Connectionist Temporal Classification

This is a reading note of the Connectionist Temporal Classification (CTC), an output layer designed to work with RNN to build speech recognition systems. When I was reading the paper, I thought it would be easier to understand if we can rephrase it. Here is my trial.

The Problem of Speech Recognition

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

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

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

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

The problem of speech recognition is to map $\mathbf{x}$ into a sequence of letters.

Step 1: from frames to letter distributions

An intuitive solution seems to map each $\mathbf{x}_t$ into a $\mathbf{y}_t$:

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

where $\mathbf{y}_t$ is a probability distribution over the K-dimensional alphabet:

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

where $y_{t,k} = P(l_t = k)$ and l_t is a letter.

Step 2: from letter distributions to letters

The next step is to convert the $T \times K$ lattice $\mathbf{y}$ into a sequence of letters, or a path

$$\pi = \{\pi_1 \dots \pi_T\} = \arg\max_{\pi} \prod_{t=1}^{T} y_{t,\pi_t} P(\pi_t | \pi_{t-1})$$

where $P(\pi_t|\pi_{t-1})$ comes from the language model.

The Viterbi algorithm can do the above optimization; however, π is not what we want. Suppose that a speaker says "a", the inputs frames $\mathbf{x}$ should be mapped into a sequence of consecutive `a`'s, with variable lengths, and leading or trailing slicence/blanks. All the following sequences are reasonable π's for this case:

```
aaaaaaaa
aaaaa___
aaa_____
a_______
_______a
_____aaa
___aaaaa
__aaaa__
```

Step 3: from letters to labels

What we really want is a sequence of *labels*:

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

where $S \leq T$, and the CTC layer is all about inferring $\mathbf{l}$ from $\mathbf{x}$ by optimizing the following likelihood

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

where $P(\mathbf{l} \mid \pi) = 1/N$, if π *matches* $\mathbf{l}$, and N is the number of π's that match $\mathbf{l}$. For example, in the above one, all those enlisted π's match the label $\mathbf{l} = \{a\}$. All the following π's match label $\mathbf{l} = \{b, e\}$:

```
bbbeeee
__bbeee
_bee___
bb__eee
_bb_eee
b__ee__
__b_ee_
```

Please be aware that blanks could appear before, between, and after consequent appearances of `h`'s and `e`'s.

Step 4: blank-separated labels

It is practically easier to represent a mapping if we replace the comma in labels by a special symbol blank _. For example, if we represent

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

by

$$\mathbf{l}' = \{\_b\_e\_e\_\}$$

we can use the following mapping between π and $\mathbf{l}'$

```
___bbeeee____
\\||/|||/|///
  || ||| /
  _b_e_e_
```

to tell that the utterance $\mathbf{x}$ is the word `bee`, but not `be`.

Please be aware that there is one and only one $\mathbf{l}'$ for each $\mathbf{l}$.

Also please be aware that in $\mathbf{l}'$, we add the leading and the trailing blanks, which are useful because we can map the leading and trailing silence of an utterance to them.

We should allow that the first frame be mapped to the first non-blank letter of $\mathbf{l}'$, so could we handle the case that there is no silence at the beginning of the utterance, as illustrated below:

```
 bbbeeee____
 |||||||||/
 |||||/
_b_e_e_
```

Similarly, we should allow that the last frame be mapped to the last non-blank letter of $\mathbf{l}'$, so could we handle the case that there is no silence at the end of the utterance.

The Forward-Backward Algorithm

The CTC paper presents a dynamic programming algorithm that derives $\mathbf{l}'$ from $\mathbf{x}$.

The Forward Algorithm

To state the fact that l'_s is assigned to π_t, we create a notation

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

An intuitive illustration is as follows Fig 1.(a):

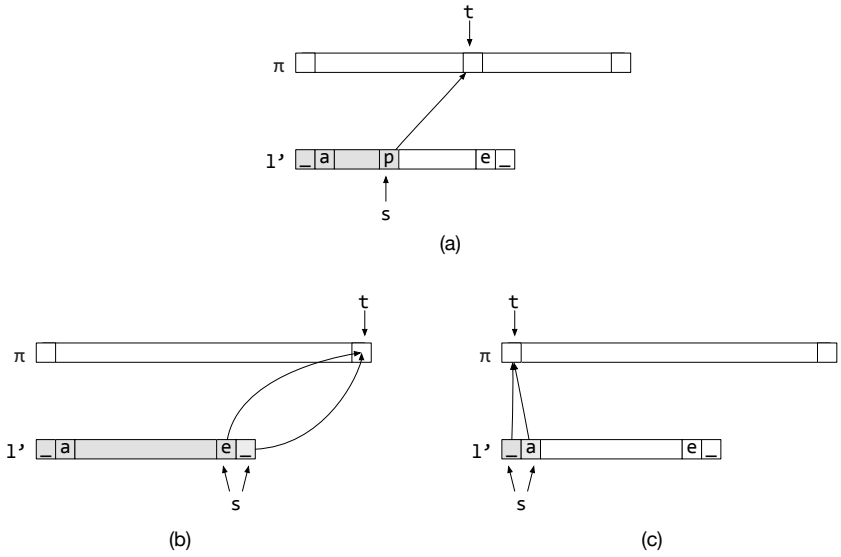

Figure 1: The α quantity.

Recall that in the above example, we show that π_T could be mapped to either $l'_{|\mathbf{l}'|}$, the trailing blank, or $l'_{|\mathbf{l}'|-1}$, the last letter in $\mathbf{l}$, as illustrated as Fig. 1(b).

Using the α notation:

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

The above example also shows that π_1 could be mapped to either l'_1, the leading blank of $\mathbf{l}'$, or l'_2, the first letter of $\mathbf{l}'$, as illustrated as Fig. 1(c).

Using the α notation, 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 π_1 to l'_1, the leading blank of $\mathbf{l}'$, if $\arg\max_k y_{1,k} =$`_`:

```
123456789
|
|
_h_e_
```

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

```
123456789
|
\
_h_e_
```

Then we think a step further – mapping π_2, or more generally, π_s. Roughly, it is reasonable to map π_t to

1. where π_{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 $l'_{s(t-1)+1} =$`_` and we want to skip that blank.

The first two cases are easy to understand, whereas case 3. is only reasonable if we want to skip that blank. Let's take a look at an example. Suppose that we want to map $\pi = \{h, h, h, h, e, e, e\}$ to $\mathbf{l}' = \{\_, h, \_, e, \_\}$. In this example, we don't have to map any π_t to any blank in $\mathbf{l}'$ in order to recognize the word `he`.

```
hhhheee
\|//|//
 |  /
_h_e_
```

However, case 3. is not reasonable when $l'_{s(t-1)+2} = l'_{s(t-1)}$, because in this case, we should not skip that blank. For example, to recognize the word "bee", we have $\mathbf{l}' = \{\_, b, \_, e, \_, e, \_\}$. In these case, if we skip over the blank between the two *e*s in $\mathbf{l}'$, we cannot tell which frames being mapped to the first `e` and which to the second.

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

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

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

The Backward Algorithm

Similar to the forward variable $\alpha(t, s)$, we can define the backward variable $\beta(t, s)$

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

Because the time warping must map the π_T to either $\mathbf{l}'_{|\mathbf{l}'|}$, the padding blank, with probability 1 or $\mathbf{l}'_{|\mathbf{l}'-1|}$, the last element in $\mathbf{l}$, with probability 1, we have

$$\beta(T, |\mathbf{l}'|) = 1$$
$$\beta(T, |\mathbf{l}'| - 1) = 1$$

Similar to the generalization rule of the forward algorithm, we have

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

The Search Space

This general rule for α shows that, to compute $\alpha(t, s)$, we need $\alpha(t-1, s)$, $\alpha(t-1, s-1)$ and $\alpha(t-1, s-2)$. Similarly, to compute $\beta(t, s)$, we need $\beta(t+1, s)$, $\beta(t+1, s+1)$, $\beta(t+1, s+2)$. Some of these values are obviously zero. The following figure (Figure 7.2 in Alex Graves' Ph.D. thesis) helps us understand which are zeros.

Every circle in this figure shows a possible state in the search space. These states are aligned in the grid of t and s. Arrows connect a state with its consequent states. These connected states are *possible* states, whereas the rest are *impossbile* and should have zero probability. We do not need to go into the *impossible* area to the top-right of those connected states when we compute $\alpha(t, s)$:

$$\alpha(t,s) = 0, \forall s < |\mathbf{l}'| - 2(T-t) - 1$$

And we do not need to go into the impossible area to the left-bottom when we compute $\beta(t, s)$:

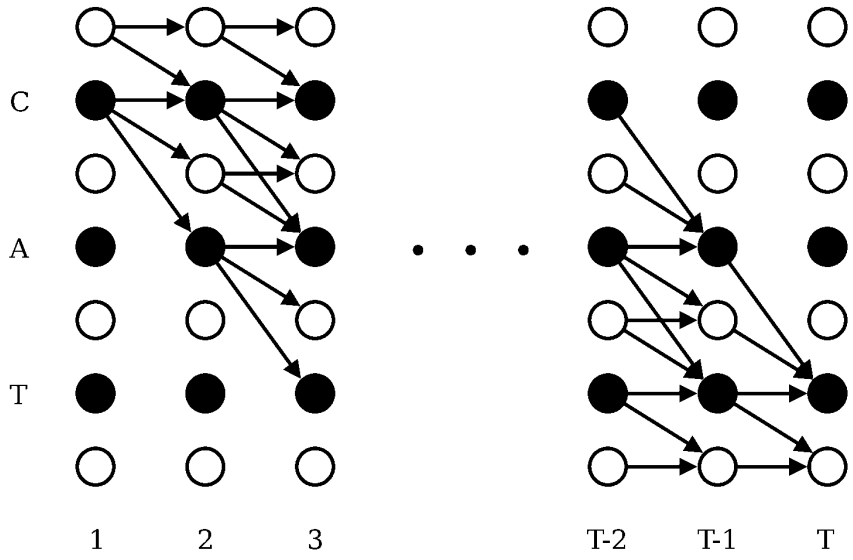

Figure 2: The search space of the forward-backward algorithm

$$\beta(t, s) = 0, \forall s > 2t$$

Actually, in order to bound the dynamic programming algorithm, we also need

$$\alpha(t, 0) = 0, \forall t$$

and

$$\beta(t, |\mathbf{l}'| + 1) = 0, \forall t$$

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.

2. Suppose that we can motion data collected from a sensor placed on our elbows. The data is a sequence of frames, and each frame records the 3D position and velocity of the sensor. Can we train a CTC network that counts how many push-ups we are doing?

3. How can we extend CTC into two dimensional case, and use this extended version of CTC for object recognition in computer vision?