Seq2Seq Losses (CTC)

Jerry Ding & Ryan Brigden

11-785 Recitation 6

February 23, 2018

Outline

Tasks suited for recurrent networks

Losses when the output is a sequence

Kinds of errors

Losses to use

CTC Loss Intuition

More formal description of CTC Loss

Computing CTC Loss

Visual representation

Tasks suited for recurrent networks

From the point of view of the input data

- When there's a clear notion of time
- When the lengths of the sequences are variable length
- When arbitrarily distant context information could be important.
- When the values are ordered but are not so continuous
- When it's natural to analyze an input in distinct steps, keeping track of some kind of state while performing the analysis

Digital logic analogy

MLP's $\rightarrow$ combinational logic
RNN's $\rightarrow$ Finite State Machine

Tasks suited for recurrent networks

From the point of view of the input data

- "Many to one" architecture
  - When there's a single classification to be made at the end of sequential input
- "Many to many (streaming)" architecture
  - Each output label is associated with a small segment of the input sequence
  - Outputs generated "on the fly" even without one-to-one correspondence from input to label
- "Generative" architecture
  - The output is itself a sequence, generated from the input sequence
  - Prior belief that the output sequence can only be built after seeing the entire input sequence

Tasks suited for recurrent networks

- **Example** "Many to one" architecture
  - Something like sentiment analysis (classify sentences as positive or negative)
- **Example** "Many to many (streaming)" architecture
  - Something like speech recognition (listen to audio and generate phonemes as you see them)
- **Example** "Generative" architecture
  - Something like machine translation (look at sentence and regenerate in a new language)
  - Alternative approach for the same task: attention networks

Losses when the output is a sequence

The losses we talked about so far compare one output vs one label

How to generalize this to multiple outputs?

Alice: Easy! Use the sum of the losses for each item in the output sequence.

Losses when the output is a sequence

The losses we talked about so far compare one output vs one label

How to generalize this to multiple outputs?

Alice: Easy! Use the sum of the losses for each item in the output sequence.
Bob: Wait. That will create bigger losses for longer output sequences
Alice: Uh... Use the mean!

Losses when the output is a sequence

The losses we talked about so far compare one output vs one label

How to generalize this to multiple outputs?

Alice: Uh... Use the mean!
Bob: Hm this just occurred to me, but what if the outputs and the labels don't even have the same length? How'd you match outputs to labels?
Alice: Try all pairs? Wait that makes no sense. Hm...

Kinds of errors

It's easy to tell if an output sequence is wrong, but how can we quantify exactly "how wrong" it is?

We need some kind of differentiable loss so we can use gradient descent.

The choice of loss function depends heavily on the nature of the errors we will get.

Kinds of errors

- "Many to one" architecture
  - Misclassification, low confidence in output, large root mean square error...etc
  - "5" vs "1", "0.526" vs "11.563"
- "Many to many (streaming)" architecture
  - Missing / extra labels, misclassifying the labels, wrong order of labels
  - "3 1 4 1 9 2 6" vs. "3 1 4 1 5 9 2 6", "A B C D" vs. "Z B D C"
- "Generative" architecture
  - Output is similar overall to the ground truth, but one important value was wrong
    - "The cat sat on the mat" vs "The car sat on the mat"
  - The output has bad style or grammar
  - The output contains some kind of noise that makes it clearly feel artificial
    - "so much depends on, a red wheel. Barrow is glazed with heavy rain poured beside the white chickens."

Losses to use

- "Single output" architecture:
  - Cross entropy, L2
- "Multiple outputs before the end of the input" architecture
  - CTC loss functions
- "Multiple outputs after the whole input" architecture
  - Still is kind of a hard problem...
  - Goal: Avoid being a bad teacher that reads off the answer sheet and grades everything as if there's "only one correct answer."
  - Rather, consider "edit distance", the minimum number of changes (insertions, substitutions, deletions) required to reconstruct the ground truth from the output.
  - Embedding losses, discriminator networks (the A in GAN's), n-gram losses

CTC Loss Intuition

1. Add a blank symbol to the set of possible labels.
2. Design the network to output a classification at each input value.
3. "Collapse" repeated outputs and blanks
   - AAAAA $\rightarrow$ A
   - AAAABB $\rightarrow$ AB
   - AAAA___BB $\rightarrow$ AB
   - AAA__AA $\rightarrow$ AA (blanks separate outputs of the same symbol)
4. Compare collapsed output with the ground truth label

CTC Loss Intuition

Input

Network outputs before training

_CA_AB_C_BB_AC_A_B_AA_CC_

Collapsed

CAABCBACABAC

Ground truth

ABCC

CTC Loss Intuition

Input

Network outputs after training

____AAAA__BBBCC___CCCC

Collapsed

ABCC

Ground truth

ABCC

Probabilistic values

At each time step, the network actually outputs a probability distribution over all the possible labels + the blank symbol.

If we randomly select a label independently from each of these distributions, what is the probability that we will get an output sequence that collapses into the ground truth sequence?

The CTC loss is the negative logarithm of this probability.

More formal description of CTC Loss

Let L be the set of labels and L' be the set of labels with the "blank" label.

For a sequence of length T, we denote the set of possible paths ${L'}^T = \pi$

Given a sequence of inputs $\mathbf{x}$ and labelling $\mathbf{z}$, where $|\mathbf{z}| \leq |\mathbf{x}|$, try to maximize the probability of the labelling given the sequence (maximum likelihood estimate).

$$\hat{\theta} = \arg\max_\theta \prod_{i=1}^{N} p(\mathbf{z}^{(i)}|\mathbf{x}^{(i)}; \theta)$$

More formal description of CTC Loss

Define the many-to-one map $\mathbf{B} : {L'}^T \rightarrow L^{\leq T}$ that maps length T label sequence of alphabet L' to their labelling equivalent in L, while removing all blanks and repeated labels. In effect, $\mathbf{B}$ performs the "collapsing" operation discussed previously. For example:

$$\mathbf{B}(\_\_AAAA\_BBBCC\_\_CCC) = \mathbf{B}(A\_\_BBBB\_CC) = ABC$$

Define $\mathbf{B}^{-1}$ to map a label sequence $\mathbf{z}$ to the set of all possible label sequences (paths in) that collapse to $\mathbf{z}$. So $\{\mathbf{B}(x)|x \in \mathbf{B}^{-1}(y)\} = y$.

Therefore, we can consider the likelihood of a given labelling $\mathbf{z}$ as the sum of the probabilities of all the paths that can collapse to $\mathbf{z}$.

$$p(\mathbf{z}|\mathbf{x};\theta) = \sum_{\pi \in \mathbf{B}^{-1}(\mathbf{z})} p(\pi|\mathbf{x};\theta)$$

More formal description of CTC Loss

Our objective is now more clear and we can plug in our formulation of the likelihood of a labelling. The objective of the argmin is the CTC loss.

$$\hat{\theta} = \arg\max_{\theta} \prod_{i=1}^{N} \sum_{\pi \in \mathbf{B}^{-1}(\mathbf{z}^{(i)})} p(\pi | \mathbf{x}^{(i)}; \theta)$$

$$= \arg\min_{\theta} \; - \prod_{i=1}^{N} \sum_{\pi \in \mathbf{B}^{-1}(\mathbf{z}^{(i)})} p(\pi | \mathbf{x}^{(i)}; \theta)$$

Computing the probability

Naive method

Generate all possible sequences with their probabilities, collapse each sequence, sum the probabilities of those that collapsed to the correct value.

Let N be the number of inputs, M be the number of labels, C be the number of classes that can be output at each input (including the blank symbol).

The computational complexity is $O(C^N N)$

Computing the probability

Better method

We can derive a recursive formula for the probability and exploit the existence of common sub-paths (think dynamic programming).

Let's create some mathematical notation, and a visual representation of it.

First, we modify the sequence of labels by adding blanks between each symbol, and at the beginning and the end.

$$\text{ABCC} \rightarrow \text{\_A\_B\_C\_C\_}$$

Let M' be the length of the modified sequence of labels,

$$M' = 2M + 1$$

Computing the probability

Better method

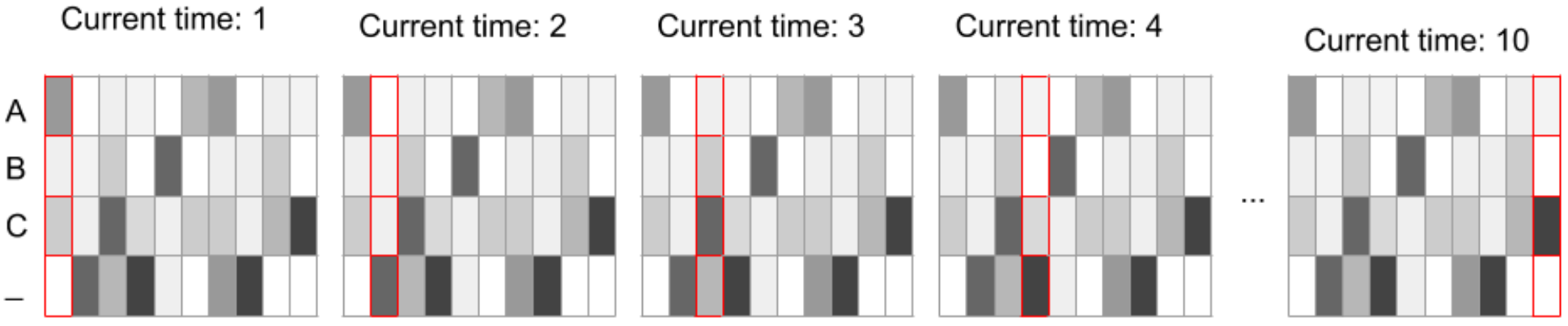
Imagine that the sequence of labels correspond to hidden states. The network transitions through these hidden states while it's generating the outputs.

_ : Network is trying to generate blank symbols
A : Network is trying to generate A's
Ditto for states B and C.

We're allowed to skip blank states, but only if the next state is different.

E.g. starting at state C, we can't skip state _ to enter state C again

Visual representation



Generating: _

_A_B_C_C_

Example path:

_…

_

Generating: _

_A_B_C_C_

Example path:

_…

__

Generating: A

_A_B_C_C_

Example path:

_…

__A

Generating: B

_A_B_C_C_

Example path:

_…

__AB

Generating: C

_A_B_C_C_

Example path:

__AB_C_CC

Mathematical formulation

Let t be an index into the network output sequence ($1 \leq t \leq N$)
Let s be an index into the sequence of labels ($1 \leq s \leq M'$)

$\alpha_t(s)$: Sum of the probabilities of all forward paths from time 1 to t, that have transitioned through states 1 to s.

$\beta_t(s)$: Sum of the probabilities of all backward paths from time N to t, that have transitioned through states M' to s.

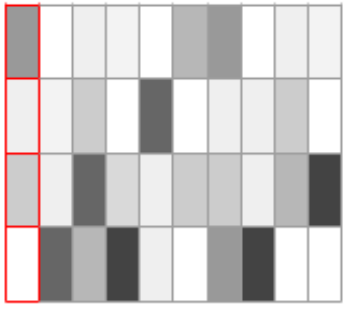
For now, we focus on $\alpha_t(s)$. It's sufficient for computing the CTC loss.

$\beta_t(s)$ is only needed to compute the gradient of the CTC loss.

Forward probabilities

What is $\alpha_1(1)$?

Current time: 1



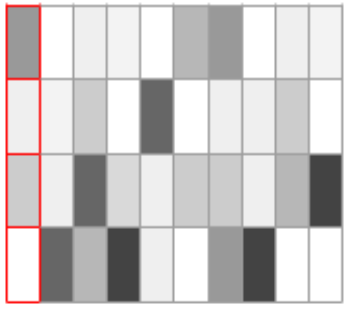
Generating: _

_A_B_C_C_

$\alpha_1(1)$: _

What is $\alpha_1(2)$?

Current time: 1



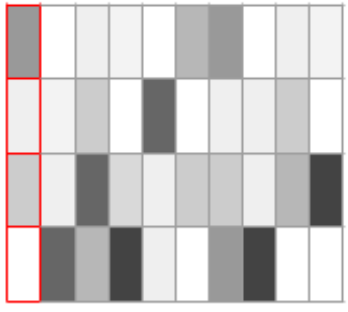
Generating: A

_A_B_C_C_

$\alpha_1(2)$: _

What is $\alpha_1(3)$

Current time: 1



Generating: _

_A_B_C_C_

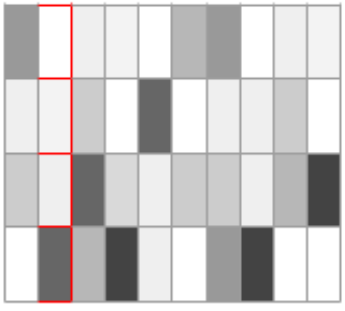
$\alpha_1(3)$: 0

Because we can't skip the A state.

Forward probabilities

What is $\alpha_2(1)$?

Current time: 2



Generating: _

_A_B_C_C_

$\alpha_2(1)$: _

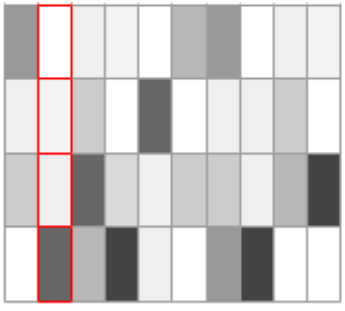
There's one valid path up to $t = 2$, ending on the first _ state.

It's the one that simply outputted two _ symbols.

Forward probabilities

What is $\alpha_2(2)$?

Current time: 2



Generating: A

_A_B_C_C_

$\alpha_2(2)$: __ + __

There are two valid paths up to $t = 2$ that end on the A state.

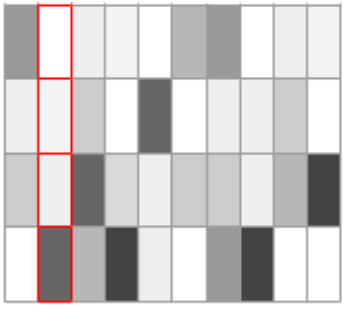
Path 1 is where we output AA

Path 2 is where we output _A

Forward probabilities

What is $\alpha_2(3)$?

Current time: 2



Generating: _

_A_B_C_C_

$\alpha_2(3)$:

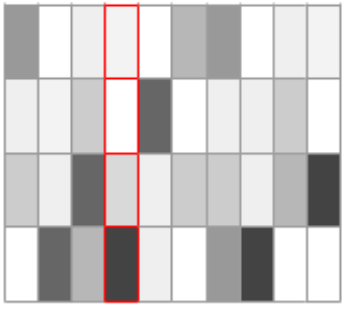
There is one path up to $t = 2$ that ends on the second _ state.

It's the one where we output A_

Forward probabilities

What is $\alpha_4(4)$?

Current time: 4



Generating: B

_A_B_C_C_

$\alpha_4(6)$: Big sum

10 paths up to $t = 4$ that end on the B state:

__AB _A_B _AAB _ABB A__B A_BB AA_B

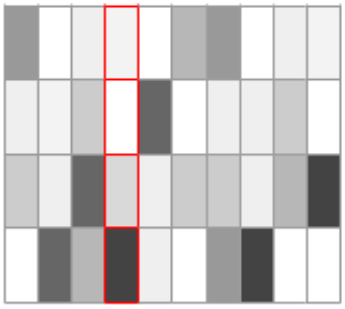
AAAB AABB ABBB

Note: these paths can't end in the _ symbol, because we aren't supposed to reach the 3rd _ state yet.

Forward probabilities

What is $\alpha_4(5)$?

Current time: 4



Generating: _

_A_B_C_C_

$\alpha_4(6)$: Big sum

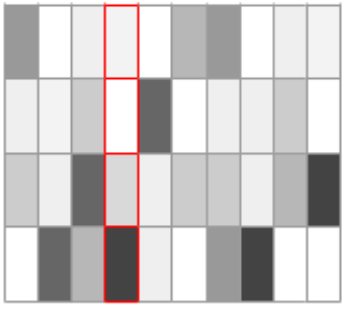
5 paths up to $t = 4$ that end on the third _ state:

_AB_ A_B_ AAB_ AB__ ABB_

Forward probabilities

What is $\alpha_4(6)$?

Current time: 4



Generating: C

_A_B_C_C_

$\alpha_4(6)$: Big sum

6 paths up to $t = 4$ that end on the first C state:

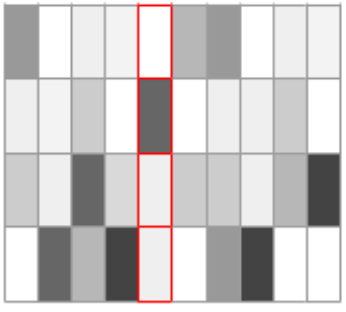
_ABC A_BC AABC AB_C ABBC ABCC

Note: these paths can't end in the _ symbol, because we aren't supposed to reach the 4th _ state yet.

Forward probabilities

What is $\alpha_5(6)$?

Current time: 5



Generating: C

_A_B_C_C_

$\alpha_5(6)$: Big sum

21 paths up to $t = 5$ that ends on the first C state:

__ABC	_A_BC	_AABC	_AB_C	_ABBC	_ABCC	A__BC
A_B_C	A_BBC	A_BCC	AA_BC	AAABC	AAB_C	AABBC
AABCC	AB__C	AB_CC	ABB_C	ABBBC	ABBCC	ABCCC

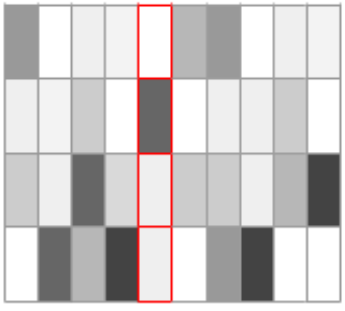
Is there a an easier way to sum the probabilities of the paths?

Hint: 10 + 5 + 6 = 21

Forward recurrence relation

What is $\alpha_5(6)$?

Current time: 5



Generating: C

_A_B_C_C_

$\alpha_5(6)$:

$\_(\alpha_5(6) + \alpha_5(6) + \alpha_5(6))$

Let $p_5(C)$ be the probability mass that the output has assigned to label C at $t = 5$. In the picture, the relevant cell is colored ___.

We have several choices for how to reach the first C state.

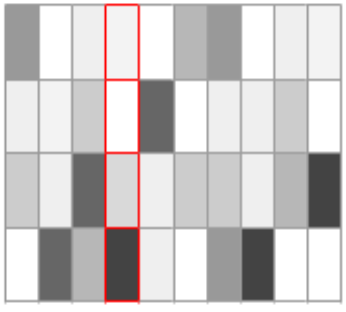
1. Reach state B at $t = 4$, and skip over state _ to state C.
2. Reach state _ at $t = 4$, and move to state C.
3. Reach state C at $t = 4$, and stay in state C.

In all cases we need to output an extra C symbol at $t = 5$.

$\therefore \alpha_5(6) = p_5(C)\,(\alpha_4(4) + \alpha_4(5) + \alpha_4(6))$

Forward recurrence relation

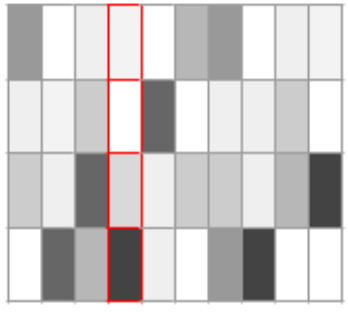
Current time: 4



Generating: B

_A_B_C_C_

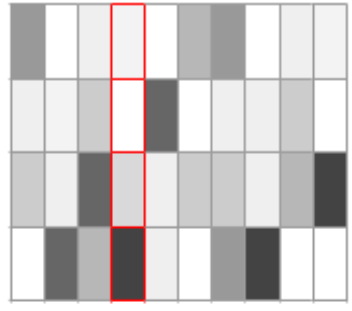
Current time: 4



Generating: _

_A_B_C_C_

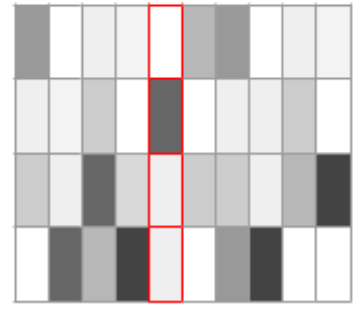
Current time: 4



Generating: C

_A_B_C_C_

Current time: 5



Generating: C

_A_B_C_C_

Forward probability pseudocode

Algorithm 1: Forward algorithm

Data: If $x \leq 1$, `labels[x]` is *Undefined*, define
$p_t(\textit{Undefined}) = 0$.

$\alpha_1(1) = p_1(\texttt{labels[1]})$;

$\alpha_1(2) = p_1(\texttt{labels[2]})$;

for $s = 3$ **to** M' **do**

  $\alpha_1(s) = 0$;

for $t = 2$ **to** N **do**

  for $s = 3$ **to** M' **do**

    if *`labels[s]` is not the same as `labels[s-2]`* **then**

      # We could skip `labels[s - 1]` $\alpha_t(s) =$
      $p_t(\texttt{labels[s]})(\alpha_{t-1}(s-2) + \alpha_{t-1}(s-1) + \alpha_{t-1}(s))$;

    else

      # This means we can't skip labels[s - 1]
      $\alpha_t(s) = p_t(\texttt{labels[s]})(\alpha_{t-1}(s-1) + \alpha_{t-1}(s))$

CTC loss value

Paths that collapse into the ground truth sequence must end in either

- The final _ state, or
- The final non-blank state.

The sum of the probability of the valid paths is $a_N(M')$ in the first case and $\alpha_N(M'-1)$ in the second case.

$$\mathcal{L}_{\text{CTC}} = -\log(a_N(M') + a_N(M'-1))$$

CTC loss gradient

How much will the CTC loss change with a small change in one of the output probabilities?

Suppose we want to calculate

$$\frac{d(\mathcal{L}_{\text{CTC}})}{d(p_t(x))}$$

Note that $\alpha_t(s) = p_t(\texttt{labels[s]})K$

Where K is the weighted sum of $a_?(?)$ terms for past time steps. K is constant w.r.t. $p_t(\texttt{labels[s]})$.

$$\frac{d(\alpha_t(s))}{d(p_t(\texttt{labels[s]}))} = K = \frac{\alpha_t(s)}{p_t(\texttt{labels[s]}}$$

Easy to compute! Can we do something similar for the CTC loss?

CTC loss gradient

Recall

$\beta_t(s)$: Sum of the probabilities of all backward paths from time N to t, that have transitioned through states M' to s.

We can define backward probabilities $\beta_t(s)$ in a way very similar to $\alpha_t(s)$.

Backward probability pseudocode

Algorithm 2: Forward algorithm

Data: If $x < 1$, `labels[x]` is *Undefined*, define
$p_t(\textit{Undefined}) = 0$.

$\beta_1(1) = p_N(\texttt{labels[M']})$;

$\beta_1(2) = p_N(\texttt{labels[M'-1]})$;

for $s = 1$ **to** $M' - 2$ **do**

  $\alpha_1(s) = 0$;

for $t = N$ **to** 1 **do**

  for $s = 1$ **to** M' **do**

    if *`labels[s]` is not the same as `labels[s+2]`* **then**

      # We could skip `labels[s + 1]` $\beta_t(s) =$
      $p_t(\texttt{labels[s]})(\beta_{t+1}(s+2) + \beta_{t+1}(s+1) + \beta_{t+1}(s))$;

    else

      # This means we can't skip labels[s + 1]
      $\beta_t(s) = p_t(\texttt{labels[s]})(\beta_{t+1}(s+1) + \beta_{t+1}(s))$

CTC loss gradient

We end up with nice properties relating $\alpha_t(s)$, $\beta_t(s)$, and the CTC loss. For example:

$$\mathcal{L}_{\text{CTC}} = -\log\left(\sum_s \frac{\alpha_t(s)\beta_t(s)}{p_t(\texttt{labels[s]})}\right)$$

Also,

$$\beta_t(s) = p_t(\texttt{labels[s]})L$$

Where L is the weighted sum of $b_?(?)$ terms for future time steps. L is also constant w.r.t. $p_t(\texttt{labels[s]})$. This means:

$$\frac{d(\beta_t(s))}{d(p_t(\texttt{labels[s]}))} = L = \frac{\beta_t(s)}{p_t(\texttt{labels[s]})}$$

CTC loss gradient

Therefore,

$$\frac{d\mathcal{L}_{\texttt{CTC}}}{d(p_t(x))} = [\texttt{some simple calculus...}]$$

$$= \frac{-1}{\mathcal{L}_{\texttt{CTC}}} \left[\sum_{s|\texttt{labels[s]}=x} \frac{\alpha_t(s)\beta_t(s)}{p_t^2(labels[s])} \right)$$

Inference

What sequence of labels do we output at test time?

When we used the "many-to-one" architecture, this was easy. For classification tasks, we picked the argmax of the network outputs. For regression, we just returned the network output unchanged.

For CTC, it's less obvious what to do. We want to output the most likely sequence of "collapsed" labels. But a large number of paths can collapse into that sequence of labels.

- We can generate the most likely path, and collapse that. This is a greedy approach though, and doesn't guarantee that we get the most likely sequence of labels.
- We can run a beam search, a method where we keep track of multiple possible best paths as we iterate over the network's output probabilities to reduce the greediness of our search.
- We can use a similar dynamic programming algorithm to compute the true answer.

Practical Concerns

1. Both $\alpha_t(s)$ and $\beta_t(s)$ can be very close to 0 during computation. Close enough to risk floating point underflow!
   - Solution: only compute $\log(\alpha_t(s))$ and $\log(\beta_t(s))$. Replace all instances of $\times$ with $+$, and all instances of $+$ with $\texttt{LogSumExp}(\cdot, \cdot)$
2. CTC loss can still be computationally costly without optimization. Can even dominate over the forward and backward passes if you're not careful.
   - Solution: use a library like Sphinx, code in C or C++, or vectorize the code

Variants of CTC

CTC actually describes a family of loss functions. The formulation with blanks is just the original example. Other examples:

- If you can guarantee that the same label cannot occur twice in succession, you can create a version of CTC without blanks.
- There are times when there are multiple types of blanks, perhaps one for each possible non-blank symbol.
- Can even follow certain finite state machines. For instance, the formulation in HW1 where each phoneme contains three sections that must occur in some order. You can create a CTC that only considers paths that outputs symbols in the right order.

Variants of CTC

- In any case, you'd still use the forward-backward algorithm to efficiently compute the CTC loss value / gradients. The exact same practical concerns apply.
- The only difference is the recurrence relation used to define $\alpha_t(s)$ and $\beta_t(s)$.
- Given $\alpha_t(s)$ and $\beta_t(s)$, the same formula can be used to compute the CTC loss and gradients. Many frameworks let you define this recurrence relation.