Lecture 20: Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks

Mark Hasegawa-Johnson
All content CC-BY 4.0 unless otherwise specified.

ECE 537, Fall 2022
Temporal Classification

- \( \mathbf{x} = [x_1, \ldots, x_T] \) is the input. Each \( x_t \) is usually a vector, 
  \( x_t = [x^t_1, \ldots, x^t_m] \).
- \( \mathbf{z} = [z_1, \ldots, z_U] \) is the desired network output, where \( z_u \in L \) 
  comes from some alphabet \( L \). \( U \leq T \).
- The goal is to train a function \( h(\mathbf{x}) \) so that \( y = h(\mathbf{x}) \) is similar 
  to \( \mathbf{z} \).
Temporal classification maps from a sequence of speech frames (top) to a sequence of phoneme or character labels (bottom).

Graves et al., 2006, Figure 1. (c) ICML
An RNN outputs a sequence of vectors, $y = [y_1, \ldots, y_T]$, where each $y_t = [y^t_1, \ldots, y^t_{|L|}]$ is a pmf:

$$y^t_k \geq 0, \quad \sum_{k=1}^{|L|} y^t_k = 1$$

Thus, if $z$ is time-aligned, we can interpret

$$y^t_k = P(z_t = k | x)$$
Recurrent Neural Net (RNN)

Image CC-SA-4.0 by Ixnay,

https://commons.wikimedia.org/wiki/File:Recurrent_neural_network_unfold.svg
A recurrent neural net defines nonlinear recurrence of a hidden vector, $h_t$:

$$h_t = \sigma (Ux_t + Vh_{t-1})$$

$$y_t = \text{softmax} (W h_t)$$

The weight matrices, $U$, $V$, and $W$, are chosen to minimize the loss function. For example, suppose we’re using a cross-entropy loss with target sequence $z$, then

$$\mathcal{L} = - \sum_{t=1}^{T} \ln y_{z_t}^t$$
Partial vs. Full Derivatives

With one-step recurrence, as shown here, $\mathcal{L}$ depends on $h_t$ in exactly two different ways:

$$\frac{d\mathcal{L}}{dh_t} = \frac{d\mathcal{L}}{dy_t} \frac{\partial y_t}{\partial h_t} + \frac{d\mathcal{L}}{dh_{t+1}} \frac{\partial h_{t+1}}{\partial h_t}$$
Partial vs. Full Derivatives

\[
\frac{dL}{dh_t} = \frac{dL}{dy_t} \frac{\partial y_t}{\partial h_t} + \frac{dL}{dh_{t+1}} \frac{\partial h_{t+1}}{\partial h_t}
\]

where

- \( \frac{dL}{dh_t} \) is the total derivative, and includes all of the different ways in which \( L \) depends on \( h_t \).
- \( \frac{\partial h_{t+1}}{\partial h_t} \) is the partial derivative, i.e., the change in \( h_{t+1} \) per unit change in \( h_t \) if \( x_t \) is held constant.
Here’s a flow diagram that could represent:

\[
h_t = \sigma(Ux_t + Vh_{t-1}), \quad y_t = \text{softmax}(Wh_t),
\]

\[
\mathcal{L} = -\sum_{t=0}^{T} \ln y_{zt}^t
\]
Back-propagation through time does this:

\[
\frac{d\mathcal{L}}{dh_t} = \frac{d\mathcal{L}}{dy_t} \frac{\partial y_t}{\partial h_t} + \frac{d\mathcal{L}}{dh_{t+1}} \frac{\partial h_{t+1}}{\partial h_t}
\]
Partial vs. Full Derivatives

So for example, if

\[ \mathcal{L} = -\sum_{t=0}^{T} \ln y_{zt} \]

then the partial derivative of \( \mathcal{L} \) w.r.t. \( h_k^t \) is

\[ \frac{\partial \mathcal{L}}{\partial h_k^t} = -\frac{1}{y_{zt}} \frac{\partial y_{zt}^t}{\partial h_k^t} \]

and the total derivative of \( \mathcal{L} \) w.r.t. \( h_k^t \) is

\[ \frac{d\mathcal{L}}{dh_k^t} = -\frac{1}{y_{zt}} \frac{\partial y_{zt}^t}{\partial h_k^t} + \sum_i \frac{d\mathcal{L}}{dh_i^{t+1}} \frac{\partial h_i^{t+1}}{\partial h_k^t} \]
The basic idea of back-prop-through-time is divide-and-conquer.

1. **Synchronous Backprop**: First, calculate the **partial derivative** of $L$ w.r.t. $h^t_k$, assuming that all other time steps are held constant.

   $$\frac{\partial L}{\partial h^t_k} = -\frac{1}{y^t_{zt}} \frac{\partial y^t_{zt}}{\partial h^t_k}$$

2. **Back-prop through time**: Second, iterate backward through time to calculate the **total derivative**

   $$\frac{dL}{dh^t_k} = -\frac{1}{y^t_{zt}} \frac{\partial y^t_{zt}}{\partial h^t_k} + \sum_i \frac{dL}{dh^{t+1}_i} \frac{\partial h^{t+1}_i}{\partial h^t_k}$$
In the previous slides, notice we’ve assumed that the correct labeling, \( z \), is time-aligned to the speech waveform, i.e., \( z = [z_1, \ldots, z_T] \).

That’s rarely true! Usually we know the correct phones or characters, \( z = [z_1, \ldots, z_U] \), but not their time alignment, i.e., \( U \leq T \).

The old solution (pre-CTC):
- Train a mixture Gaussian HMM.
- Use the Viterbi algorithm to time-align \( z \) to \( x \).
- Use the time-aligned \( z \) to train the RNN.

CTC was proposed as a better way.
Outline

1. Temporal Classification
2. Recurrent Neural Networks
3. From Network Outputs to Labellings
4. The CTC Forward Algorithm
5. Conclusions
Many-to-One Mapping

The key idea of CTC is that, since $U \leq T$, the mapping from $y$ to $z$ is many-to-one. For example, consider an utterance with a 5-frame speech file, and a 3-character output. We can map from 5 frames to 3 characters by just eliminating sequential duplicates, like this:

$$y = [f, f, e, d, d] \quad \rightarrow \quad \mathcal{B} \quad \rightarrow \quad z = [f, e, d]$$

$$y = [f, e, e, e, d] \quad \rightarrow \quad \mathcal{B} \quad \rightarrow \quad z = [f, e, d]$$

$$y = [f, f, e, e, d] \quad \rightarrow \quad \mathcal{B} \quad \rightarrow \quad z = [f, e, d]$$

But notice the problem: there is no way to generate the output $z = [f, e, e, d]$! By eliminating duplicates, it becomes impossible to generate a sentence with repeated letters.
CTC makes repeated letters possible by using a blank character, –. The many-to-one mapping now has two steps: (1) eliminate all duplicate characters, (2) THEN eliminate all blanks.

\[ y = [f, -, e, e, d] \quad \rightarrow \quad B \quad \rightarrow \quad z = [f, e, d] \]

\[ y = [f, e, -, e, d] \quad \rightarrow \quad B \quad \rightarrow \quad z = [f, e, e, d] \]

\[ y = [f, -, f, e, d] \quad \rightarrow \quad B \quad \rightarrow \quad z = [f, f, e, d] \]
With these definitions, the probability of $z$ given $x$ is:

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

- $\pi = [\pi_1, \ldots, \pi_T]$ is a time-aligned label sequence called a “path.” Each path element is a label or a blank: $\pi_t \in L \cup \{-\}$.

$$p(\pi|x) = \prod_{t=1}^{T} y_{\pi_t}^t$$

- $B^{-1}(z)$ is the set of all paths that match the label sequence $z$.

For example,

$$B^{-1}([f, e, d]) = \begin{cases} [f, e, e, e, d] \\ [f, -, e, -, d] \\ [f, f, -, e, d] \end{cases}$$
The temporal classification problem is now just:

\[
h(x) = \arg\max_{l \in L \leq T} p(l|x)
\]

\[
= \arg\max_{l \in L \leq T} \sum_{\pi \in B^{-1}(l)} p(\pi|x)
\]

\[
= \arg\max_{l \in L \leq T} \sum_{\pi \in B^{-1}(l)} \prod_{t=1}^{T} y_{\pi t}^t
\]

- \( l = [l_1, \ldots, l_V] \) is a label sequence of any length \( V \leq T \) where \( l_v \in L \).
- \( \pi = [\pi_1, \ldots, \pi_T] \) is a path of length \( T \) where \( \pi_t \in L \cup \{-\} \).
1. Temporal Classification

2. Recurrent Neural Networks

3. From Network Outputs to Labellings

4. The CTC Forward Algorithm

5. Conclusions
The problem now is: how can we search the entire set $\pi \in \mathcal{B}^{-1}(\mathbf{l})$, for every possible label sequence?

Answer: the forward algorithm!
In order to express the CTC forward algorithm, we need to define a modified label sequence, $l'$. $l'$ is equal to $l$ with blanks inserted between every pair of letters. Thus if

$$l = [f, e, d],$$

then

$$l' = [-, f, -, e, -, d, -].$$

If the length of $l$ is $|l|$, then the length of $l'$ is $2|l| + 1$. 

We also need to define the following partial sequences:

\[ x_{1:t} = [x_1, \ldots, x_t] \]
\[ \pi_{1:t} = [\pi_1, \ldots, \pi_t] \]
\[ l'_{1:s} = [l'_1, \ldots, l'_s] \]

\[ = \begin{cases} 
[-, l_1, -], l_2, \ldots, l_{s/2}] & s \text{ even} \\
[-, l_1, -], l_2, \ldots, l_{(s-1)/2}, -] & s \text{ odd}
\end{cases} \]
The CTC Forward Algorithm

Definition: $\alpha_t(l'_1:s) \equiv p(l'_1:s|x_{1:t})$. Computation:

1. Initialize:

   $$\alpha_1([-]) = y_1$$
   $$\alpha_1([-, l_1]) = y_{l_1}$$

The neural network can either start out by generating a blank, in which case $l' = [-]$, or it can start out by generating a real character $l_1$, in which case $l' = [-, l_1]$. 
The CTC Forward Algorithm

1 Initialize:

\[ \alpha_1([-]) = y^-_1 \]
\[ \alpha_1([-, l_1]) = y^1_{l_1} \]

2 Iterate:

\[ \alpha_t(l'_1:s) = \begin{cases} 
(\alpha_{t-1}(l'_1:s) + \alpha_{t-1}(l'_1:s-1)) \times y^t_{l'_s} \\
\ldots \ldots \ldots \text{if } l'_s = - \text{ or } l'_s = l'_s-2 \\
(\alpha_{t-1}(l'_1:s) + \alpha_{t-1}(l'_1:s-1) + \alpha_{t-1}(l'_1:s-2)) \times y^t_{l'_s} \\
\ldots \ldots \ldots \text{otherwise}
\end{cases} \]

Repeating the same character \((\alpha_{t-1}(l'_1:s))\) or adding one more character \((\alpha_{t-1}(l'_1:s-1))\) are always possible. Adding two more characters \((\alpha_{t-1}(l'_1:s-2))\) is OK if the current character is not a blank or a repeat.
Repeating the same character ($\alpha_{t-1}(l'_{1:s})$) or adding one more character ($\alpha_{t-1}(l'_{1:s-1})$) are always possible. Adding two more characters ($\alpha_{t-1}(l'_{1:s-2})$) is OK if the current character is not a blank or a repeat.
The CTC Forward Algorithm

1 Initialize:

\[ \alpha_1([-]) = y_1 \]
\[ \alpha_1([-, l_1]) = y_{l_1}^1 \]

2 Iterate:

\[ \alpha_t(l'_1:s) = \begin{cases} (\alpha_{t-1}(l'_1:s) + \alpha_{t-1}(l'_1:s-1)) \times y_{l_s}^t \\ \text{.............. if } l'_s = - \text{ or } l'_s = l'_{s-2} \\ (\alpha_{t-1}(l'_1:s) + \alpha_{t-1}(l'_1:s-1) + \alpha_{t-1}(l'_1:s-2)) \times y_{l_s}^{t'} \\ \text{.............. otherwise} \end{cases} \]

3 Terminate:

\[ p(l_{1:U}|x) = \alpha_T(l'_{1:2U}) + \alpha_T(l'_{1:2U+1}) \]
The CTC Forward Algorithm

\[ p(l_{1:U} | x_{1:T}) = \alpha_T(l'_{1:2U}) + \alpha_T(l'_{1:2U+1}) \]
In the HMM, $\alpha_t(q)$ only depended on the final state. In CTC, $\alpha_t(l'_1:s)$ depends on the entire label sequence up to position $s$. The complexity of this search is exponential in $s$. To make it computationally tractable, use a beam search:
- Discard all but the best $N$ candidates for each $t$.
- Compute all possible extensions to time $t + 1$.
- Repeat
- As in the HMM, $\alpha_t(l_1:s)$ becomes very small, so Graves et al. recommend using a scaled forward algorithm.
Computational Issues #1: Beam Search

- $\alpha_t(l'_{1:s})$ depends on the entire label sequence up to position $s$. The complexity of this search is exponential in $s$ (shown).
- To make it computationally tractable, use a beam search (not shown).

Graves et al., 2006, Fig. 2. (c) ICML
An RNN computes the probability of a label sequence, $z$, given an input sequence $x$.

The key idea of CTC is a many-to-one mapping from paths to label sequences. The recognition probability is then

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

The CTC forward algorithm is just like the HMM forward algorithm, except that, instead of $\alpha_t(q)$, we compute

$$\alpha_t(l'_1:s) \equiv p(l'_1:s|x_{1:t})$$