

# Connectionist Temporal Classification

Yi Wang

## The Speech Recognition Problem

The input is an audio clip  $\mathbf{x}$ , as a sequence of frames:

$$\mathbf{x} = \{\mathbf{x}_1 \dots \mathbf{x}_T\}$$

where each frame at time  $t$ ,  $\mathbf{x}_t$ , consists of weights of  $G$  spectrograms:

$$\mathbf{x}_t = [x_{t,1} \dots x_{t,G}]^T$$

We can design a RNN to output  $\mathbf{y}_t$  for each input frame  $\mathbf{x}_t$ :

$$\mathbf{y} = \{\mathbf{y}_1 \dots \mathbf{y}_T\}$$

where each output  $\mathbf{y}_t$  is a probability distribution over an alphabet:

$$\mathbf{y}_t = [y_{t,1} \dots y_{t,K}]^T$$

where  $y_{t,k} = P(l_t = k)$ , and  $l_t$  denotes the letter pronounced at time  $t$ .

Now, the problem comes – how can we interpret  $\mathbf{y}$  as a sequence of letters.

At the first sight, this is straightforward – we just find a path

$$\boldsymbol{\pi} = \{\pi_1 \dots \pi_T\}$$

where  $\pi_t \in [1, K]$ , by

$$\boldsymbol{\pi} = \arg \max_{\boldsymbol{\pi}} P(\boldsymbol{\pi}|\mathbf{y})$$

where  $P(\boldsymbol{\pi}|\mathbf{y})$  could be defined as, say,

$$P(\boldsymbol{\pi}|\mathbf{y}) = \prod_{t=1}^T y_{t,\pi_t}$$

or

$$P(\boldsymbol{\pi}|\mathbf{y}) = \prod_{t=1}^T y_{t,\pi_t} P(\pi_t|\pi_{t-1})$$

where  $P(\pi_t|\pi_{t-1})$  comes from a pre-trained n-gram language model.

However, these are NOT what we need. Imagine that people are saying “a”. There could be silence (or *blank*) before and/or after the pronunciation. So the path  $\boldsymbol{\pi}$  could be

```

aaaaaaaa
aaaaa___
____aaaa
__aaaaa_

```

This does not even consider that the period of silences and pronunciations vary. So the following cases all correspond to “a”:

```

aa_____
aaa_____
_aaa_____
___aaaa_
_aaaaaa_

```

This tells that the output we really want is not  $\mathbf{y}$  nor  $\boldsymbol{\pi}$ . Instead, it is some sequence called *label* and denoted by  $\mathbf{l}$ :

$$\mathbf{l} = \{l_1 \dots l_S\}$$

where  $S \leq T$ .

For above example, all those paths correspond to the same label:  $\mathbf{l} = \{a\}$ .

Few more examples help reveal the relationship between path  $\boldsymbol{\pi}$  and labels  $\mathbf{l}$ . All the following examples correspond to  $\mathbf{l} = \{h, e\}$ :

```

hhheeee
__heeee
_hee___
hh_eee
_hh_eee
h__ee__
__h_ee_

```

Please note that blanks could appear before, between, and after consequent appearances of “h” (and “e”).

A little more complex example is that all the following paths correspond to label  $\mathbf{l} = \{b, e, e\}$ :

```
bbbeee_ee
_bb_ee__e
__bbbe_e_
```

except for

```
_b_eeeeee
bbb__eeee
_bb_eee__
```

which correspond to  $\mathbf{l} = \{b, e\}$ . This example shows the importance of blank in separating consecutive letters.

In summary, the output of speech recognition is not alphabet probability distribution  $\mathbf{y}$  or path  $\boldsymbol{\pi}$ , but label  $\mathbf{l}$ . And the CTC approach is all about infer  $\mathbf{l}$  from  $\mathbf{y}$ , by integrating out  $\boldsymbol{\pi}$ :

$$P(\mathbf{l}|\mathbf{x}) = \sum_{\boldsymbol{\pi}} P(\mathbf{l}|\boldsymbol{\pi})P(\boldsymbol{\pi}|\mathbf{x})$$

where

$$P(\mathbf{l}|\boldsymbol{\pi}) = \begin{cases} 1 & \text{if } \boldsymbol{\pi} \text{ matches } \mathbf{l} \\ 0 & \text{otherwise} \end{cases}$$

$$P(\boldsymbol{\pi}|\mathbf{x}) = \prod_{t=1}^T y_{t, \pi_t}$$

## Dynamic Time Warping

Because the length of  $\mathbf{y}$  might differ from (often longer than)  $\mathbf{l}$ , so the inference of  $\mathbf{l}$  from  $\mathbf{y}$  is actually a dynamic time warping problem.

There is a dynamic programming algorithm that can solve this time warping problem. And this algorithm enables us to train the RNN using  $\mathbf{x}$  and  $\mathbf{l}$  pairs, instead of  $\mathbf{x}$  and  $\mathbf{y}$  pairs. This is very valuable because there is no need to segment  $\mathbf{y}$  and align  $\mathbf{y}$  with  $\mathbf{x}$  before training.

When we say time warping, we mean that we want to map each frame  $\mathbf{y}_t$  to some  $l_s$ . The second example above shows that we actually want to make  $\mathbf{y}_t$  to either a  $l_s$  or a blank, because only if we do so, we have the chance to recognize successive letters like “ff” and “ee” in “coffee”, and “ee” in “bee”.

Here is a more detailed example about this. Suppose that we have an audio clip of “bee”. The most probable path  $\pi$  is

```
___bbeeee_____
```

we would like to wrap it to  $\mathbf{l} = \{b, e, e\}$ . It looks like we can map as

```
___bbeeee_____
  \|||//
   |||
   bee
```

but the truth is that we do not have much information to prevent the algorithm from mapping all e’s in  $\pi$  into the first e in  $\mathbf{l}$ :

```
___bbeeee_____
  \|||//
   ||
   bee
```

And another problem is that to whom should we map those blanks in  $\pi$ ?

A solution other than to map  $\pi$  to  $\mathbf{l}$  is to map  $\pi$  to  $\mathbf{l}'$ , which is constructed by inserting blanks before, after, and into  $\mathbf{l}$ . For the above example, we have

$$\mathbf{l}' = \{, b, , e, , e, \}$$

and the warping could work as:

```
___bbeeee_____
\\|||/|||/|///
  || ||| /
  _b_e_e_
```

An a little more special case could be that  $\pi$  does not contain leading blanks. In this case, it is reasonable to map the first frame to “b”, instead of the leading “\_”, of  $\mathbf{l}'$ :

```

bbbeeee_----
||/////////
||||| /
_b_e_e_

```

A similar edge case is that  $\pi$  does not have padding blanks, therefore no frame should be mapped to the padding blank of  $\mathbf{l}'$ . These two cases are what we need to care about in designing the algorithm that computes the map from  $\pi$  to  $\mathbf{l}'$ .

## The Forward-Backward Algorithm

Here we derive the dynamic programming algorithm for computing  $P(\mathbf{l}'|\mathbf{x})$ , which, is notably equivalent to  $P(\mathbf{l}|\mathbf{x})$ , because  $\mathbf{l}'$  is constructed to be unique for the given  $\mathbf{l}$ .

The above time warping example shows that  $\pi_T$  could be mapped to either  $l'_{|\mathbf{l}'|}$ , the padding blank of  $\mathbf{l}'$ , or  $l'_{|\mathbf{l}'|-1}$ , which, according to the construction rule of  $\mathbf{l}'$ , is the last element of  $\mathbf{l}$ . So we have:

$$P(\mathbf{l}'|\mathbf{x}) = P(\mathbf{l}', \pi_T = l'_{|\mathbf{l}'|}|\mathbf{x}) + P(\mathbf{l}', \pi_T = l'_{|\mathbf{l}'|-1}|\mathbf{x})$$

We can generalize  $P(\mathbf{l}', \pi_T = l'_{|\mathbf{l}'|}|\mathbf{x})$  and  $P(\mathbf{l}', \pi_T = l'_{|\mathbf{l}'|-1}|\mathbf{x})$  to be

$$\alpha(t, s) = P(\mathbf{l}'_{1:s}, \pi_t = l'_s|\mathbf{x})$$

so

$$P(\mathbf{l}'|\mathbf{x}) = \alpha(T, |\mathbf{l}'|) + \alpha(T, |\mathbf{l}'| - 1)$$

Above time warping example also shows that  $\pi_1$  could be mapped to either  $l'_1$ , the leading blank of  $\mathbf{l}'$ , or  $l'_2$ , the first element of  $\mathbf{l}$ . So we have

$$\alpha(1, 1) = y_{1, \_}$$

$$\alpha(1, 2) = y_{1, l_1}$$

$$\alpha(1, s) = 0, \forall s > 2$$

Here let us take an example. Suppose that  $\mathbf{l}' = \{\_, h, \_, e, \_\}$  and a  $\mathbf{y}$  (which, for the simplicity, is illustrated as a sequence of subscripts). It is reasonable to map  $\pi_1$  to  $l'_1$ , the leading blank of  $\mathbf{l}'$ , if  $\arg \max_k y_{1,k} = \_$ :

```

123456789
|
|
_h_e_

```

or to  $l'_2 = \text{"h"}$ , if  $\arg \max_k y_{1,k} = \text{"h"}$ :

```

123456789
|
\
_h_e_

```

Then we think a step further – mapping  $\pi_2$ , or more generally,  $\pi_s$ . Roughly, it is reasonable to map  $\pi_t$  to

1. where  $\pi_{t-1}$  was mapped to, denoted by  $l'_{s(t-1)}$ ,
2. the element next to  $l'_{s(t-1)}$ , denoted by  $l'_{s(t-1)+1}$ , or
3.  $l'_{s(t-1)+2}$ , if  $l'_{s(t-1)+1} = \text{"_"}$

Among these, case 3 is reasonable in case that we want to skip that blank  $l'_{s(t-1)+1} = \text{"_"}$ . An example is that when we want to map  $\pi = \{h, h, h, h, e, e, e\}$  to  $I' = \{\_, h, \_, e, \_\}$  – it is not mandatory to map any  $\pi_t$  to any blank in  $I'$ , in order to recognize the word “he”.

```

hhhhheee
\|//|//
| /
_h_e_

```

But case 3 is not reasonable when  $l'_{s(t-1)+2} = l'_{s(t-1)}$ . In this case, we should not skip that blank. For example, to recognize the word “bee”, we have  $I' = \{\_, b, \_, e, \_, e, \_\}$ . In these case, if we skip over the blank between the two  $e$ s in  $I'$ , we would misunderstand the double-e as a singel e. In the example below, even if frame ( $y_5$ ) *sounds* more like  $e$  than blank, we want to map it to  $l'_5 = \text{"_"}$ , so to recognize the word “bee”.

```

bbbeeeeee
\|||||//
|/|||
_b_e_e_

```

Summarizing above three cases, we get the following generalization rule for  $\alpha(t, s)$ :

$$\alpha(t, s) = \begin{cases} y_{t, l'_s} \sum_{i=s-1}^s \alpha(t-1, i) & \text{if } l'_s = \_ \text{ or } l'_s = l'_{s-2} \\ y_{t, l'_s} \sum_{i=s-2}^s \alpha(t-1, i) & \text{otherwise} \end{cases}$$

This general rule shows that, to compute  $\alpha(t, s)$ , we need  $\alpha(t-1, s)$ ,  $\alpha(t-1, s-1)$  and  $\alpha(t-1, s-2)$ . Some of these values are obviously zero. The following figure from Alex Graves' Ph.D. thesis helps us understand in particular which are zeros.

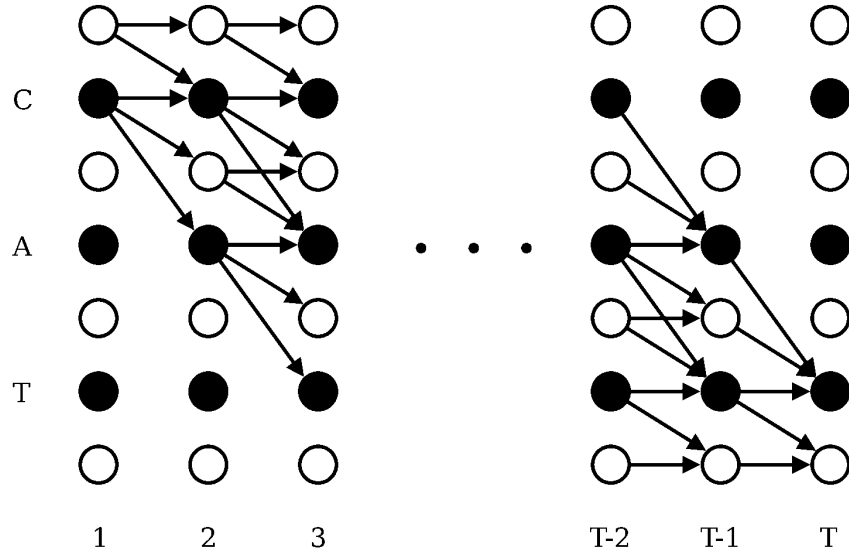


Figure 1: The search space of the forward-backward algorithm

## Quiz

1. Please illustrate the search space of the forward-backward algorithm given  $\mathbf{l} = \{b, e, e\}$ , like Alex Graves illustrates the case of  $\mathbf{l} = \{c, a, t\}$  with Figure 7.2 in his Ph.D. thesis.