Multi-path CTC

Varun Nayyar <vnayy@dolby.com>

23/04/2018

1 Intro to CTC

CTC (Graves 2006) provides a bedrock to ASR efforts of the past decade. CTC removed the need for phone labelling in datasets, as it would manage the alignment of such labels itself. CTC is defined

$$p(\pi|x) = \prod_{t=1}^{T} y^t_{\pi_t}$$
$$p(l|x) = \sum_{\forall \pi \in l} p(\pi|x)$$

Where l is the labelling provided and π are the possible paths that map to l. Note that $|l| = N$ while $|\pi| = T$, where $T \geq N$. See Graves 2006/preprint for more detail on the notation and the function for mapping $l \rightarrow \pi$

2 Multi-path CTC

One of the issues with using CTC for ASR is that CTC only works for a single labelling, but people speak in different ways. The word "water" for example has very different prononciations in British vs American English, especially noticeable with the t sound. Even within the same accent, words like tomato can have different prononciations and in countries like US and England, there is significant regional variation of accent so accent labelling accent is only so helpful, and datasets are providing to be a huge cost in this endevour.

2.1 Loss

Let us consider the multipath ctc $p_m(\mathbf{l}|x)$ where $\mathbf{l}$ is the set of unqiue CTC labellings possible for the multipath CTC. For example, water would have two different paths, one corresponding to the British version with a hard t, while the American version would have a soft t and an explicit r finish. Hence, by expanding the summation across the different paths, we can express this as the

sum of single path CTC.

$$\begin{aligned} p_m(\mathbf{l}|x) &= \sum_{\forall \pi \in \mathbf{l}} p(\pi|x) \\ &= \sum_{\forall l \in \mathbf{l}} \sum_{\forall \pi \in l_i} p(\pi|x) \\ &= \sum_{\forall l \in \mathbf{l}} p(l|x) \end{aligned} \tag{1}$$

I.e. the multipath CTC loss is equivalent to the sum of the CTC losses for the different unique CTC labellings. This tells us that the

$$p_m(\mathbf{l}|x) \geq p(l|x)$$

for $l \in \mathbf{l}$ since the probabilities of each path are non-negative. Finally,

$$\begin{aligned} L_m(\mathbf{l}, x) &= -\ln \sum_{\forall l} p(l|x) \\ &= -\ln \sum_{\forall l} \exp\left(-L(l, x)\right) \end{aligned} \tag{2}$$

2.2 Gradients

As per Graves, we have the gradient of each $p(l|x)$.

$$\frac{\partial}{\partial y_k^t} p(l|x) = \frac{1}{y_k^t} \sum_{u \in B(l,k)} \alpha(t,u)\beta(t,u)$$

Hence

$$\begin{aligned} \frac{\partial}{\partial y_k^t} p_m(\mathbf{l}|x) &= \frac{\partial}{\partial y_k^t} \sum_{\forall l \in \mathbf{l}} p(l|x) \\ &= \sum_{\forall l \in \mathbf{l}} \frac{\partial}{\partial y_k^t} p(l|x) \end{aligned}$$

and

$$\begin{aligned} L_m(\mathbf{l}, x) &= -\ln p_m(\mathbf{l}|x) \\ \therefore \frac{\partial}{\partial y_k^t} L_m(\mathbf{l}, x) &= \frac{-1}{p_m(\mathbf{l}|x)} \frac{\partial}{\partial y_k^t} p_m(\mathbf{l}|x) \\ &= \frac{-1}{p_m(\mathbf{l}|x)} \sum_{\forall l \in \mathbf{l}} \frac{\partial}{\partial y_k^t} p(l|x) \end{aligned} \tag{3}$$

Let us put this in the term of the simple CTC loss gradients, $\frac{\partial}{\partial y_k^t} L(l, x)$. Given

$$\frac{\partial}{\partial y_k^t} L(l, x) = \frac{-1}{p(l|x)} \frac{\partial}{\partial y_k^t} p(l|x)$$
$$\therefore \frac{\partial}{\partial y_k^t} p(l|x) = -p(l|x) \frac{\partial}{\partial y_k^t} L(l, x)$$

Hence

$$\begin{aligned} \frac{\partial}{\partial y_k^t} L_m(\mathbf{l}, x) =& \frac{-1}{p_m(\mathbf{l}|x)} \sum_{\forall l \in \mathbf{l}} \frac{\partial}{\partial y_k^t} p(l|x) \\ =& \frac{1}{p_m(\mathbf{l}|x)} \sum_{\forall l \in \mathbf{l}} p(l|x) \frac{\partial}{\partial y_k^t} L(l, x) \\ =& \sum_{\forall l \in \mathbf{l}} \frac{p(l|x)}{\sum_{\forall z \in \mathbf{l}} p(z|x)} \frac{\partial}{\partial y_k^t} L(l, x) \end{aligned} \tag{4}$$

3 Consequences

1. Multi-path CTC can be implemented as a sum of simple CTC, though some work may be needed depending on what the CTC function returns as gradients and loss. This allows development to focus on other things and use efficient existing CTC implementations available as part of open source projects.

2. Loss gradients are effectively weighted by their likelihood, as per (4). This has an interesting consequences,

   (a) If the two paths are equally likely, we have a situation where the gradients are equally weighted and so the CTC is unable to make a distinction and maintains the status quo. As in point (b), unless the phone probabilities are exact, this position can't be sustained.

   (b) The CTC makes the more likely path even more likely as the weighted gradient drags it over to predict itself more. This is non-idempotent, which means that each training epoch will make the more likely paths even more likely, so even an initial mild imbalance will end up being overwhelmingly counted. **It is quite likely with a large number of epochs, that our branching path training is not actually resulting in a phone network that is allowing two different pronunciations and would be equivalent to choosing the more common pronunciation.**

      i. The above behavior can be seen as incorrect. Take situation where we have [1,2,4] or [1,3,4] labelling, our CTC function has absolutely no information on how to make a discernment between phones. However if [1,2,4] is more likely than [1,3,4] by a small

percentage, repeated training on this example will overwhelmingly train to identify only the 2 phone and not the 3 phone.

ii. It would be prudent to see how this works on actual training, can we train on American and Australian accents, does our phone network actually learn an accent?

iii. It would also be prudent to see what happens if we train a network with just a simple CTC path.

iv. The above issue may not be a big problem when seen in the context of a large amount of training data - we're getting our phone information from many different examples, so if we're surer about one path over another, we should make that path more likely. With a balanced dataset - each pronunciation is equally likely, the gradients should balance each other out overall. However, in an unbalanced dataset, we may end up converging to the more common pronunciation.

3. If we combine the losses as per a tensor/automatic differentiation style approach (PyTorch et al), we get the expected gradient. This was done experimentally, but the idea behind automatic differentiation makes this seem obvious

4 Further Thoughts

Consider a full training procedure, where we train our network on phones, single CTC words/phrases at a time. At this point our network has a good understanding of each phone. So imagine feeding a corpus of "tomato", where we have tom - a - to and tom - ah - to variants present.

1. Case 1, we assume that the a vs ah phone is close to 1, it's that confident about the different phones. In this case, what will the gradients be? In this situation, since the gradients are weighted by likelihood, the correct path is identified and the gradients should be mostly 0. Hence, our network should be stable, but this is expected since it's already converged on the correct phones - the fact that we are training isn't helpful at this point.

2. What about a situation where the a vs ah sound is not so clear, say 0.7 for the correct phone and 0.3 for the wrong phone. In this case, each training example will make the correct pronunciation more correct as we have discussed in 2(b). In the context of a batch, the gradients are now effectively weighted by the makeup of the pronunciations.

   (a) I.e. if the batch is evenly weighted 50-50 "ah" and 'a' then the gradients effectively cancel the other out and again, we learn nothing between the phones in the batch. In this case, I believe we would eventually lose out ability to identify between 'a' and 'ah', always putting out 0.5, 0.5 for the phones (or would it just learn nothing in this case?)

(b) If the batch is not evenly weighted. The pronunciation that is more common will dominate and our network will switch to predicting the more common pronunciation and the other phone would no longer be predicted at all.