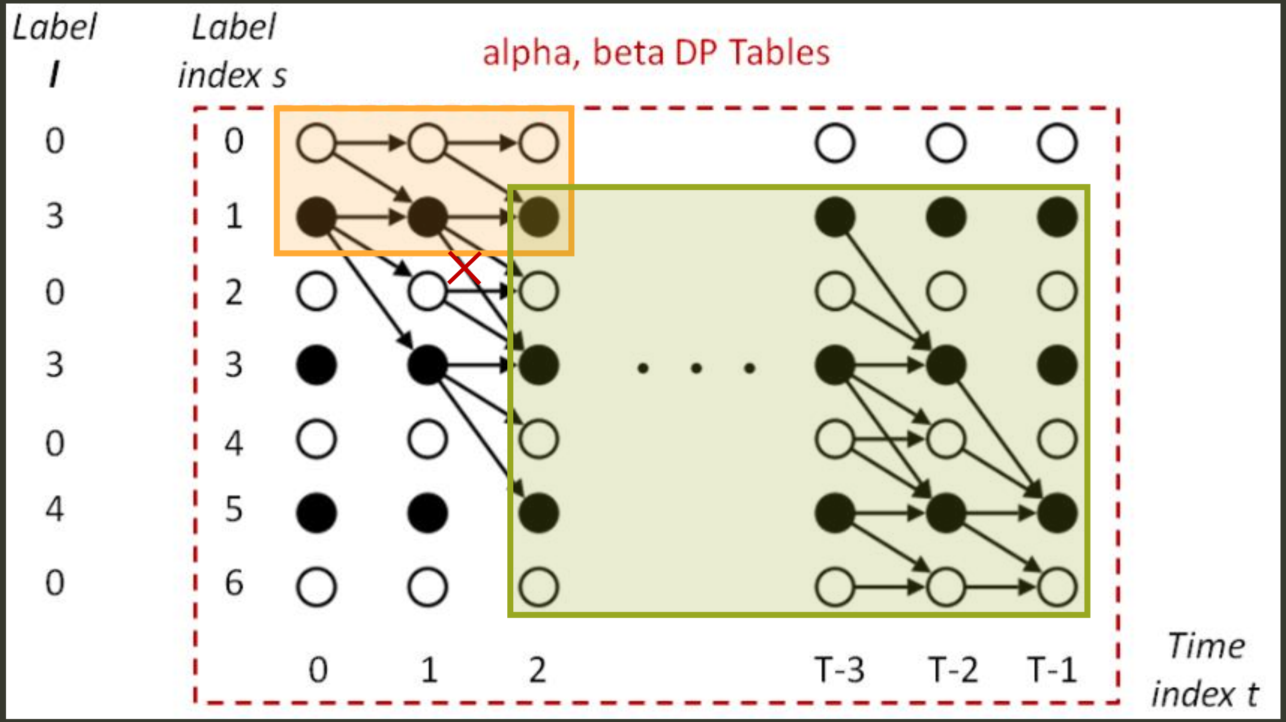
所有經過該
node的路徑

$$\alpha_2(0)\beta_2(0)$$



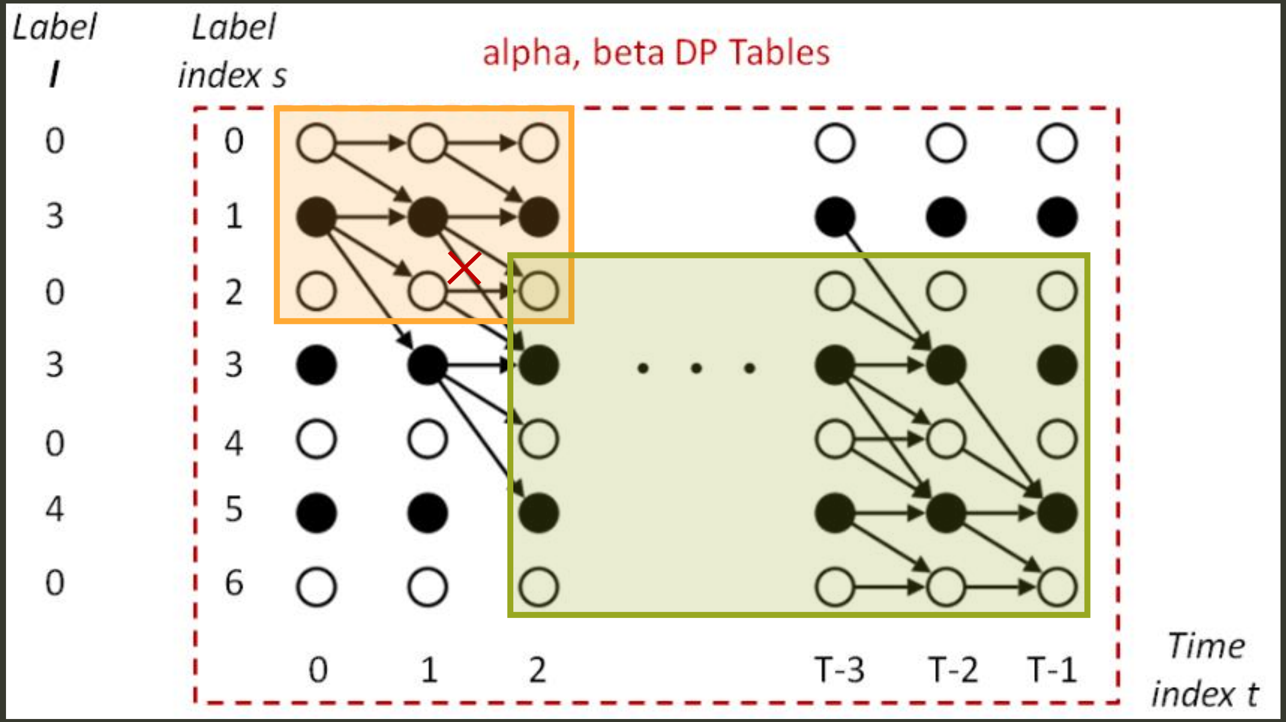
$$P(334|\mathrm{X}) = \sum_s P_{t,s}(334|\mathrm{X})$$

$$\alpha_2(1)\beta_2(1)$$



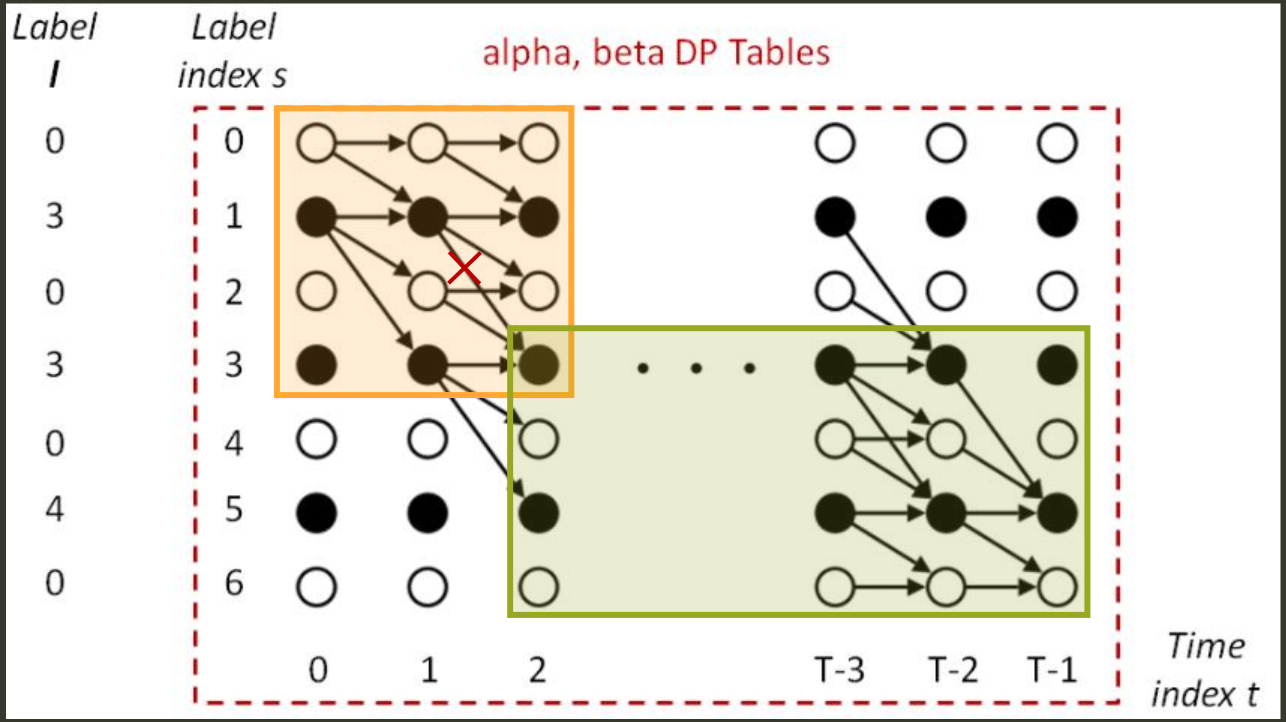
$$P(334|X) = \sum_s P_{t,s}(334|X)$$

$\alpha_2(2)\beta_2(2)$



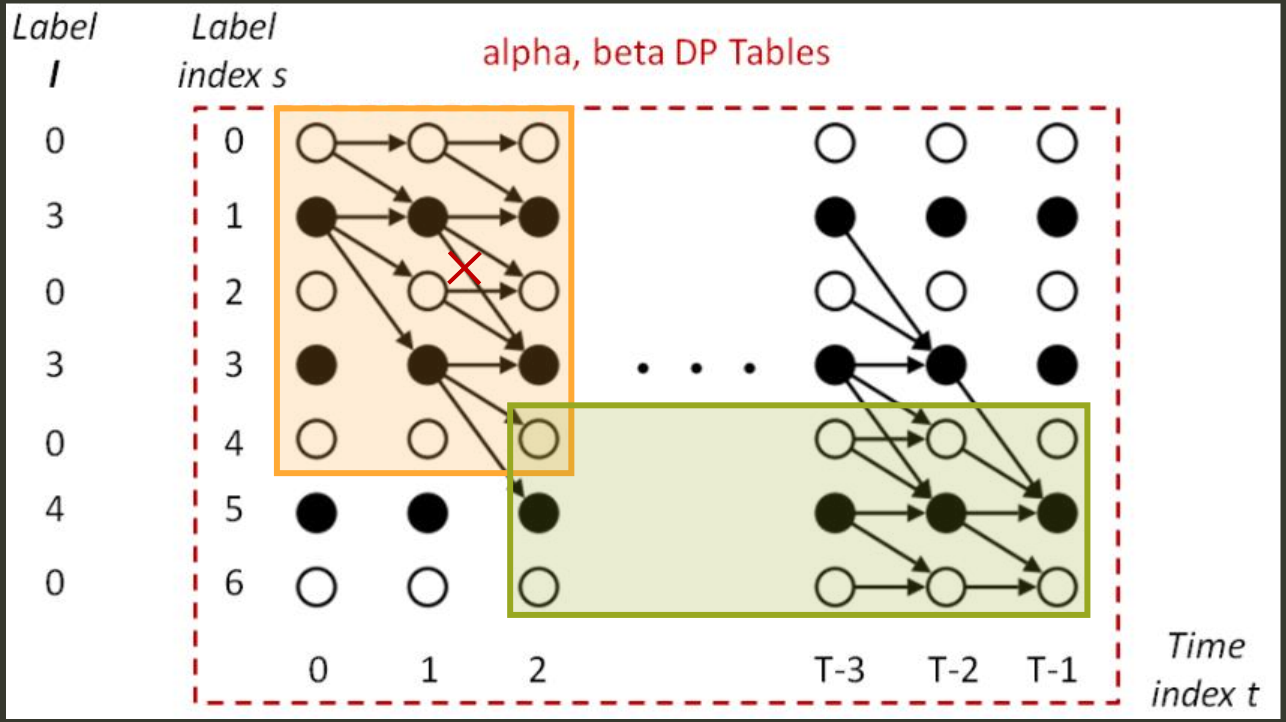
$$P(334|X) = \sum_s P_{t,s}(334|X)$$

$$\alpha_2(3)\beta_2(3)$$



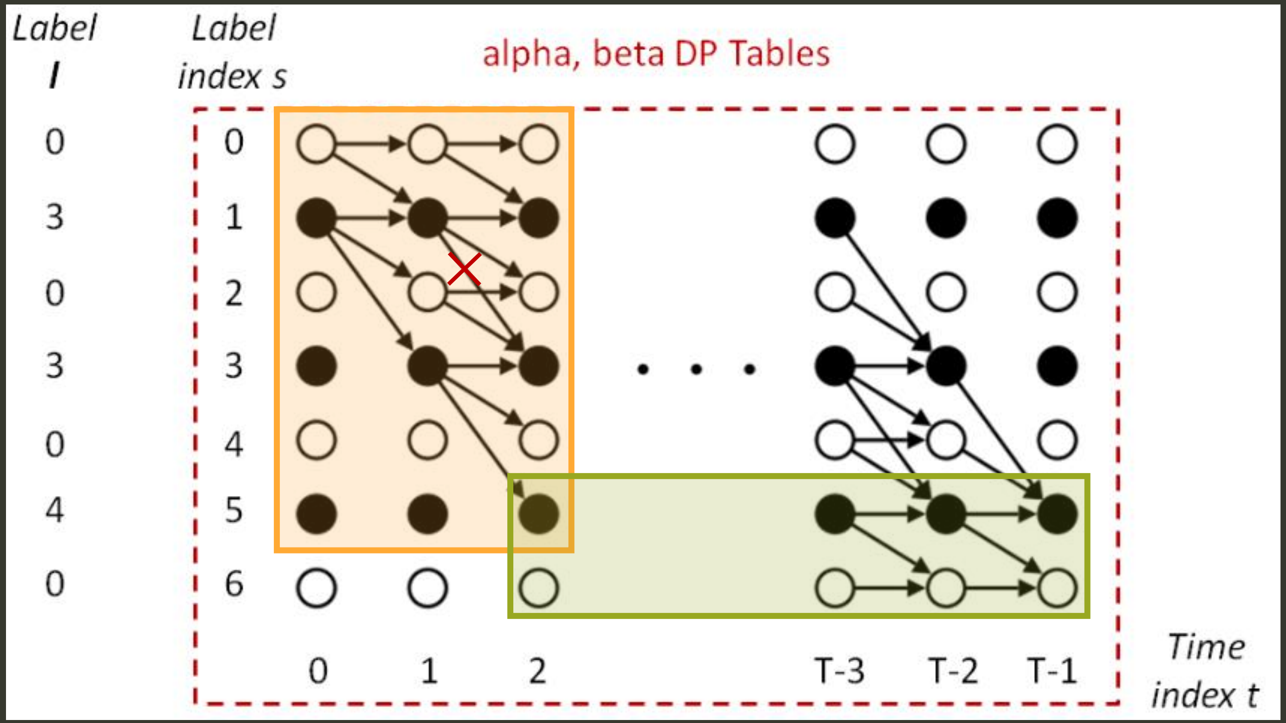
$$P(334|\mathrm{X}) = \sum_s P_{t,s}(334|\mathrm{X})$$

$$\alpha_2(4)\beta_2(4)$$



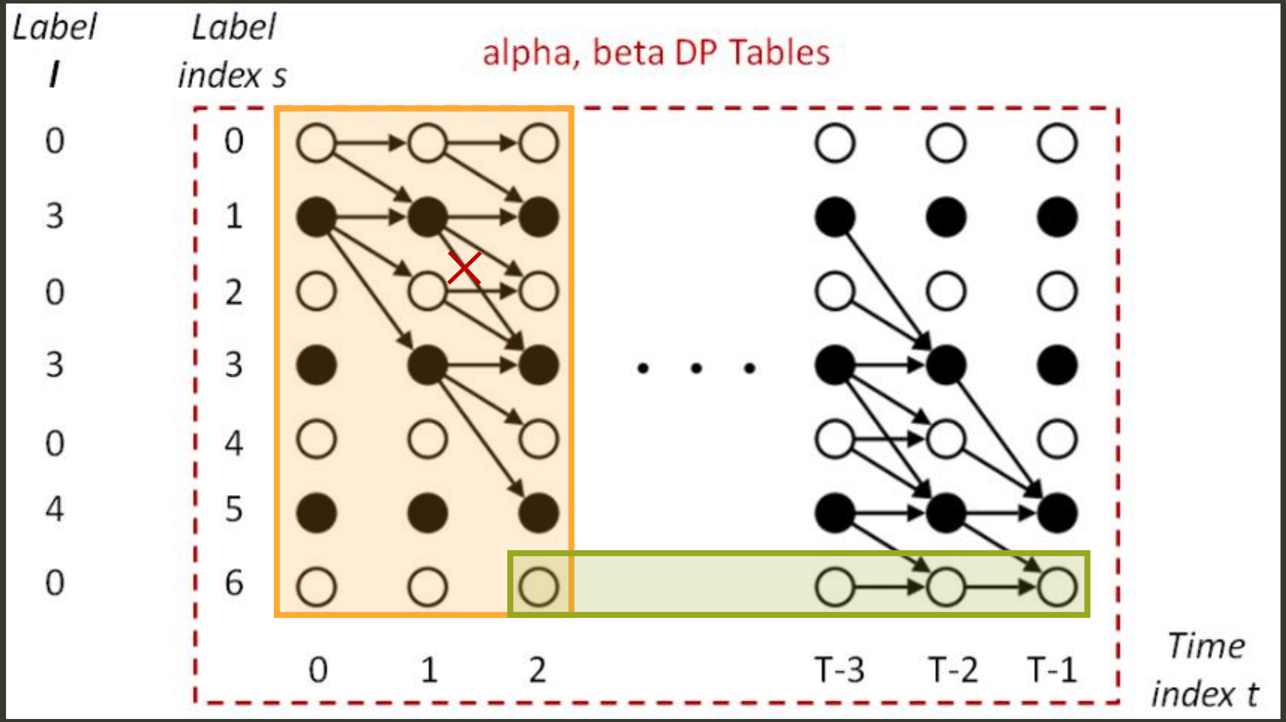
$$P(334|X) = \sum_s P_{t,s}(334|X)$$

$$\alpha_2(5)\beta_2(5)$$



$$P(334|X) = \sum_s P_{t,s}(334|X)$$

$$\alpha_2(6)\beta_2(6)$$



$$P(334|\mathrm{X}) = \sum_s P_{t,s}(334|\mathrm{X})$$

$P(334|\mathrm{X})$

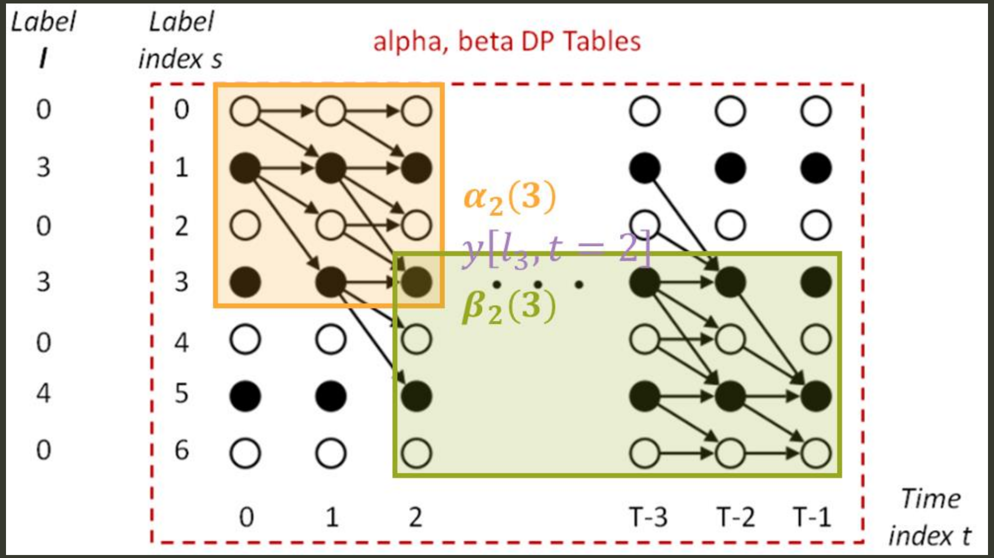
1. $\alpha_2(3)\beta_2(3) = P_{t=2,s=3}(334|\mathrm{X})y[l_3, t = 2]$
2. $P(334|\mathrm{X}) = \sum_s P_{t,s}(334|\mathrm{X})$

- 由 1. and 2. 得知:
- $P(334|\mathrm{X}) = \sum_s P_{2,s}(334|\mathrm{X}) = \sum_s \alpha_2(s)\beta_2(s)/y[l_s, 2]$

- 可以使用 MLE 最佳化了, Gradient-based optimization
- $\frac{\partial P(334|\mathrm{X})}{\partial y[l_3,2]} = \frac{\partial}{\partial y[l_3,2]}(\alpha_2(3)\beta_2(3)/y[l_3, 2])$, 只跟 $s = 3$ 時有關

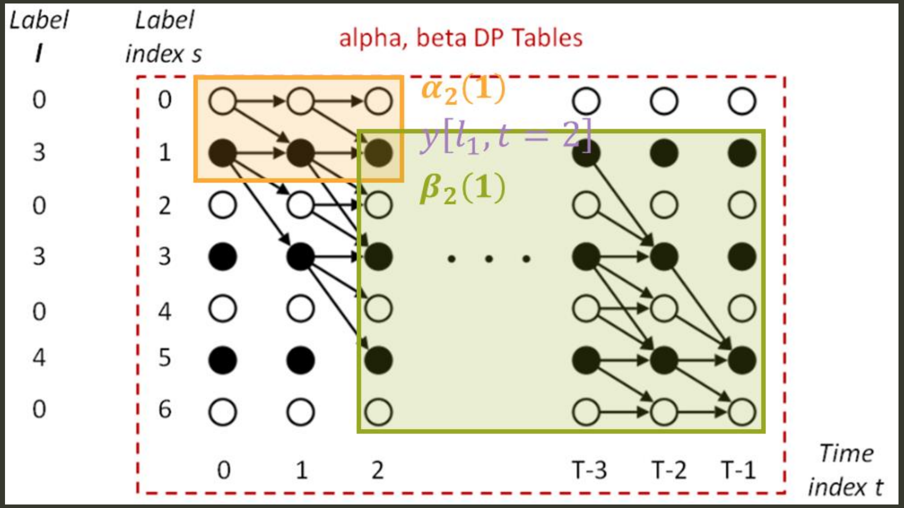
$$\frac{\partial P(334|X)}{\partial y[l_3, 2]}$$

- 可以使用 MLE 最佳化了, Gradient-based optimization
  - 經過一番努力推導
- $\frac{\partial P(334|X)}{\partial y[l_3,2]} = \frac{\partial}{\partial y[l_3,2]}(\alpha_2(3)\beta_2(3)/y[l_3, 2])$
- $= \frac{\partial}{\partial y[l_3,2]}\left[\left(\alpha_1(2) + \alpha_1(3)\right)y[l_3, t = 2]\left(\beta_3(3) + \beta_3(4) + \beta_3(5)\right)\right]$
- $= \left(\alpha_1(2) + \alpha_1(3)\right)\left(\beta_3(3) + \beta_3(4) + \beta_3(5)\right)$
- $= \alpha_2(3)\beta_2(3)/(\, y[l_3, 2]\,)^2$

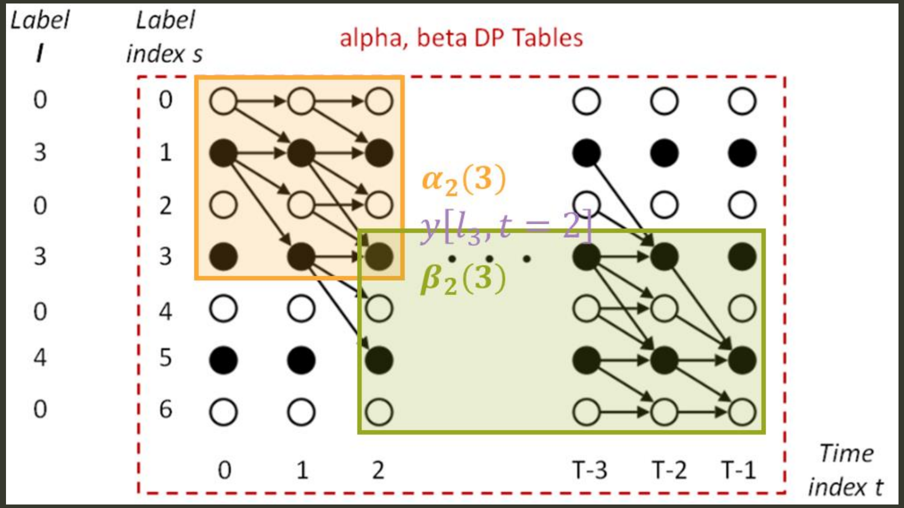


$$\frac{\partial P(334|X)}{\partial y[3,2]}$$

- 其實 $y[l_1, 2] = y[l_3, 2] = y[3,2]$
- 所以
  - $\frac{\partial P(334|X)}{\partial y[3,2]} = \frac{\partial P(334|X)}{\partial y[l_1,2]} + \frac{\partial P(334|X)}{\partial y[l_3,2]} = \frac{\alpha_2(1)\beta_2(1)}{(y[l_1,2])^2} + \frac{\alpha_2(3)\beta_2(3)}{(y[l_3,2])^2}$



+



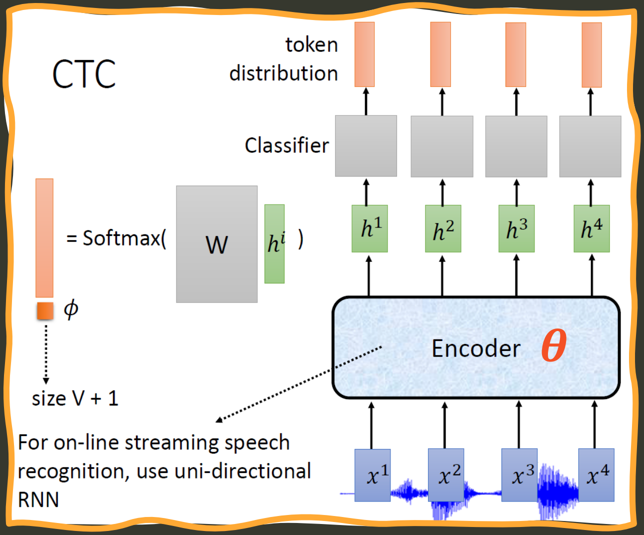
$$\frac{\partial P(334|\mathrm{X})}{\partial y[3,2]}$$

- 其實 $y[l_1,2] = y[l_3,2] = y[3,2]$
- 所以
  - $\frac{\partial P(334|\mathrm{X})}{\partial y[3,2]} = \frac{\partial P(334|\mathrm{X})}{\partial y[l_1,2]} + \frac{\partial P(334|\mathrm{X})}{\partial y[l_3,2]} = \frac{\alpha_2(1)\beta_2(1)}{(y[l_1,2])^2} + \frac{\alpha_2(3)\beta_2(3)}{(y[l_3,2])^2}$
- 論文裡神奇的 gradient 公式就此產生

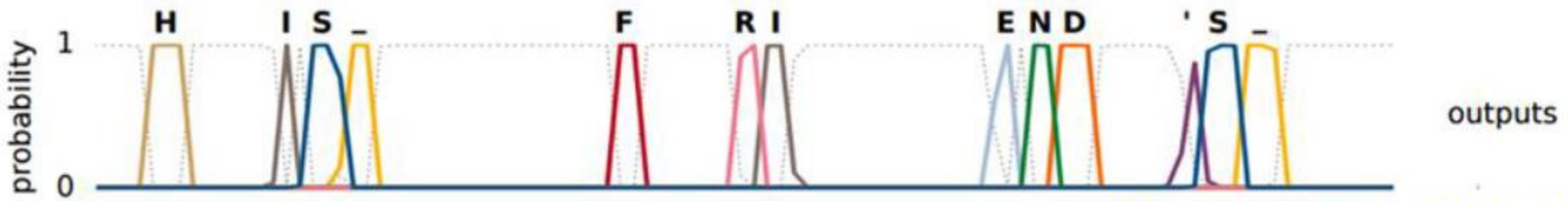
$$\frac{\partial p(\mathbf{l}|\mathbf{x})}{\partial y_k^t} = \frac{1}{{y_k^t}^2} \sum_{s \in lab(\mathbf{l},k)} \alpha_t(s)\beta_t(s). \qquad (15)$$

喘口氣的結論

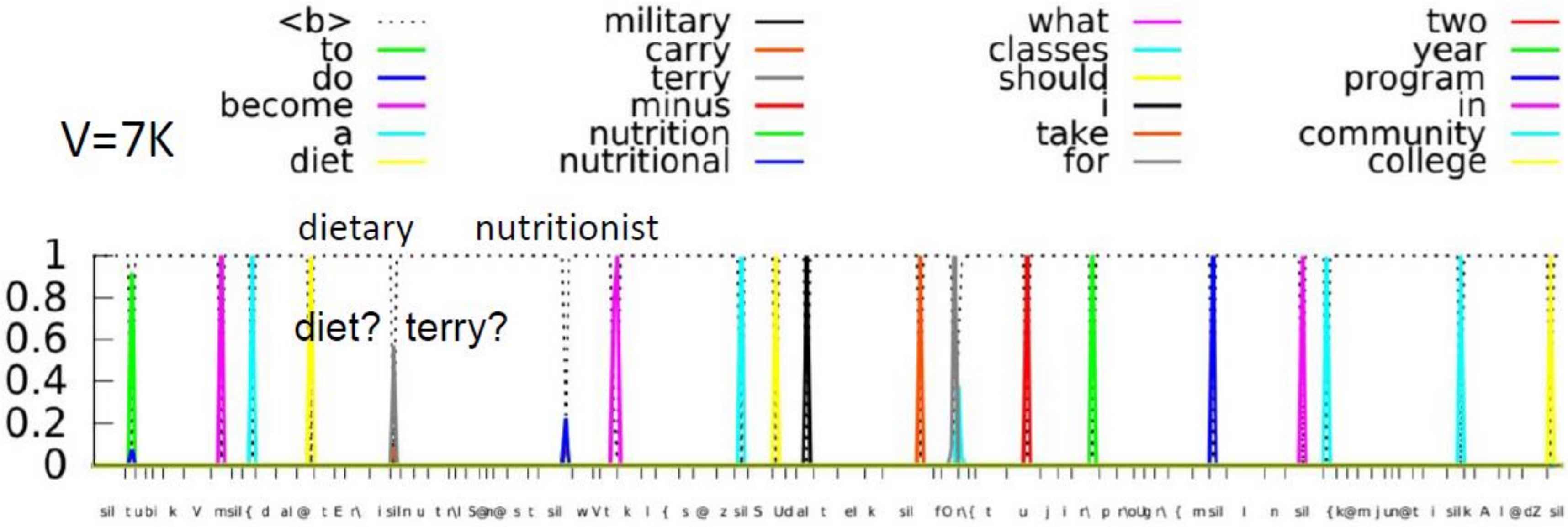
- CTC 是個 decoder 為獨立的 linear classifier 的 seq2seq model
- CTC 是個 loss function
  - $Loss_{CTC} = -\log P(text|\{h_1, h_2, h_3, \dots, h_T\})$
    $= -\log P(text|\{x^1, x^2, x^3, \dots, x^T\}, \theta)$
  - 所有 alignment 滿足去重和去 ϕ 的規則後 $= text$ 都算進機率中
    - 比 HMM 的 alignment 更彈性
    - RNN-T 更彈性, but 更複雜



Does CTC work?



[Graves, et al., ICML'14]



One can increase V to obtain better performance

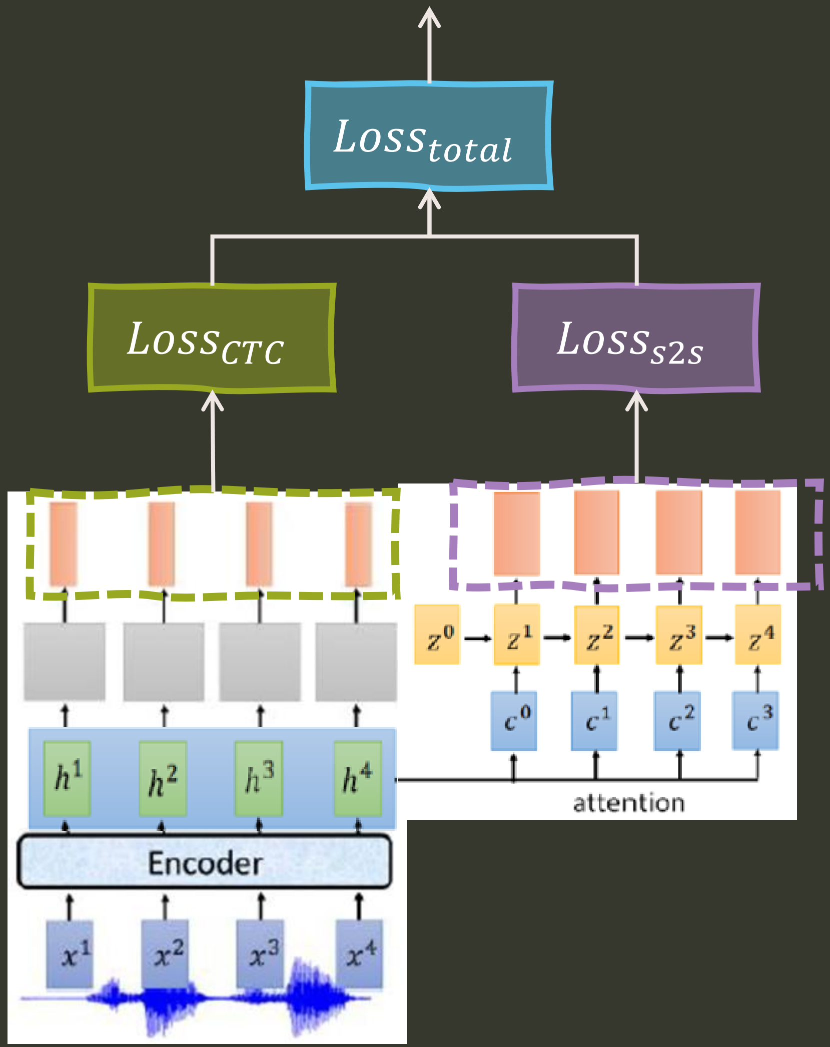
[Sak, et al., INTERSPEECH'15]

其他

- Very good reference: [DingKe完整實作ctc]
- Tensorflow 支援 CTC loss 的 op: 參考 [my_practice]
- $Loss_{CTC}$ 直接對 logits 計算 gradient: 參考 [my_practice]
- Forward/backward 數值穩定技巧
  - Log domain 方法,參考 [my_practice]
  - Scaling 方法 (原始 paper 方式), [DingKe完整實作ctc]
- Decoding [distill], [DingKe完整實作ctc]
  - Greedy
  - Beam search
  - Prefix beam search (挺複雜的)
- RNN-T 請參考李宏毅老師的課程 [ref1] [ref2]
  - 最好的 rnn-t 介紹, 沒有之一!

近期 state-of-the-art 的 E2E ASR 架構

- CTC + seq2seq(attention)
  - 或 RNN-T + seq2seq(attention)
- $Loss_{CTC}$ 幫助 encoder 出來的 $\{h^t\}$ 本身就有良好結構
  - 因為能做 ASR
- 因此這樣的 $\{h^t\}$ 拿去給 decoder 做 $Loss_{s2s}$ 會比較容易



[ESPnet] end-to-end speech processing toolkit

- Kaldi style complete recipe
- ASR: Automatic Speech Recognition
- TTS: Text-to-speech
- ST: Speech Translation & MT: Machine Translation
- VC: Voice conversion
- DNN Framework
    - Flexible network architecture thanks to chainer and pytorch

[ESPnet] end-to-end speech processing toolkit

ASR: Automatic Speech Recognition

- **State-of-the-art performance** in several ASR benchmarks (comparable/superior to hybrid DNN/HMM and CTC)
- **Hybrid CTC/attention** based end-to-end ASR
  - Fast/accurate training with CTC/attention multitask training
  - CTC/attention joint decoding to boost monotonic alignment decoding
  - Encoder: VGG-like CNN + BiRNN (LSTM/GRU), sub-sampling BiRNN (LSTM/GRU) or Transformer
- Attention: Dot product, location-aware attention, variants of multihead
- Incorporate RNNLM/LSTMLM/TransformerLM trained only with text data
- Batch GPU decoding
- **Transducer** based end-to-end ASR
  - Available: RNN-Transducer, Transformer-Transducer, mixed Transformer/RNN-Transducer
  - Also support: attention mechanism (RNN-decoder), pre-init w/ LM (RNN-decoder), VGG-Transformer (encoder)